บทที่ 1 ทำความรู้จักกับภาษา Python

ภาษาไพทอน (Python programming language)

ภาษาไพทอน (Python programming language) เป็นภาษาโปรแกรมภาษาระดับสูงแบบอินเทอร์พรีเตอร์ (Interperter) ที่สร้างโดย กีโด ฟาน รอสซัม (Guido van Rossum) ในปีพ.ศ. 2533 ปัจจุบันดูแลโดย มูลนิธิซอฟต์แวร์ไพทอน (Python Software Foundation (PSF)) ซึ่งภาษาไพทอนสามารถใช้งานได้บนระบบปฏิบัติการไม่ว่าจะเป็น Unix, Linux, Windows NT, Windows 2000, Windows 95/98/MEXP หรือ OS/X โดย Version ล่าสุดของ Python ตอนนี้คือ 3.6.5

จุดเด่นของภาษาไพทอน

ไพทอนเป็นภาษาสคริปต์ ทำให้ใช้เวลาในการเขียนและคอมไพล์ไม่มาก ทำให้เหมาะกับงานด้านการดูแลระบบ (System administration) เป็นอย่างยิ่ง ได้มีการสนับสนุนภาษาไพทอนโดยเป็นส่วนหนึ่งของระบบปฏิบัติการยูนิกซ์, ลินุกซ์ และสามารถติดตั้งให้ทำงานเป็นภาษาสคริปต์ของวินโดวส์ ผ่านระบบ Windows Script Host ได้อีกด้วย และ Python เองก็ได้ถูกนำมาพัฒนา Web application อย่างแพร่หลาย ซึ่งมี Framework สำหรับทำเว็บของ Python ที่ได้รับความนิยมอย่างมากคือ Django

ไวยากรณ์อ่านง่าย

ภาษาไพทอนจะคล้ายกับภาษา C มาก โดยจะมีโครงสร้างที่ไม่ซับซ้อน มีไวยากรณ์ที่ได้กำจัดการใช้สัญลักษณ์ที่ใช้ในการแบ่งบล็อกของโปรแกรม และใช้การย่อหน้าแทน ทำให้สามารถอ่านโปรแกรมที่เขียนได้ง่าย นอกจากนั้นยังมีการสนับสนุนการเขียน docstring ซึ่งเป็นข้อความสั้นๆ ที่ใช้อธิบายการทำงานของฟังก์ชัน, คลาส, และโมดูลอีกด้วย

ความเป็นภาษากาว

ไพทอนเป็นภาษากาว (Glue Language) ได้อย่างดีเนื่องจากสามารถเรียกใช้ภาษาโปรแกรมอื่นๆ ได้หลายภาษา ทำให้เหมาะที่จะใช้เขียนเพื่อประสานงานโปรแกรมที่เขียนในภาษาต่างกันได้

ไลบรารีในไพทอน

การเขียนโปรแกรมในภาษาไพทอนโดยใช้ไลบรารีต่าง ๆ เป็นการลดภาระของโปรแกรมเมอร์ได้เป็นอย่างดี ทำให้โปรแกรมเมอร์ไม่ต้องเสียเวลากับการเขียนคำสั่งที่ซ้ำๆ เช่นการแสดงผลข้อมูลออกสู่หน้าจอ หรือการรับค่าต่าง ๆ ไพทอนมีชุดไลบรารีมาตรฐานมาให้ตั้งแต่ติดตั้งอินเตอร์พรีเตอร์ นอกจากนั้นยังมีผู้พัฒนาจากทั่วโลกดำเนินการพัฒนาไลบรารีซึ่งช่วยอำนวยความสะดวกในด้านต่าง ๆ โดยจะเผยแพร่ในรูปแบบของแพ็คเกจต่าง ๆ ซึ่งสามารถติดตั้งเพิ่มเติมได้อีกด้วย สุดท้ายคือ ภาษาไพทอน ทำงานเร็วที่สุดเมื่อเทียบกับภาษา script ด้วยกัน เช่น php, jsp, asp จะพูดว่า ไพทอน เขียนน้อยได้งานมาก ทำงานเร็วก็ไม่ผิดนัก

บทที่ 2 โครงสร้างและลักษณะการเขียนโปรแกรม

ภาษา python จัดเป็นภาษาที่อยู่ในระดับสูงเทียบกับภาษา Visual Basic บางกรณีที่ต้องการใช้งานเชิงลึกในระดับ Kernel ของระบบปฏิบัติการ การเรียกใช้พอร์ตฮาร์ดแวร์ต่างๆ การจัดการระดับหน่วยความจำ การเขียนโปรแกรมระบบเครือข่ายเชิงลึก ตามที่กล่าวไว้บางส่วน ภาษา python ไม่มีความสามารถเข้าไปจัดการได้เลย ต้องอาศัยภาษาอื่นๆ ที่มีความสามารถในการเขียนโปรแกรมเชิงลึก เช่น C, C + +, Java ฯลฯ สร้างเป็นโมดูลพิเศษไว้เชื่อมต่อกับ python ที่จะเรียกใช้ในลักษณะเป็นฟังก์ชันย่อยภายในของโมดูลพิเศษเหล่านั้นอีกที

คำสั่งที่เราใช้ในการอ้างอิงโมดูลมาตรฐาน หรือโมดูลเพิ่มเติมพิเศษใน python ประกอบด้วย 2 คำสั่งดังนี้

1. คำสั่ง import
2. คำสั่ง from

คำสั่ง import

ตัวอย่างที่ 1 การเขียนคำสั่งอ้างอิงโมดูลที่ 1

```
Import      datetime
```

ตัวอย่างที่ 2 การเขียนคำสั่งอ้างอิงโมดูลมากกว่า 1 โมดูล

```
Import      datetime
Import      decimal
Import      os
```

ตัวอย่างที่ 3 การเขียนคำสั่งอ้างอิงโมดูลมากกว่า 1 โมดูล แบบใช้คำสั่ง import 1คำสั่ง

```
Import      datetime, decimal, os
```

คำสั่ง import จะเป็นการอ้างอิงโมดูลที่เราต้องการใช้กลุ่มคำสั่งข้างในโมดูล การอ้างอิงโมดูลแบบนี้ Python จะมองเห็นกลุ่มคำสั่งทั้งหมดในโมดูล และพร้อมนำไปใช้งานได้ตลอดเวลา แต่บางครั้งหากเราต้องการอ้างอิงโมดูล และอยากจะเลือกใช้เฉพาะคำสั่ง หรือกลุ่มคำสั่งภายในบางคำสั่ง ก็ต้องใช้คำสั่ง from ที่จะกล่าวถึงในหัวข้อต่อไป

คำสั่ง from

ตัวอย่างที่ 1 รูปแบบการเขียนคำสั่ง from เพื่ออ้างอิงโมดูล และกลุ่มคำสั่งทั้งหมด

from ชื่อโมดูล import คำสั่งหรือกลุ่มคำสั่ง

```
From os import *
```

สัญลักษณ์ * ใช้สำหรับเรียกใช้งานทุกคำสั่ง หรือกลุ่มคำสั่งในโมดูลนี้

ตัวอย่างที่ 2 รูปแบบการเขียนคำสั่ง from สำหรับเรียกใช้บางคำสั่ง หรือกลุ่มคำสั่งในโมดูลที่กำลังอ้างอิงในปัจจุบัน

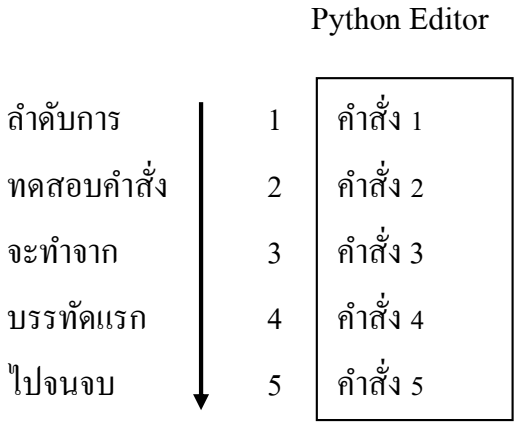
from ชื่อโมดูล import กลุ่มคำสั่งที่ 1, กลุ่มคำสั่งที่ 2

```
From os import error, chdir, getcwd
```

การเขียนโปรแกรมโดยทั่วไปด้วยภาษา python จะมีรูปแบบเป็นการเขียนกลุ่มคำสั่งลงในโพรซีเจอร์หรือฟังก์ชัน แล้วทำการเรียกใช้งานตามต้องการ เหมือนหลักการเขียนโปรแกรมด้วยภาษาอื่นๆ เช่น C, C++, Java ฯลฯ เพื่อง่ายต่อการทำความเข้าใจถึงวิธีการเขียนโปรแกรม python ด้วย editor พอสรุปได้เป็น 5 วิธีดังต่อไปนี้

1. รูปแบบการเขียนด้วยวิธีที่ 1
2. รูปแบบการเขียนด้วยวิธีที่ 2
3. รูปแบบการเขียนด้วยวิธีที่ 3 : แบบ Object Oriented
4. รูปแบบการเขียนด้วยวิธีที่ 4 : แบบสร้าง Object Oriented พร้อมกลุ่มคำสั่งเรียกใช้งาน Object
5. รูปแบบการเขียนด้วยวิธีที่ 5 : แบบ Object Oriented ที่เกิดจากวิธีที่ 3 และ 4 มาประยุกต์ใช้งานร่วมกัน

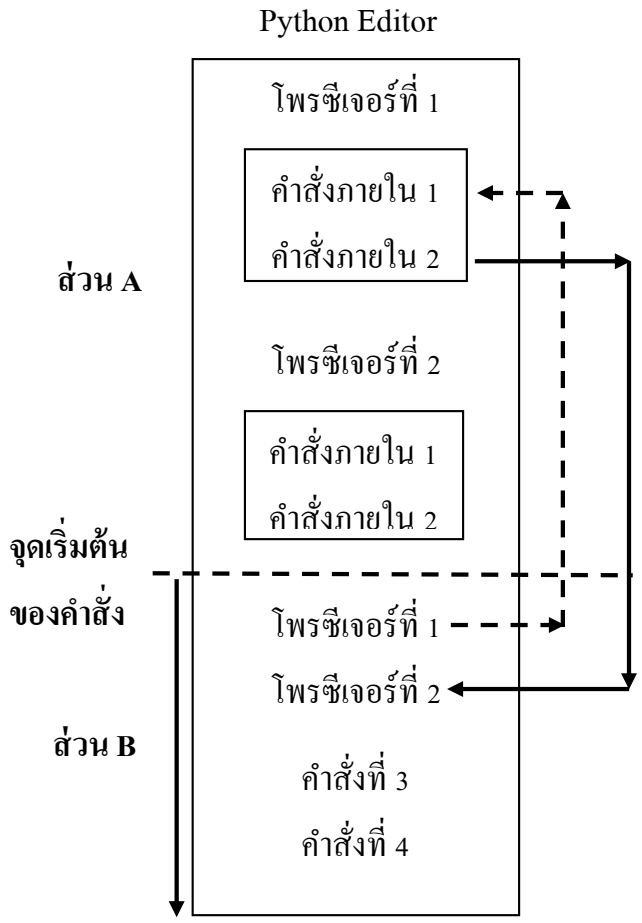
รูปแบบการเขียนด้วยวิธีที่ 1



รูปที่ 2.3 แสดงการเขียนโปรแกรมภาษา python ด้วยคำสั่งต่างๆ โดยไม่ได้ใช้โพรซีเจอร์หรือฟังก์ชันของ python

ตามรูปที่ 2.3 การเขียนคำสั่งต่างๆ จะทำการเขียนบรรทัดต่อบรรทัดจากบรรทัดแรกไอจนถึงบรรทัดสุดท้าย เมื่อทำการทดสอบโปรแกรม python จะทำงานคำสั่งในบรรทัดแรกไปเรื่อยๆ จนกระทั่งถึงบรรทัดสุดท้ายหรือคำสั่งสุดท้าย

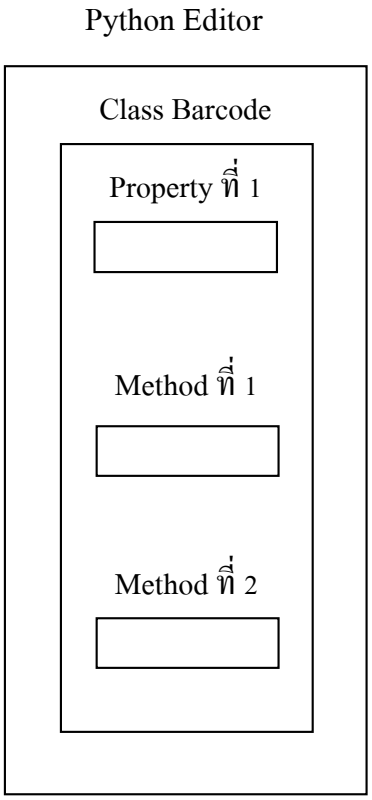
รูปแบบการเขียนด้วยวิธีที่ 2



รูปที่ 2.4 แสดงการเขียนโปรแกรม python ด้วยการสร้างโพรซีเจอร์ร่วมกับการใช้คำสั่งปกติ

ตามรูปที่ 2.4 เป็นรูปแบบการเขียนโปรแกรมที่นิยมใช้กัน เนื่องจากกลุ่มคำสั่งบางกลุ่มจำเป็นต้องทำงานร่วมกันตามลำดับขั้นตอน เราก็นำกลุ่มคำสั่งเหล่านั้นไปจัดสร้างไว้ในโพรซีเจอร์หรือฟังก์ชัน ซึ่งเพื่อความง่ายในการจัดสร้างโพรซีเจอร์หรือฟังก์ชัน ควรจะสร้างไว้ในฐานะ A ตามรูปที่ 2.4 ส่วนการเรียกใช้งานโพรซีเจอร์ ก็ให้ไปเขียนในสวน B (ตามรูปที่ 2.4) วิธีการเขียนตามรูปแบบนี้จะดีกับผู้เขียนโปรแกรมในระยะยาว ยกตัวอย่าง ถ้าผู้พัฒนาได้สร้างโปรแกรมขึ้นมาสักโปรแกรมหนึ่ง หลังจากสร้างเสร็จผ่านไปสัก 1 ปี แล้วต้องกลับมาแก้ไขเพิ่มเติมในโปรแกรม ถ้าจัดรูปแบบการเขียนไว้ดี เราก็จะหาจุดที่ต้องแก้ไข หรือเพิ่มเติมให้รวดเร็ว

รูปแบบการเขียนด้วยวิธีที่ 3 : แบบ Object Oriented

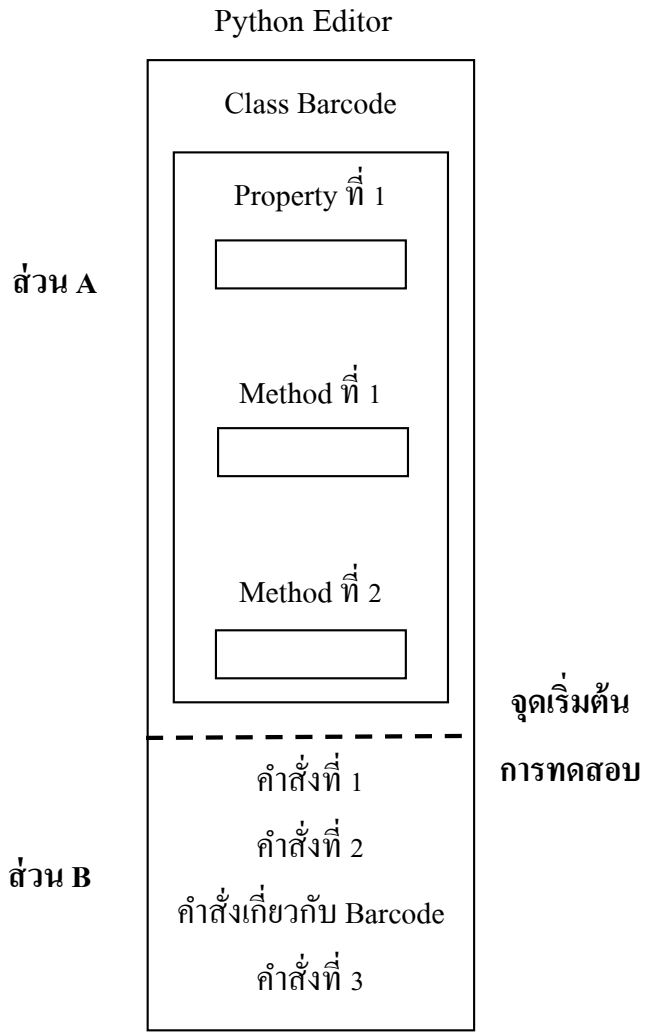


รูปที่ 2.5 แสดงการเขียนโปรแกรมเชิงวัตถุ(OO)
เพื่อสร้าง Object ด้วยภาษา python

ตามรูปที่ 2.5 เป็นวิธีการเขียนโปรแกรมเชิงวัตถุตามรูปแบบของ python ซึ่งเป็นวิธีการสร้าง Object ขึ้นมาเพื่อรอให้มีการนำไปใช้งาน ดังนั้นจะเห็นว่าภายใต้โครงสร้างแบบนี้จะไม่มีส่วนสำหรับเรียกทดสอบโปรแกรม

รูปแบบการเขียนด้วยวิธีที่ 4 : แบบ Object Oriented พร้อมกลุ่มคำสั่งเรียกใช้งาน Object

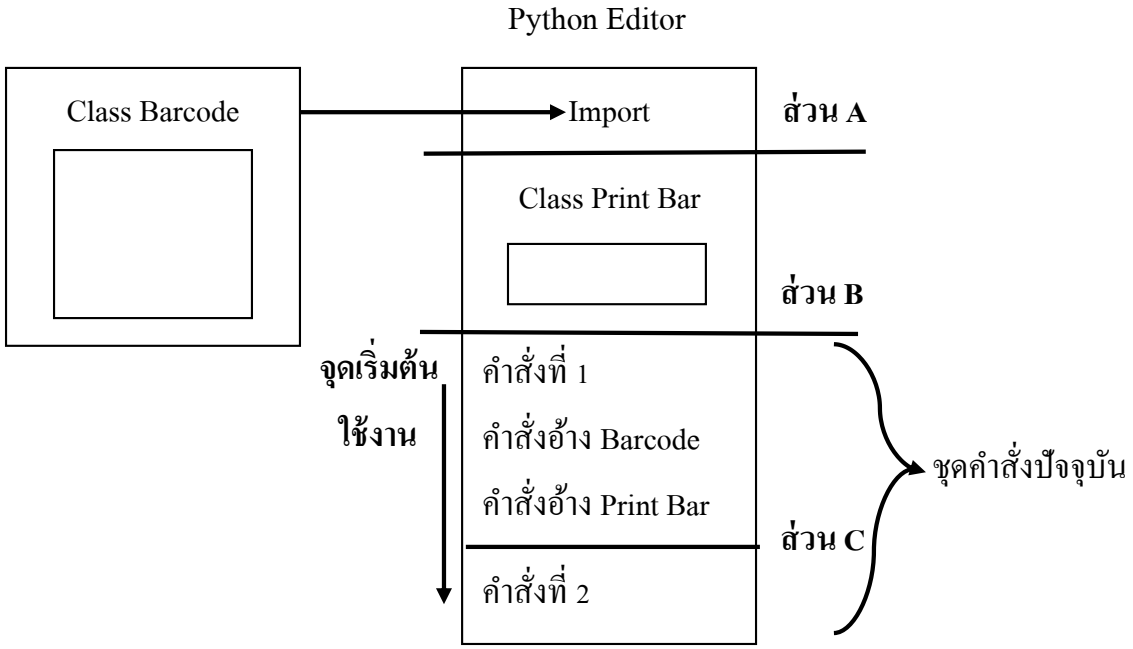
ในรูปที่ 2.6 ก็จะคล้ายการเขียนโปรแกรมตามรูปแบบที่ 3 เพียงแค่เพิ่มกลุ่มคำสั่งในส่วน B (ตามรูปที่ 2.6) สำหรับเรียกใช้ส่วนA ที่เป็นการสร้าง Class Object มาใช้งานภายใต้โครงสร้างเดียวกันเลย



รูปที่ 2.6 แสดงวิธีการเขียนโปรแกรมแบบ Object Oriented ร่วมกับกลุ่มคำสั่งที่เรียกใช้งาน Object ภายใต้โครงสร้างเดียวกัน

รูปแบบการเขียนด้วยวิธีที่ 5 : แบบ Object Oriented ที่เกิดจากวิธีที่ 3 และ 4 มาประยุกต์ใช้งานร่วมกัน

ในรูปที่ 2.7 ก็เป็นอีกวิธีที่นิยมใช้งานเชิงประยุกต์ สำหรับการเขียนโปรแกรมเชิงวัตถุของ python โดยบางกรณีก็มีการสร้าง Class object จัดเก็บไปเฉพาะ เมื่อไรที่ต้องการนำมาใช้งานก็จะทำการ Import เข้ามาใช้ร่วมกับโปรแกรมที่กำลังพัฒนาในปัจจุบัน

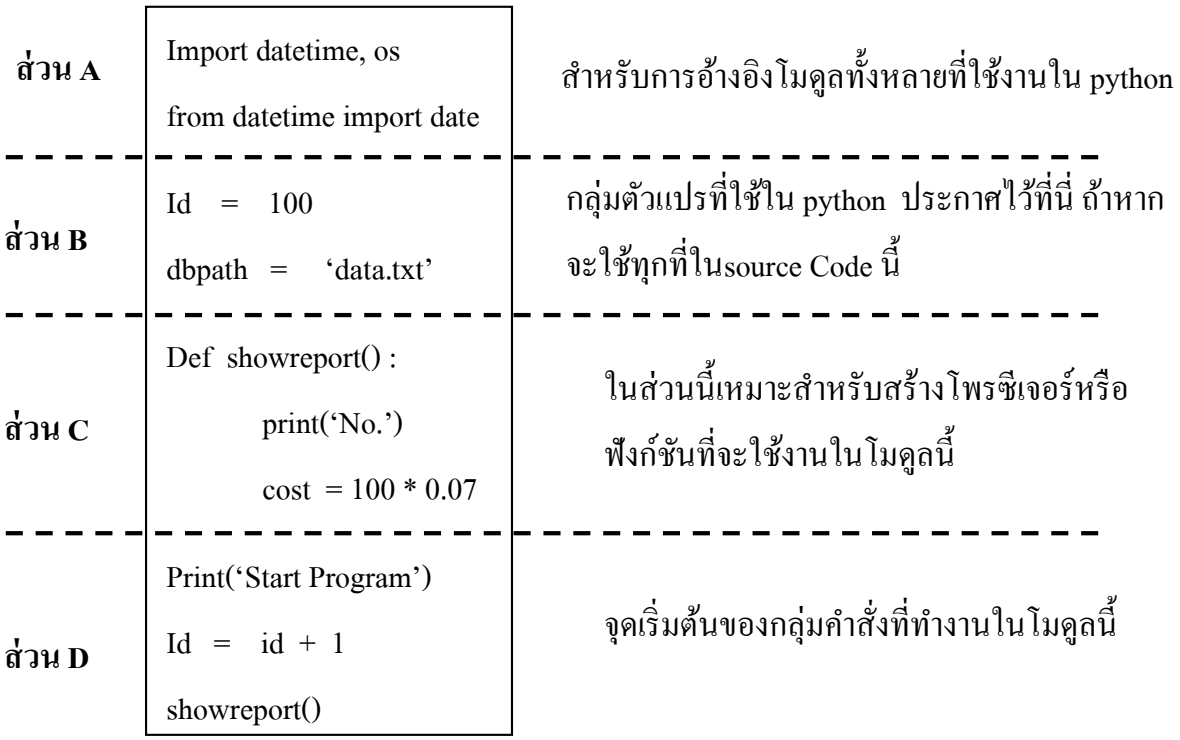


รูปที่ 2.7 แสดงการเขียนโปรแกรมเชิงวัตถุโดยการเรียกใช้โมดูลที่เป็น Class object จากภายนอกมาใช้งานเพิ่มเติมร่วมกับชุดคำสั่งปัจจุบัน

ด้วยลักษณะวิธีการเขียนโปรแกรมทั้ง 4 วิธี ที่กล่าวไปข้างต้นสำหรับภาษา python จะทำการจัดเก็บกลุ่มคำสั่ง หรือ (source Code) ในรูปแบบไฟล์ที่ใช้นามสกุล .py ซึ่งจะเป็นไฟล์แบบข้อความ (text file) หรืออาจแปลงให้เป็นรูปแบบไฟล์ไบนารี โดยจะใช้นามสกุลเป็น.pyc

ไฟล์ .py เป็นไฟล์ source Code แบบ ข้อความ
ไฟล์ .pyc เป็นไฟล์ source Code แบบ ไบนารี

สำหรับการเขียนซอร์สโค้ดด้วย editor ของ python เพื่อให้ง่ายต่อการเขียนโปรแกรม หรือการค้นหา หรือทำการปรับปรุงซอร์สโค้ดในระหว่างการเขียนโปรแกรม หรือแก้ไขเพิ่มเติมให้กับโปรแกรมหลังจากปิดงานเขียนโปรแกรมไปแล้ว ผู้เขียนได้แบ่งโครงสร้างการเขียน source Code ของ python ออกเป็น 4 ส่วน ดังรูปที่ 2.8 ซึ่งประกอบด้วย



รูปที่ 2.8 แสดงการแบ่งพื้นที่สำหรับการใช้ตัวแปร, ฟังก์ชั่น และคำสั่งภายใต้โครงสร้างของโมดูลที่กำลังพัฒนานี้

ส่วน A กรณีที่ต้องการอ้างอิงกลุ่มคำสั่งอื่นๆ จากโมดูลมาตรฐานของ python หรือโมดูลเพิ่มเติมพิเศษ เพื่อนำมาใช้งานในการเขียนโปรแกรมในโมดูลนี้ เราจะใช้คำสั่งสำหรับอ้างอิงโมดูล คือ คำสั่ง Import และคำสั่ง from...Import

ส่วน B ใช้ในกรณีที่ต้องการสร้างตัวแปรที่สามารถเรียกใช้งานได้ในส่วน C ที่เกี่ยวกับโพรซีเจอร์หรือฟังก์ชั่น และส่วนD ซึ่งเป็นจุดที่กลุ่มคำสั่งทำงาน

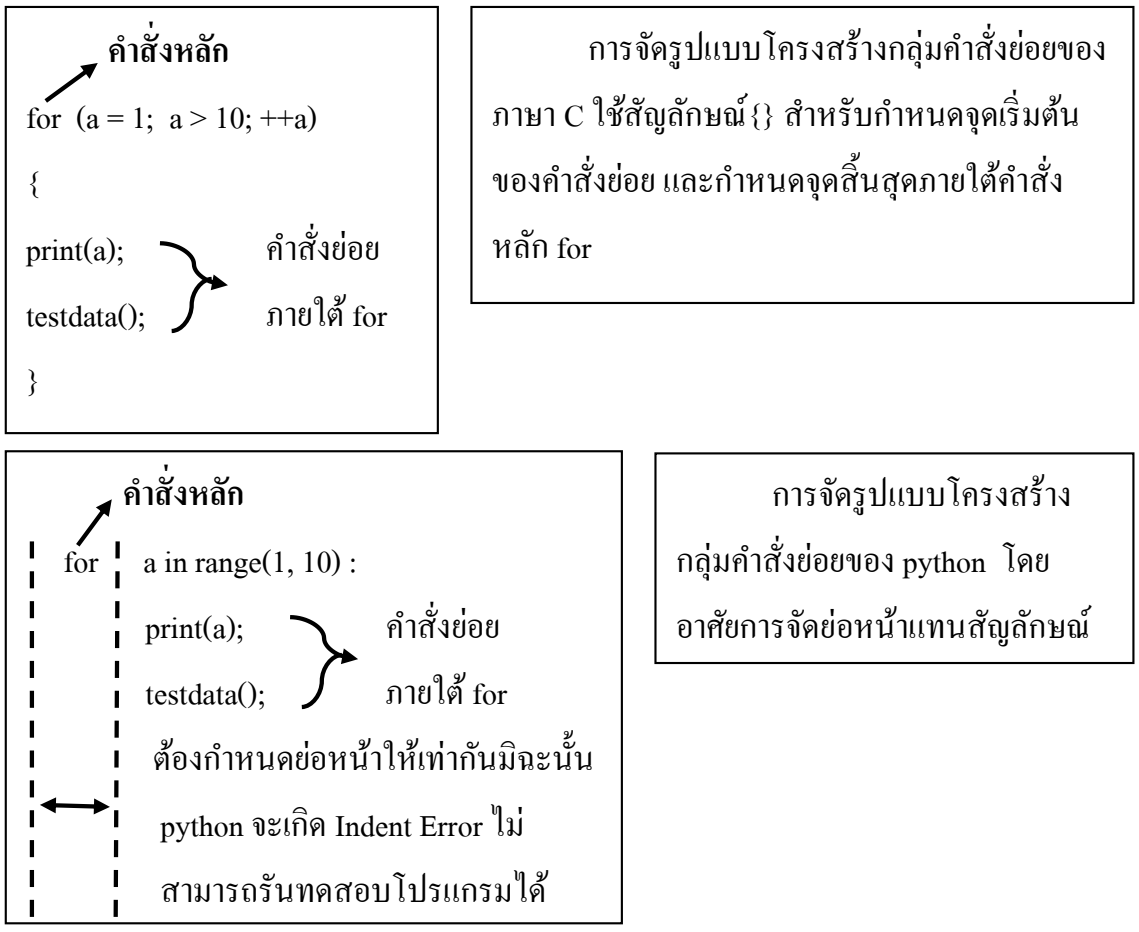
ส่วน C มีไว้สำหรับทำการสร้างกลุ่มโพรซีเจอร์หรือฟังก์ชัน เพื่อให้สามารถเรียกใช้งานภายในโมดูลนี้

ส่วน D เป็นจุดเริ่มต้นของการทำงานในโมดูลนี้ ที่นี่เราจะเขียนคำสั่งต่างๆ เพื่อใช้งานร่วมกับตัวแปรในส่วน B หรือการเรียกใช้งานโพรซีเจอร์หรือฟังก์ชันในส่วน C ของโมดูลนี้

หลังจากทำความเข้าใจโครงสร้างของการสร้างและเขียน source Code ของ python ก็เหลือสิ่งที่ควรรู้ก่อนลงมือเขียนโปรแกรม python อีกไม่กี่เรื่อง เช่น การกำหนดคำสั่งกลุ่มย่อยให้กับคำสั่งหลัก (Indenting), การทำความเข้าใจการใช้เครื่องหมาย ‘ หรือ “ สำหรับตัวแปรชนิดข้อความ (String), การใช้เครื่องหมาย () [] {} ร่วมกับตัวแปรหรือฟังก์ชันภายในของ python (ฟังก์ชันภายใน ได้แก่ คำสั่ง format, tolower, ฯลฯ) และวิธีการทดสอบโปรแกรมภายใต้ editor ของ python ก็จะทำให้ผู้ที่ต้องการศึกษาสามารถทดสอบเขียนโปรแกรมและทดสอบเองได้ เมื่อเขียนโปรแกรมคล่องขึ้น ก็สามารถค้นหาตัวอย่าง source Code ที่มีความหลากหลาย รวมทั้งเทคนิคการเขียนโปรแกรมแบบพิเศษจะ internet มาทดลอง และประยุกต์เข้ากับงานที่กำลังศึกษา หรือสร้างเพื่อนำไปใช้งาน

การกำหนดคำสั่งกลุ่มย่อยให้กับคำสั่งหลัก (Indenting)

วิธีการจัดรูปแบบโครงสร้างของกลุ่มคำสั่งย่อย ภายใต้คำสั่งหลักภาษา python จะอาศัยการจัดย่อหน้าแทนการใช้สัญลักษณ์เหมือนภาษาอื่น ตามรูปที่ 2.9 แสดงการจัดรูปแบบโครงสร้างของกลุ่มคำสั่งย่อยของ python เทียบกับภาษา C



รูปที่ 2.9 แสดงการเปรียบเทียบการจัดรูปแบบโครงสร้างกลุ่มคำสั่งย่อยของภาษา python

โดยตัวอย่างตามรูปที่ 2.9 จะเป็นการใช้คำสั่ง for ซึ่งถือว่าเป็นคำสั่งหลัก และมีคำสั่งย่อยเพื่อให้เกิดกระบวนการทำงานตามรอบการวนลูป คือคำสั่ง print และคำสั่งที่ 2 คือเรียกไปที่ฟังก์ชั่น Test Data ในโมดูลใช้งาน จะเห็นว่าการเขียนคำสั่งย่อยของ python จะไม่มีการใช้สัญลักษณ์ใดๆ เป็นตัวบอกการเป็นคำสั่งย่อย เมื่อเทียบกับภาษาC ที่ใช้สัญลักษณ์{ เป็นจุดเริ่มต้นและใช้สัญลักษณ์} สำหรับบอกจุดสิ้นสุด ภาษา python จะใช้การย่อหน้า ด้วยวิธีการกดปุ่ม Space Bar หรือกดปุ่ม Tab ก็แล้วแต่ผู้เขียนโปรแกรมจะถนัดแบบใด (แนะนำการย่อหน้าของคำสั่งย่อย ควรใช้ปุ่ม Tab ดีกว่า โอกาสเกิดข้อผิดพลาดจากย่อน่าจะน้อยลง) ถ้าหากผู้ศึกษาเจอข้อความ error ตามรูปที่ 2.10 ก็แสดงว่าการย่อหน้าของคำสั่งมีปัญหา ซึ่งส่วนใหญ่จะเกิดจากการย่อหน้าไม่เท่ากัน

สรุป การใช้ย่อหน้าเพื่อแบ่งคำสั่งย่อย จะใช้กับกลุ่มคำสั่งเช่น if, for, while, def (สำหรับสร้างฟังก์ชันเพื่อเรียกใช้งานในโมดูลนี้), Class (สำหรับสร้างโครงภายในเป็นวัตถุ เช่น Property, Method ฯลฯ) เป็นต้น

```
SyntaxError : expected
an indented block
```

รูปที่ 2.10 แสดงข้อความผิดพลาดเกี่ยวกับการกำหนดย่อหน้าหรือ Indent ผิดพลาดในการเขียนโปรแกรม

การทำความเข้าใจการใช้เครื่องหมาย ‘ หรือ “ สำหรับข้อความหรือตัวแปรชนิดข้อความ

ใช้สัญลักษณ์ร่วมกับกลุ่มข้อความในภาษา python มีให้เลือก 2 แบบ คือ การใช้สัญลักษณ์ฝนทองหรือ ‘ (Single Quote) และสัญลักษณ์ ? หรือ “ (Double Quote) โดยผู้เขียนโปรแกรมสามารถเลือกใช้สัญลักษณ์ดังกล่าวตามความถนัด สำหรับวิธีการใช้งานทั้ง 2 สัญลักษณ์จะแบ่งออกเป็น 2 วิธีดังนี้

1. วิธีใช้งานสัญลักษณ์ร่วมกับข้อความแบบบรรทัดเดียว (In-Line)

```
a = 'This is my book.'
b =  "วันนี้อยากทุบกระป๋อง"
```

2. การใช้งานสัญลักษณ์ร่วมกับข้อความแบบหลายบรรทัด เมื่อใช้สัญลักษณ์กำหนดจุดเริ่มต้นและจุดสิ้นสุดของข้อความ ให้พิมพ์สัญลักษณ์อย่างละ 3 เครื่องหมาย ตามตัวอย่างต่อไปนี้

```
a = '''This is my book.
        The cover is Yellow color.
        Book owner write something.'''
b = """ปูเป้ออกเดินทางไปเมืองที่แสนไกล
        ไกล แล้วไกลออกไปเรื่อยๆ จนในที่สุด
        ปูเป้ก็... รออ่านต่อฉบับถัดไป """
```

การใช้เครื่องหมาย (), [], {} ร่วมกับตัวแปรหรือฟังก์ชัน

python จะใช้เครื่องหมาย (), [], {} สำหรับการเขียนผสมผสานร่วมกับตัวแปร, คำสั่ง, ฟังก์ชันภายใน หรือบางกรณีก็ทำตัวเป็นฟังก์ชันภายในแบบอัตโนมัติ โดยไม่ต้องพิมพ์คำสั่ง ด้วยความหลากหลายต่อวิธีการนำไปใช้ และให้ง่ายต่อการทำความเข้าใจ ผู้เขียนจึงได้รวบรวมวิธีการเขียนสำหรับการใช้งานไว้ 4 ประเภทดังนี้

1. ประเภทการใช้เครื่องหมายร่วมกับตัวแปร
2. ประเภทการใช้เครื่องหมายร่วมกับโพรซีเจอร์หรือฟังก์ชัน
3. ประเภทการใช้เครื่องหมายร่วมกับคำสั่งหรือฟังก์ชันภายใน
4. ประเภทการใช้เครื่องหมายที่ทำหน้าที่เสมือนเป็นฟังชั่นภายในแบบอัตโนมัติ

ประเภทการใช้เครื่องหมายร่วมกับตัวแปร

โดยส่วนใหญ่ถ้ามีการใช้เครื่องหมายร่วมกับตัวแปร ลักษณะของตัวแปรเหล่านั้นจะเป็นประเภท Sequence (ขายตัวแปรอาร์เรย์ในภาษาอื่น) ดังตัวอย่างต่อไปนี้

```
a = (1, 2, 3, 4)
b = ['TV ', 'หลอดไฟ ', 'ปลาทู ']
c = {100: 'TV ', 200: 'หลอดไฟ ', 300: 'ปลาทู '}
```

จากตัวอย่างเป็นการกำหนดค่าต่างๆ ลงในตัวแปรประเภท Sequence ซึ่งประกอบด้วย 3 ชนิด ได้แก่ Tuple จะใช้เครื่องหมาย (), List จะใช้เครื่องหมาย [] และDictionary จะใช้เครื่องหมาย {} ตัวอย่างต่อไปนี้เป็นการใช้เครื่องหมาย [] สำหรับการอ่านค่าในตัวแปรประเภท Sequence

ตัวอย่าง print(a[2], b[1], c[300])

ผลลัพธ์ >>>3 หลอดไฟ ปลาทู

ประเภทการใช้เครื่องหมายร่วมกับโพรซีเจอร์หรือฟังก์ชัน

ในการสร้างโพรซีเจอร์หรือฟังก์ชันในโมดูล จะมีการกำกับด้วยเครื่องหมาย () หลังชื่อโพรซีเจอร์หรือฟังก์ชันที่สร้างขึ้นมา ด้วยจะถูกนำไปใช้สำหรับการผ่านค่าอาร์กิวเมนต์ (Argument) ของโพรซีเจอร์หรือฟังก์ชัน ตามตัวอย่างดังต่อไปนี้

ตัวอย่างที่ 1 แสดงการสร้างโพรซีเจอร์หรือฟังก์ชันโดยไม่มีตัวแปรอาร์กิวเมนต์

```
def showtime ():
    a =  100
    b =   a + 10
    print(a, b)
```

← การสร้างฟังก์ชัน Showtime ต้องมีเครื่องหมาย () เสมอ

กลุ่มคำสั่งภายใน

ตัวอย่างที่ 2 แสดงการสร้างโพรซีเจอร์หรือฟังก์ชันพร้อมกับตัวแปรอาร์กิวเมนต์

```
def bookinfo(title, author) :
    a =  100
    b =   a + 10
    print(title, a)
    print(b, author)
```

← กำหนดตัวแปรอาร์กิวเมนต์ภายในเครื่องหมาย () ของชื่อฟังก์ชันนี้

นอกจากนี้จะใช้เครื่องหมาย () สำหรับใช้ร่วมกับการสร้างโพรซีเจอร์หรือฟังก์ชันแล้ว การเรียกโพรซีเจอร์หรือฟังก์ชันเพื่อใช้งานก็จำเป็นต้องมีเครื่องหมาย () ต่อท้ายชื่อขณะใช้งาน ตามตัวอย่างดังต่อไปนี้

ตัวอย่างที่ 1 วิธีการเขียนโปรแกรมเรียกใช้งานโพรซีเจอร์หรือฟังก์ชันโดยไม่มีค่าอาร์กิวเมนต์

```
Showtime ()
```

← ต้องมีเครื่องหมาย () เสมอ

ตัวอย่างที่ 2 วิธีการเขียนโปรแกรมเรียกใช้งานฟังก์ชันพร้อมส่งผ่านค่าอาร์กิวเมนต์ให้กับฟังชัน

```
Bookinfo('หนังสือชินจัง ' , 'ชิโนกุ ')
```

ประเภทการใช้เครื่องหมายร่วมกับคำสั่งหรือฟังก์ชันภายใน

รูปแบบของคำสั่งหรือฟังก์ชันภายในภาษา python วิธีการเขียนคำสั่งบางครั้งจะมีการใช้เครื่องหมาย (), {} ร่วมกับค่าต่างๆ ที่ใช้ในคำสั่งนั้นตามตัวอย่างต่อไปนี้

ตัวอย่างการใช้เครื่องหมาย () รับคำสั่ง print + ค่าที่ต้องการแสดงบนจอภาพ print ('ปลาทู 2 เข่ง ')

ตัวอย่างการใช้เครื่องหมาย () กับคำสั่ง tolower + ค่าที่ต้องการแปลงจากอักษรตัวใหญ่ไปอักษรตัวเล็ก

a = tolower ('COUNT ')

ตัวอย่างการใช้เครื่องหมาย {} กับคำสั่ง format + ค่าที่ต้องการจัดรูปแบบการแสดงค่าตัวเลข Print()'{0} to {1} '.Format(100, 'TV '))

ประเภทการใช้เครื่องหมายที่ทำหน้าที่เสมือนเป็นฟังชั่นภายในแบบอัตโนมัติ

python เป็นภาษาที่ได้รับการปรับปรุงให้การเรียกใช้ฟังก์ชันภายในที่ใช้งานบ่อยๆ ง่ายมากขึ้น วิธีการเขียนหรือใช้งานก็กระชับต่อการทำความเข้าใจและจดจำ ตามตัวอย่างต่อไปนี้

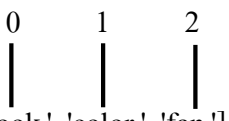
ตัวอย่างแสดงการใช้เครื่องหมาย [] ที่ทำหน้าที่แทนฟังก์ชันภายในร่วมกับตัวแปร โดยการใช้เครื่องหมาย [] ต่อท้ายตัวแปรเสมอการเรียกใช้ฟังก์ชันภายในที่ทำหน้าที่ตัดค่า (เหมือน Sub String ในภาษาอื่น)

กำหนดค่าตัวแปร a = 'This is my book '

print(a[1])

ผลลัพธ์ที่ได้ก็คือ ตัวอักษร h จะปรากฏบนจอภาพ

ตัวอย่างการใช้เครื่องหมาย [] ที่ทำหน้าที่แทนฟังก์ชันภายในร่วมกับตัวแปรประเภท Sequence



กำหนดค่าให้ตัวแปร a = ['book ', 'color ', 'fan ']

print(a[1][0])

ผลลัพธ์ที่ได้ คือ ตัวอักษร C จะปรากฏบนจอภาพ การใช้เครื่องหมาย [0] ต่อท้าย a[1] หมายถึงการตัดตัวอักษรตัวแรกของ a[1] ซึ่งค่าใน a ตำแหน่งที่ 1 ก็คือ color

การใช้เครื่องหมายคอมเมนต์ (Comment)

ด้วยการทำงานของภาษา python ซึ่งเป็นแบบ interpreter ทำให้การประมวลผลคำสั่งทีละบรรทัด การใช้ คอมเมนต์ อธิบายหมายเหตุต่างๆ จึงต้องใช้เครื่องหมายคอมเมนต์ ได้ทีละบรรทัด โดยใช้เครื่องหมาย # (Shape Sing) ตามตัวอย่างต่อไปนี้

ตัวอย่างการใช้เครื่องหมาย # สำหรับใส่คำอธิบายหรือหมายเหตุ

```
# sample python code  <-- บรรทัดที่จะไม่มีการประมวลผลใดเนื่องจากมีการคอมเมนต์
a = 100
b = 20
c = a + b
# print(a, b)
print(c)
b = 20 * 100          # คูณอีก 100 เท่า
print(b)
```

Python กับการใช้ตัวอักษรตัวเล็กหรือใหญ่ (Case Sensitive)

การเขียนโปรแกรมด้วยภาษา python และต้องใช้ความระมัดระวังเรื่องการใช้ตัวอักษรเล็กหรือใหญ่ สำหรับการประกาศตัวแปร การเขียนคำสั่งภายใน การเรียกชื่อโพรซีเจอร์หรือฟังก์ชัน และอื่นๆ ดังนั้นจึงมีความจำเป็นที่ต้องให้ความสนใจเมื่อมีการเรียกใช้งาน มิฉะนั้นจะส่งผลให้เกิดข้อผิดพลาด error เกี่ยวกับตัวอักษรเล็กหรือใหญ่ระหว่างใช้งานขึ้นได้

มาทดลองเขียนโปรแกรมด้วยภาษา python

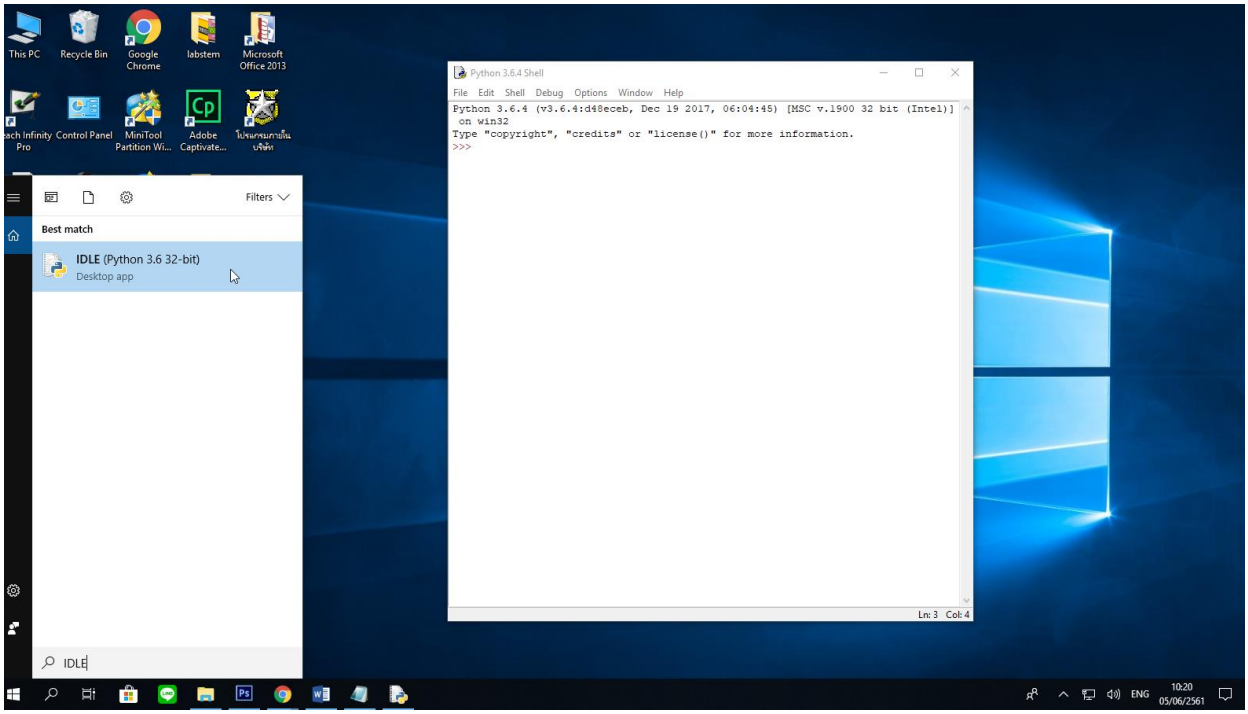
การทดลองเขียนคำสั่งต่างๆ ของภาษา python สามารถทดสอบได้ 2 วิธีคือ

1 ทดลองเขียนคำสั่งด้วย python shell

2 ทดลองเขียนคำสั่งด้วยรูปแบบไฟล์โปรแกรมแบบ python

ทดลองเขียนคำสั่งด้วย python shell

การเรียนรู้เกี่ยวกับวิธีการเขียนโปรแกรมที่ดีที่สุดคือการทดลองพิมพ์คำสั่งต่างๆเพื่อดูผลลัพธ์ที่เกิดขึ้นซึ่งอาจจะเกิดข้อผิดพลาดในการทดสอบคำสั่งหรือประสบความสำเร็จในการทำงานของคำสั่งนั้นๆ อีกทั้งทำให้ผู้ฝึกเขียนโปรแกรมได้ประสบการณ์การเขียนและทดสอบจริงมากกว่าเข้าใจจากทฤษฎีแต่เพียงอย่างเดียว พูดแล้วก็เริ่มเปิดโปรแกรม python IDLE ใน Windows ตามรูปที่ 2.11



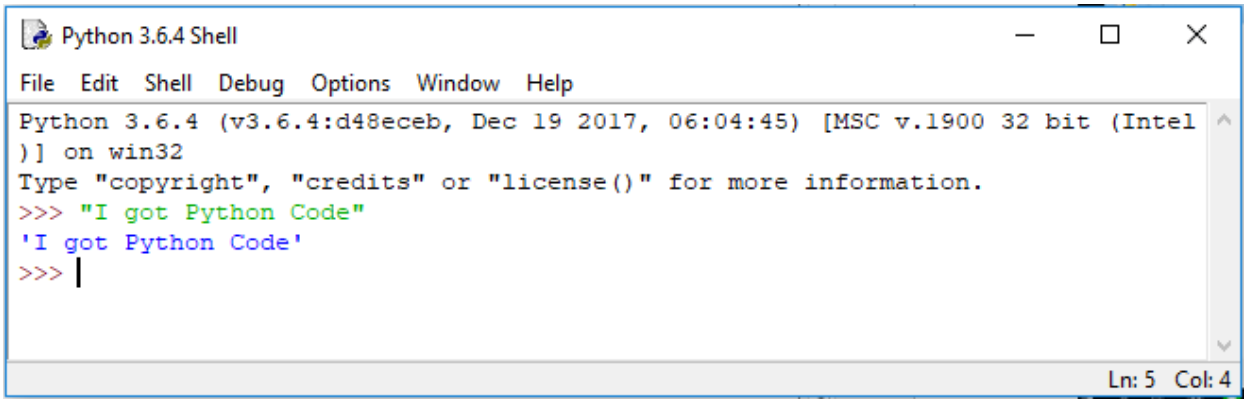
รูปที่ 2.11 แสดงขั้นตอนการเปิด python IDLE ภายใต้ระบบปฏิบัติการ Windows

หลังจากเปิด python IDLE ได้แล้วลองมาทดลองพิมพ์คำสั่งต่างๆ ตามตัวอย่างต่อไปนี้

ตัวอย่างที่ 1 ทดลองพิมพ์ข้อความลงใน python Shell

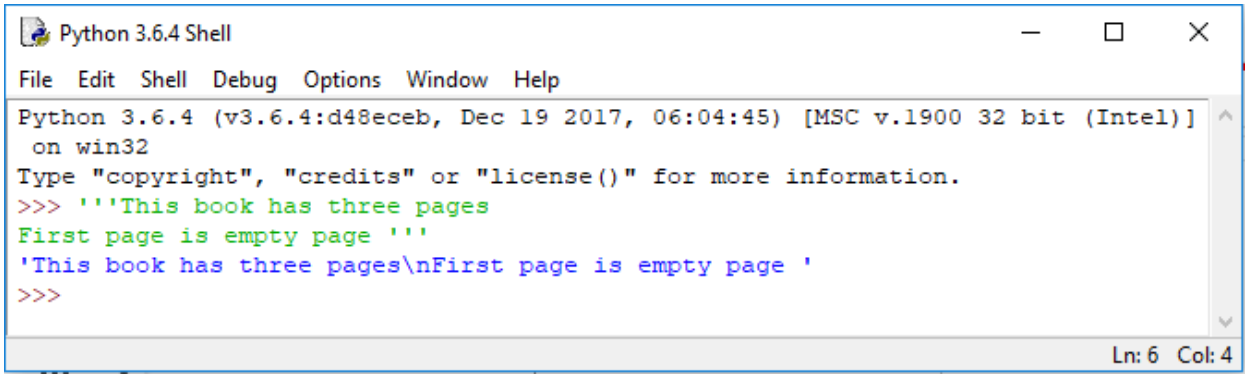
```
>>> "I got Python Code"
```

หลังจากพิมพ์ข้อความเสร็จสิ้นให้กดปุ่ม enter ผลลัพธ์ที่ได้จาก python Shell จะปรากฏข้อความดังกล่าวขึ้นมาในบรรทัดถัดไป



ตัวอย่างที่ 2 ทดลองพิมพ์ข้อความหลายบรรทัดด้วยเครื่องหมาย """ หรือ '''

```
>>> '''This book has three pages
First page is empty page '''
```



ตัวอย่างที่ 3 ทดลองใช้คำสั่ง print ร่วมกับข้อความ

สำหรับ Windows : >>> print(' My Book is ok ')

สำหรับ linux: >>> print 'My book is ok ' โดยไม่ต้องใส่วงเล็บร่วมกับฟังก์ชัน

ตัวอย่างที่ 4 แสดงการเชื่อมต่อระหว่างข้อความตั้งแต่ 2 ข้อความขึ้นไป

>>> 'Joe ' + 'Mary ' แสดงการเชื่อมต่อด้วยเครื่องหมาย +

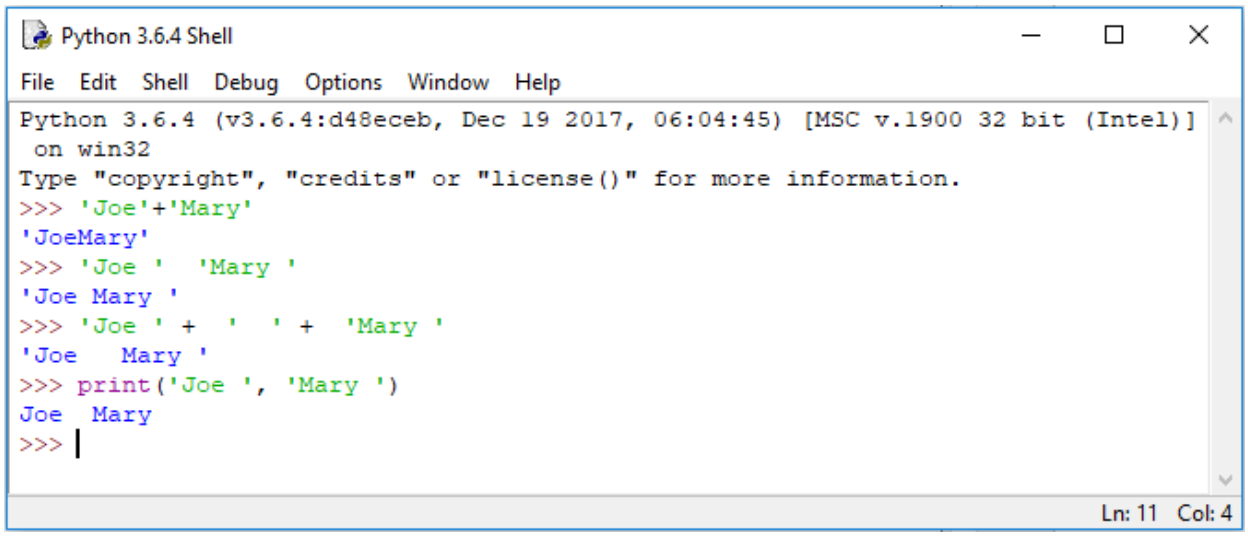
หรือ

>>> 'Joe ' 'Mary ' แสดงการเชื่อมต่อด้วยการเขียนแบบต่อเนื่อง

หรือ

>>> 'Joe ' + ' ' + 'Mary ' แสดงการเชื่อมต่อด้วยเครื่องหมาย + กับข้อความ 3 ข้อความหรือใช้ร่วมกับคำสั่ง print

>>> print('Joe ', 'Mary ') แสดงการเชื่อมต่อระหว่าง 2 ข้อความพร้อมใช้ร่วมกับคำสั่ง print



ตัวอย่างที่ 5 แสดงการใช้คำสั่งตรวจสอบชนิดของข้อมูล

>>> type(100) ตรวจสอบค่า 100 ใน python ผลลัพธ์ที่ได้จะบอกชนิดของค่าที่ตรวจสอบ

<class 'int'>

>>> type(9999999)

<class 'int'>

>>> type(3.143)

<class 'float'>

>>> type('joe')

<class 'string'>

>>> type(5+4j)

<class 'complex'>

ตัวอย่างที่ 6 ทดสอบการอ้างอิงโมดูลที่นำมาใช้งาน

กรณีที่ 1 การอ้างอิงโมดูลที่มีใช้ใน Python

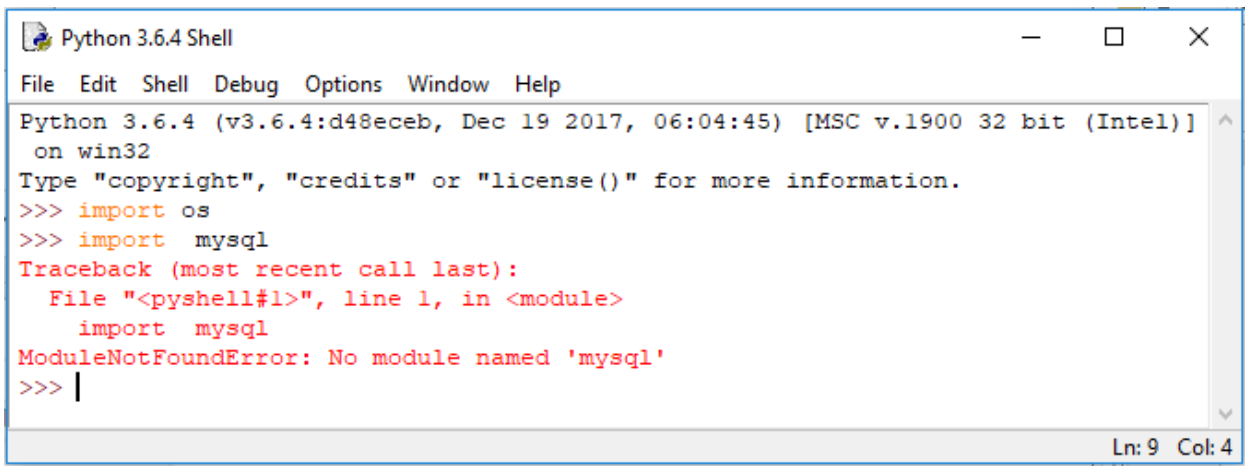
>>> Import os ทดสอบการอ้างอิงโมดูลชื่อ OS

>>> จะไม่ปรากฏอะไรถ้าหาก python รู้จักโมดูลชื่อ OS

กรณีที่ 2 การอ้างอิงโมดูลที่ไม่มีการติดตั้งใน python

>>> Import mysql ทดสอบการอ้างอิงโมดูลชื่อ mysql เพื่อทำการเชื่อมต่อกับฐานข้อมูล

>>> จะมีข้อความเกิด error ปรากฏขึ้นเพื่อแจ้งเตือนการใช้งานโมดูล

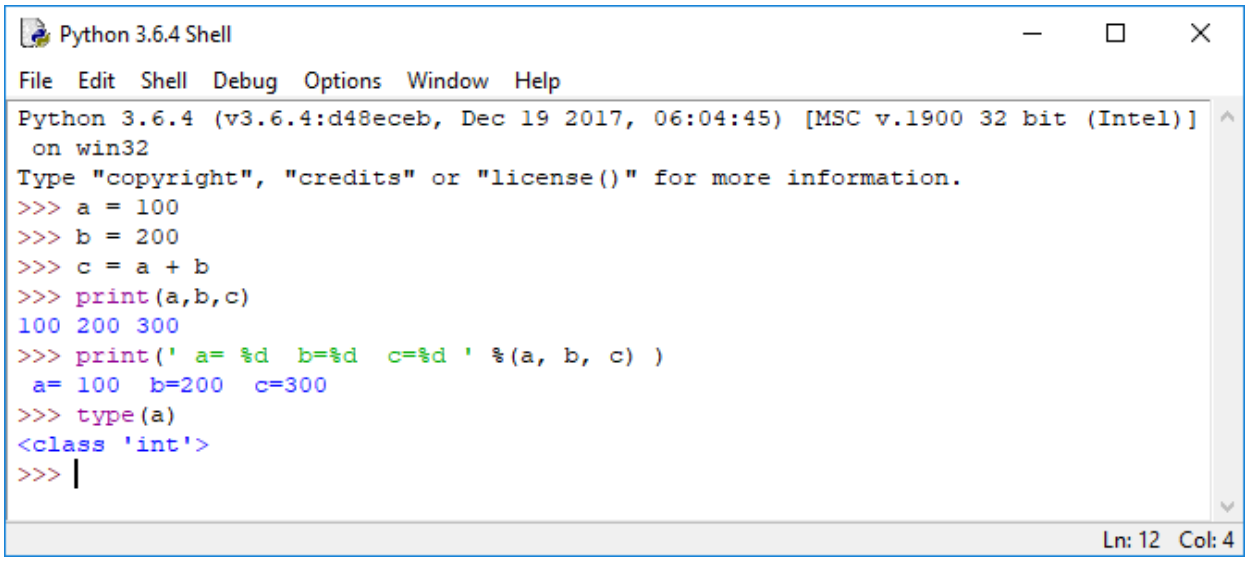


ตัวอย่างที่ 7 แสดงการใช้งานตัวแปรใน python Shell

```
>>> a = 100
>>> b = 200
>>> c = a + b
>>> print(a,b,c)
>>> print(' a= %d  b=%d  c=%d ' %(a, b, c) )
>>> type(a)
<class ' int ' >
```

พิมพ์ค่าในตัวแปร a, b, c ร่วมกับคำสั่ง print

พิมพ์ค่าในตัวแปร a, b, c พร้อมกับข้อความแบบผสมผสานกับค่าในตัวแปรด้วยคุณสมบัติของคำสั่ง print



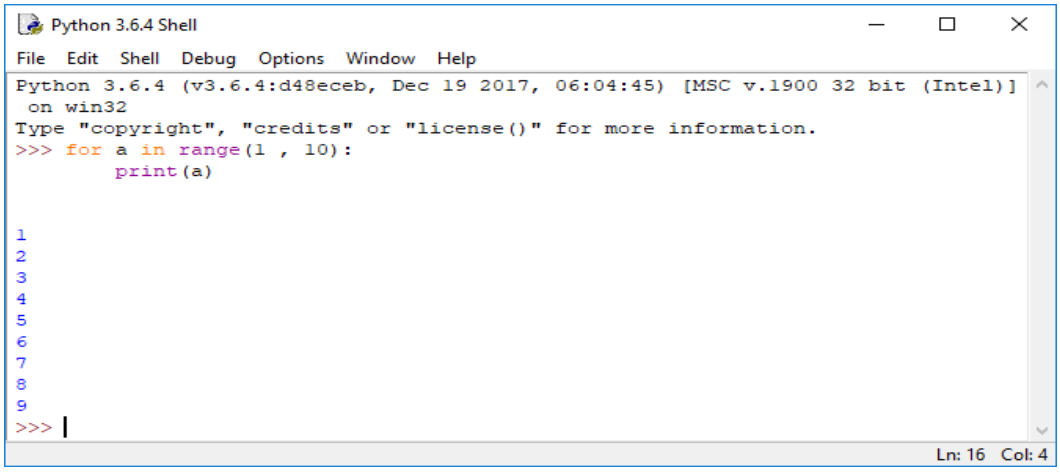
ตัวอย่างที่ 8 แสดงการใช้คำสั่งลูปด้วย python Shell

```
>>> for a in range(1 , 10):
        print(a)
```

เว้นว่างแล้วกดปุ่ม Enter

```
>>> 1
::
>>> 9
```

ผลลัพธ์ที่เกิด



สำหรับการใช้ python Shell ทดลองคำสั่งที่สามารถมีคำสั่งย่อยภายในตัวของคำสั่งหลัก (สังเกตคำสั่งหลักที่สามารถมีกลุ่มคำสั่งย่อยได้ มักจะนิยมใช้เครื่องหมาย : (Colon) ก่อนจะเริ่มแบ่งย่อหน้าและกำหนดคำสั่งย่อย) จะขึ้นบรรทัดใหม่พร้อมย่อหน้าให้อัตโนมัติเพื่อรอการพิมพ์คำสั่งย่อย วิธีที่ python Shell จะรู้ว่าคำสั่งย่อยสิ้นสุดก็คือ บรรทัดสุดท้ายไม่มีการพิมพ์ใดๆ หรือเว้นว่างไว้ (กดปุ่ม Enter สัก 2 ครั้งก็ได้ ก็ถือว่าสิ้นสุดแล้ว)

ตัวอย่างที่ 9 แสดงการใช้คำสั่งแปลงชนิดของค่าในตัวแปร

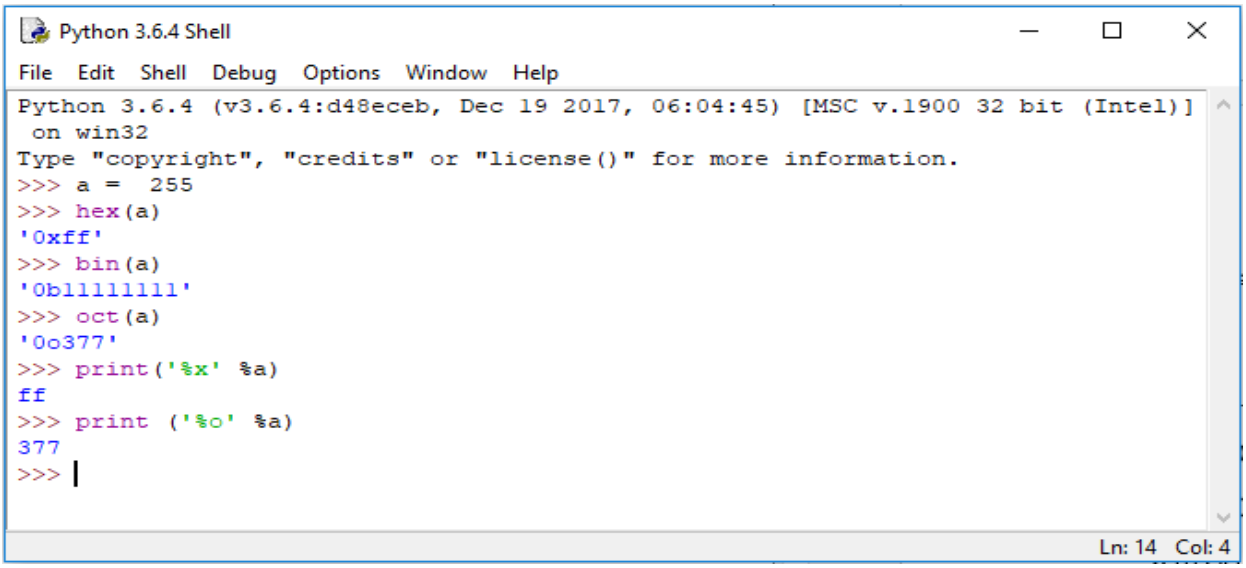
```
>>>  a = '100 '   ←—— กำหนด 100 แบบ string ให้กับตัวแปร a
>>> b = int(a)   ←—— ใช้คำสั่งฟังก์ชั่น int สำหรับแปลงค่าจากชนิด String ให้เป็นชนิด Int
>>> type(b)
<class 'int '>
>>> type(a)
<class 'string '>
```

ตัวอย่างที่ 10 แสดงการใช้คำสั่งเกี่ยวกับการแปลงเลขฐาน

```
>>> a =  255
>>> hex(a)
>>> bin(a)
>>> oct(a)
```

หรือใช้ร่วมกับคำสั่ง print

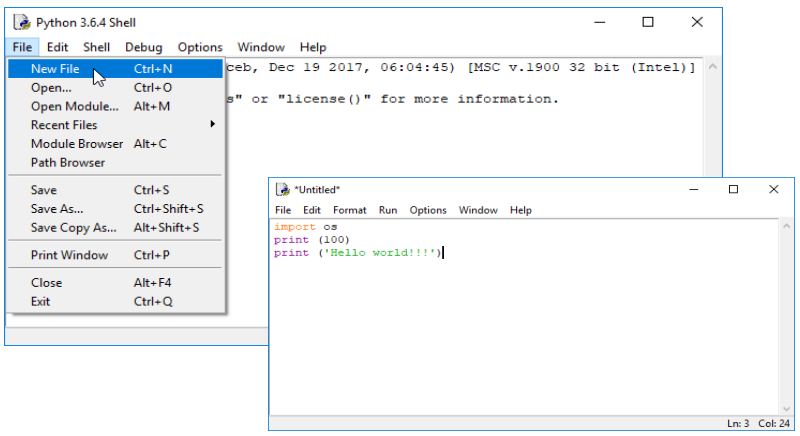
```
>>> print('%x' %a)ใช้ตัวอักษร x หรือ x สำหรับเลขฐานสิบหก (hex)
>>> print ('%o' %a)ใช้ตัวอักษร o หรือ o สำหรับเลขฐานแปด (oct)
```



หลังจากทดลองใช้ python shell กับคำสั่งต่างๆ ก็คงทำให้ผู้ใช้งานพอได้แนวทางการเขียนโปรแกรม และผลลัพธ์ที่ได้จากการทดสอบ แต่การทดสอบเขียนโปรแกรมด้วย python shell ก็มีข้อจำกัดในการจัดเก็บกลุ่มคำสั่ง ซึ่งเป็นสิ่งจำเป็นในการพัฒนางานในรูปแบบที่เป็นซอร์สโค้ด ด้วยเหตุนี้เราจึงสามารถเขียนโปรแกรมในรูปของไฟล์โปรแกรมได้ด้วย

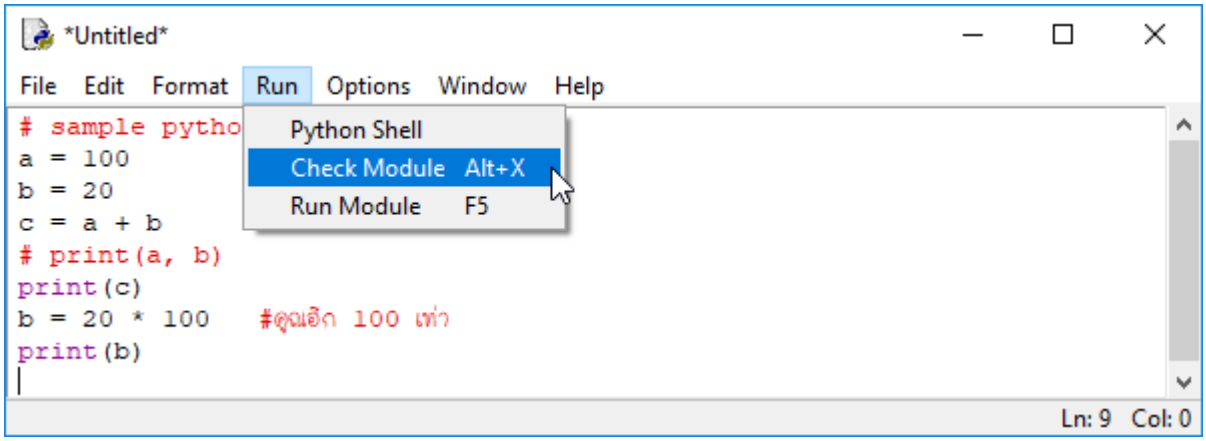
ทดลองเขียนคำสั่งด้วยรูปแบบไฟล์โปรแกรม

เมื่อพูดถึงรูปแบบการเขียนโปรแกรมที่เหมาะต่อการพัฒนางานจริงๆ จังๆแล้ว การเขียนคำสั่งทั้งหลายควรจะอยู่เป็นที่เป็นทาง ง่ายต่อการทำงานอย่างต่อเนื่องและค้นหาได้ง่าย ซึ่งก็เป็นไปตามแนวทาง การเขียนโปรแกรมด้วยเครื่องมือพัฒนาโปรแกรม ที่จัดเก็บในรูปของไฟล์ซอร์สโค้ด โดยไฟล์ซอร์สโค้ด ต้องมีนามสกุลเป็น .py ส่วนจะเก็บในโฟลเดอร์ไหนก็ได้ โดยไม่จำเป็นต้องเก็บไว้ที่เดียวกันกับโฟลเดอร์ที่ python ติดตั้งอยู่ ต่อไปนี้เป็นการทดลองสร้างไฟล์ซอร์สโค้ดไอด้วย Editor ดังรูปที่ 2.13 ซึ่งเป็นการเรียกใช้งาน Editor ด้วย python shell



รูปที่ 2.13 แสดงการเลือกเมนูใน python Shell เพื่อเรียกใช้งาน Editor

หลังจาก Editor ปรากฏขึ้น ให้ทดลองเขียนคำสั่งต่อไปนี้ เพื่อทดสอบการเขียนโปรแกรม และศึกษาการใช้เมนูต่างๆ เช่น ทดสอบการทำงานของโปรแกรม จัดเก็บไฟล์ซอร์สโค้ดลงในโฟลเดอร์ที่ต้องการ ฯลฯ



รูปที่ 2.14 ใช้เมนู Run เพื่อทดสอบการเขียนคำสั่งหรือสั่งให้โปรแกรมทำงาน

ขั้นตอนการทดสอบโปรแกรมตัวอย่าง ให้เลือกเมนู Run ใน editor เพื่อดูผลลัพธ์ที่ปรากฏใน python Shell เลือกเมนูย่อย Run Module (หรือจะกดปุ่ม F5 ก็ได้) เท่านี้ก็เสร็จสิ้นการทดสอบโปรแกรมที่กำลังพัฒนา

บทที่ 3 รู้จักการใช้ตัวแปรของ Python

พื้นฐานในการเขียนโปรแกรมที่นักพัฒนาโปรแกรมในแต่ละภาษาจำเป็นต้องรู้และเข้าใจ คือการให้งานตัวแปรของภาษา ซึ่งจะมีรายละเอียดการใช้งาน การประกาศตัวแปร การกำหนดชนิดตัวแปร การทำความเข้าใจเกี่ยวกับกลุ่มของชนิดตัวแปร ที่มีการใช้รูปแบบการเขียนตัวแปรแต่ละชนิดการใช้เครื่องหมายเช่น '(Quote) หรือ "(Double Quote) ร่วมกับค่าที่จะจัดเก็บในตัวแปร การกำหนดขอบเขตของตัวแปรสำหรับใช้งานในโมดูลที่กำลังเขียนโปรแกรมหรือการรีเซ็ต (Reset) ค่าในตัวแปรแต่ละชนิดเป็นต้น เพื่อง่ายต่อการทำความเข้าใจเกี่ยวกับตัวแปรของ python

➢ การเลือกใช้ตัวแปรสำหรับเขียนโปรแกรม

ปัจจุบันหลักการและแนวคิดการพัฒนาการเขียนโปรแกรม หรือภาษาที่ใช้ในการเขียนโปรแกรม มีขีดความสามารถรองรับการเขียนโปรแกรมในรูปแบบทั่วไป และ Object Base เชิงวัตถุ(Object Oriented) ดังนั้นการเลือกตัวแปรสำหรับใช้งานร่วมกับกลุ่มคำสั่งจึงต้องกำหนดให้เหมาะสมและถูกต้อง สำหรับตัวแปรของภาษาแบ่งออกเป็น 2 กลุ่มใหญ่ดังนี้

กลุ่มที่ 1 ตัวแปรชนิดทั่วไปได้แก่ ตัวเลข(Number), ข้อความ(String), วันที่ เวลา(Date Time), เลขทศนิยมหรือชนิดที่เป็น Sequence (คล้ายตัวแปร Array ในภาษาอื่น) เป็นต้น

กลุ่มที่ 2 ตัวแปรชนิด Object เช่น คำสั่ง Object เกี่ยวกับไฟล์ (File) หรือไดเรกทอรี (Dir), คำสั่ง object connect สำหรับการเชื่อมต่อฐานข้อมูล ฯลฯ

➢ รูปแบบการสร้างและใช้งานตัวแปร

รูปแบบการเขียนใน python ที่ได้รับการออกแบบมาให้ง่ายต่อการสร้าง และเรียกใช้งานโดยไม่ต้องมีขั้นตอนและรูปแบบยุ่งยากมากนัก โดยรูปแบบที่จะใช้ก็แค่กำหนดชื่อตัวแปรแล้วก็กำหนดค่าตัวเลข,ข้อความ, วันที่,เวลา,Object ให้กับตัวแปรตามตัวอย่างต่อไปนี้

ตัวอย่างที่ 1

```
a = 100
b = 12.76
c = 'Python' หรือ "Python"
d = MySQL.Connect("hostname","databasename")
```

อย่าลืมเครื่องหมาย ' และ "

ตัวอย่างที่ 2 ถ้าต้องการกำหนดค่า 100 ให้กับ a, b, c

a = b = c = 100

ภาษา Python จะกำหนดชนิดของตัวแปรตามค่าที่จัดเก็บลงในตัวแปรปัจจุบันโดยอัตโนมัติ ทุกครั้งที่มีการเปลี่ยนแปลงค่าที่จัดเก็บ ตามตัวอย่างดังต่อไปนี้

a = 100 เริ่มต้นกำหนด 100 ที่เป็นชนิด Int

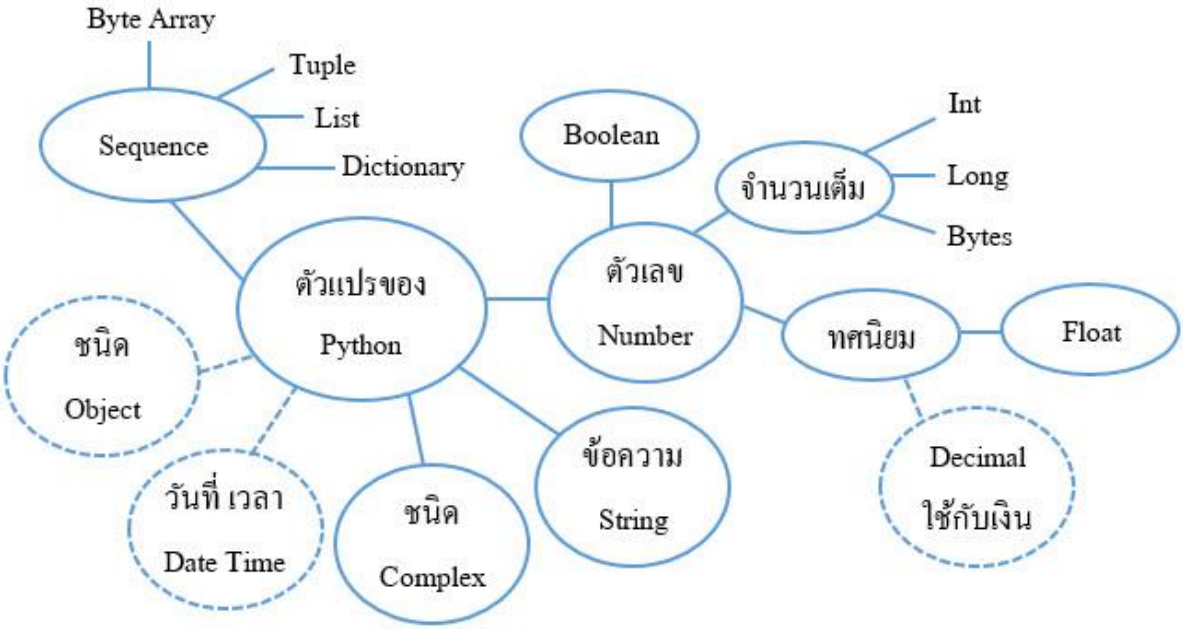
type(a) คำสั่ง type สำหรับตรวจชนิดของตัวแปร a

a = 'joe' กำหนดค่าข้อความให้กับตัวแปร a

type(a) ตรวจชนิดของตัวแปร a ซึ่งจะปรากฏเป็น String

เพื่อเป็นการตรวจสอบชนิดของตัวแปร เราสามารถใช้คำสั่งฟังก์ชันภายในที่ชื่อ Type ทำการทดสอบชนิดของตัวแปรในปัจจุบันได้

➢ ชนิดของตัวแปรที่มีการใช้งานใน Python



รูปแสดงโครงสร้างชนิดตัวแปรใน Python

ตามรูปแสดงให้เห็นถึงโครงสร้างชนิดตัวแปรใน Python โดยแบ่งชนิดของตัวแปรที่เกี่ยวกับเลขจำนวนเต็ม,ตัวเลขทศนิยม,ข้อความและชนิดที่เป็น Complex ที่ไม่ต้องใช้คำสั่ง import เนื่องจาก Python ได้กำหนดชนิดตัวแปรเหล่านี้ให้เป็นพื้นฐานที่ร่วมกับการเขียนโปรแกรมเสมอ (ในรูป ใช้เส้นวงกลมทึบสำหรับชนิดตัวแปรที่ไม่ต้อง import)

ตัวอย่างชนิดตัวแปรพื้นฐานที่สามารถใช้งานได้โดยไม่ต้อง import

ชื่อชนิดตัวแปร	ความหมาย	ตัวอย่าง
Long, Int	ตัวเลขจำนวนเต็ม	979
Float	ตัวเลขทศนิยม	3.1467
Complex หรือ Imaginary	ตัวเลขเชิงซ้อน	5+4j
String	ข้อความ	'Joe' หรือ "Joe"
Boolean	ค่าทางตรรกะ	True, False

ถ้าเป็นชนิดตัวแปรที่จำเป็นต้อง import ให้สังเกตเมื่อนำมาใช้งาน เวลาทดสอบโปรแกรมจะเกิดข้อผิดพลาด (Error) ขึ้น วิธีแก้ไข ก็ให้เพิ่มโมดูลที่อ้างอิงชนิดตัวแปรนั้นเข้าไปในโปรแกรมของเรา

ตัวอย่างชนิดตัวแปรที่จำเป็นต้อง import

ชื่อชนิดตัวแปร	**ความหมาย**	**ตัวอย่าง**
Date	ชนิดวันที่	datetime
Time	ชนิดเวลา	datetime
MySQL.Connect	ตัวเชื่อมฐานข้อมูล	MySQL
OS.path.isdir	ตรวจชื่อไดเรกทอรี	os

ทำความเข้าใจกับชนิดตัวแปรพื้นฐานของ Python

ชนิดตัวแปรพื้นฐานของ Python ถูกกำหนดไว้แล้วภายในตัว (Built-in) โดยการนำไปใช้ประกาศสร้างตัวแปร(เช่น ตัวแปรชื่อ Id สำหรับการจัดเก็บค่า 100 เมื่อเขียนเป็นคำสั่งจะได้ดังนี้ Id = 100) จะพร้อมใช้งานทันที ไม่ต้องอ้างอิงจากโมดูลภายนอก สำหรับขอบเขตของชนิดตัวแปรพื้นฐานแบบจัดสร้างภายใน (Built-in) มีรายละเอียดดังนี้

<u>ตัวแปรชนิดตัวเลขจำนวนเต็ม</u>

Python มีตัวแปรที่รองรับการใช้งานตัวเลขแบบจำนวนเต็มที่มีขนาดได้ 50 ถึง 100 หลัก ยิ่งจำนวนหลักมากเท่าใด ความเร็วในการใช้งานของตัวแปรชนิดนี้ก็จะลดลงตามขนาดความจุของตัวเลขเมื่อเทียบกับความจุชนิดตัวเลขจำนวนเต็มมาตรฐานทั่วไป จุดเด่นของชนิดตัวเลขแบบจำนวนเต็มใน Python อีกประการได้แก่ การใช้งานตัวเลขจำนวนเต็มที่เป็นเลขฐานสอง(Binary), ฐานแปด(Octal), ฐานสิบหก(Hexadecimal) ลงในตัวแปรชนิดตัวเลขจำนวนเต็มตามตัวอย่าง และการทดสอบต่อไปนี้ เริ่มจากเปิด Python IDLE แล้วพิมพ์ค่าตัวแปร a

```
>>> a = 12345678901234567890
>>> print(a)
12345678901234567890
```

ตัวแปร a จัดเก็บตัวเลขแบบจำนวนเต็มในรูปแบบตัวเลขฐานสิบ

ผลลัพธ์ที่ Python Shell จะแสดงค่า 12345678901234567890

สำหรับการจัดชนิดตัวเลขจำนวนเต็มแบบฐานสองลงในตัวแปร b จะกำหนด 0b นำหน้าเลขฐานสอง หรือใช้ฟังก์ชันภายในที่ชื่อว่า bin() สำหรับการจัดเก็บลงในตัวแปร b

ทดลองโดยการพิมพ์ค่าตัวเลขฐานสองลงในตัวแปร

```
>>> b = 0b1011
>>> print(b)
11
>>> print(type(b))
<class 'int'>
>>> b = bin(11)
>>> print(b)
0b1011
>>>
```

ป้อนค่าตัวเลข 1011 ฐานสองลงในตัวแปร b

ผลลัพธ์จะแสดงค่าตัวเลข 1011 ฐานสองหรือตรงกับตัวเลข 11 ฐานสิบ

ทดสอบเลขฐานสองโดยใช้ฟังก์ชันภายในที่ชื่อ bin สำหรับการจัดเก็บค่าลงในตัวแปร b

เช่นเดียวกับเลขฐานแปดและฐานสิบหกก็จะมีวิธีการเขียนและใช้งานคล้ายๆกับเลขฐานสองตามที่กล่าวไว้แล้ว

ชื่อเลขฐาน	คำนำหน้า	ฟังก์ชัน
สอง (Binary)	0b	bin(ตัวเลข)
แปด (Octal)	0o	oct(ตัวเลข)
สิบ (Decimal)	-	int(ตัวเลข)
สิบหก (Hexadecimal)	0x	hex(ตัวเลข)

ตารางเทียบการใช้ชนิดตัวเลขจำนวนเต็มร่วมกับเลขฐานสอง,แปด,สิบ,และสิบหก

แสดงรูปแบบการกำหนดคำนำหน้าค่าตัวเลขและการใช้ฟังก์ชันภายในที่ทำหน้าที่แปลงค่าตัวเลขจากเลขฐานสิบไปเป็นเลขฐานสอง,แปด,สิบ,และสิบหก

รูปแบบ	ลักษณะการทำงาน
i | j	เรียกว่า Bitwise OR
i ^ j	เรียกว่า Bitwise XOR
i & j	เรียกว่า Bitwise AND
i << j	เรียกว่า Shift left
i >> j	เรียกว่า Shift right
~ i	เรียกว่า Inverts i's bits

ตารางแสดงการใช้เครื่องหมายตรรกะร่วมกับตัวแปรชนิดตัวเลข

ทดลองพิมพ์ค่าตัวเลขฐานแปดและฐานสิบหกลงในตัวแปร

ตัวอย่างเลขฐานแปด

```
>>> c = 0o10
>>> print(c)
8
>>> d = oct(8)
>>> print(d)
0o10
>>>
```

ป้อนค่าตัวเลข 10 ฐานแปดลงในตัวแปร c

ป้อนค่าตัวเลข 8 ฐานสิบโดยใช้ฟังก์ชันภายในแปลงเลขฐานแปดให้กับตัวแปร d

ตัวอย่างเลขฐานสิบหก

```
>>> c = 0o10
>>> print(c)
8
>>> d = oct(8)
>>> print(d)
0o10
>>>
```

ป้อนค่าตัวเลข ff ฐานสิบหกลงในตัวแปร

ป้อนค่าตัวเลข 255 ฐานสิบแล้วใช้ฟังก์ชันภายในที่ชื่อ hex ช่วยแปลงค่าเป็นเลขฐานสิบหกเพื่อจัดเก็บในตัวแปร f

ตัวแปรชนิดบูลีน(Boolean)

Python กำหนดการใช้งานตัวแปรชนิดบูลีน ด้วยค่าตัวเลขจำนวนเต็มได้แก่ ค่าที่เป็นจริงให้มีค่าเท่ากับ 1 หรือสามารถเขียนคำว่า True แทนค่าตัวเลข และค่าที่เป็นเท็จให้มีค่าเท่ากับตัวเลข 0 หรือ False นอกจากนี้เรายังนำค่าตัวแปรชนิดบูลีนมาใช้ร่วมกับลอจิกโอเปอเรเตอร์ (Logic Operator) ซึ่งประกอบด้วย and, or, Not และรูปแบบฟังก์ชันพิเศษเฉพาะ Python ที่เรียกว่า Short-Circuit ฟังก์ชัน

การทดลองพิมพ์ค่าตัวแปรชนิดบูลีน

```
>>> a = True
>>> print(a)
True
>>> b = not a
>>> print(b)
False
>>>
```

กำหนดค่าบูลีน True ให้กับตัวแปร a

กำหนดโอเปอเรเตอร์ Not ให้กับค่าบูลีนที่อยู่ในตัวแปร a ซึ่งมีค่าเป็น True

ป้อนค่าตัวเลข b จะปรากฏเป็นค่าเท็จหรือ False ซึ่งเป็นผลมาจากการใช้โอเปอเรเตอร์ Not นั่นเอง

ตัวแปรชนิดตัวเลขที่มีทศนิยม

ชนิดตัวเลขที่มีทศนิยมสำหรับการใช้งานใน Python แบ่งออกเป็น 3 ชนิดย่อย ได้แก่

1. ตัวเลขทศนิยมชนิด Floating
2. ตัวเลขทศนิยมชนิด Complex
3. ตัวเลขทศนิยมชนิด Decimal

โดยตัวเลขทศนิยมชนิด Float และ Complex จะพร้อมใช้งานภายใน (Built-in) ร่วมกับ Python ยกเว้นชนิด Decimal ที่จำเป็นต้อง import โมดูล Decimal ก่อนใช้งาน ทั้งนี้อยู่ที่ความแตกต่างของตัวเลขที่เป็นทศนิยม ถ้าใช้คำนวณทั่วไปไม่เกี่ยวกับเงินๆทองๆ ก็ใช้ชนิด Floating แต่ถ้าเกี่ยวกับเงินทองควรใช้ชนิด Decimal ที่ต้องใช้ต่างชนิดกัน โดยเฉพาะเกี่ยวกับค่าทางการเงินเพราะเทคนิคการปัดเศษและจำนวนทศนิยมที่มีใช้ในชนิด Floating และ Decimal แตกต่างกัน สำหรับชนิด Floating อาศัยการทำงานของอัลกอริทึมที่นิยมใช้กันทั่วไปของ David Goy ซึ่งผลลัพธ์ที่ได้จากการปัดเศษทศนิยมจะมีค่าคลาดเคลื่อนนิดหน่อย แต่ก็ยังนำไปใช้งานทั่วไปได้ แต่ถ้านำไปใช้กับการเงินที่ต้องการความแม่นยำในการปัดเศษมาก แนะนำให้ใช้ชนิด Decimal ซึ่งใช้เทคนิคการปัดเศษแบบ Balance Rounding ที่มีความเร็วในการคำนวณน้อยกว่าตัวเลขทศนิยม แต่มีความคลาดเคลื่อนน้อยกว่าด้วย

math.acos(x)	math.exp(x)	math.loglp(x)
math.acosh(x)	math.fabs(x)	math.modf(x)
math.asin(x)	math.factorial(x)	math.pi
math.asinh(x)	math.floor(x)	math.pow(x,y)
math.atan(x)	math.fmod(x,y)	math.radians(d)
math.atan2(y,x)	math.frexp(x)	math.sin(x)
math.atanh(x)	math.fsum(i)	math.sinh(x)
math.ceil(x)	math.hypot(x,y)	math.aqrt(x)
math.copysign(x,y)	math.isinf(x)	math.tan(x)
math.cos(x)	math.isnan(x)	math.tanh(x)
math.cosh(x)	math.Idexp(m,e)	math.trunc(x)
math.degrees(r)	math.log(x,b)	
math.e	math.log10(x)	

ตารางแสดงกลุ่มคำสั่งเกี่ยวกับคณิตศาสตร์

ต่อไปนี้เป็นการทดลองพิมพ์ตัวเลขทศนิยม

```
>>> a = 15.20
>>> print(type(a))
<class 'float'>
>>> b,c = 5.6,7.6e-6
>>> print(a,b,c)
15.2 5.6 7.6e-06
>>> d = 15.20.hex()
>>> print(d)
0x1.e666666666666p+3
>>> e = float.fromhex(d)
>>> print(e)
15.2
>>> import math
>>> math.pi * (10**2)
314.1592653589793
>>>
```

แปลงค่า Floating ไปเป็นเลขฐานสิบหก และทำการจัดเก็บในตัวแปร d เป็นชนิดข้อความ (String)

แปลงค่าในตัวแปร d ซึ่งเก็บค่าเลขฐานสิบหกให้กลับไปเป็นค่าชนิด Float ด้วยฟังก์ชันภายในที่ชื่อว่า fromhex()

อ้างอิงกลุ่มฟังก์ชันภายในเกี่ยวกับการคำนวณภายใต้โมดูล math

ใน Python ใช้เครื่องหมาย ** สำหรับการยกกำลัง

เครื่องหมายทางการคำนวณ	ใช้งานเกี่ยวกับตัวเลข
+ , -	คำนวณบวกหรือลบตัวเลข
*	สำหรับคูณตัวเลข
**	เกี่ยวกับการหารค่าตัวเลขใน Python ใช้ได้ 2 รูปแบบ
/ , //	สำหรับการยกกำลัง เช่น 102 ก็คือ 10**2
// , %	สำหรับการตัดค่าเฉพาะเศษที่ได้จากการหารค่าตัวเลข

ตาราง แสดงเครื่องหมายการคำนวณ

ตัวแปรชนิดตัวเลขจำนวนเชิงซ้อน (Complex Number)

สำหรับการคำนวณจำนวนเชิงซ้อน เช่น 1.2 + 4j , 0.7 - 6.5j ฯลฯ ในภาษา Python ได้กำหนดตัวแปร Built-in ชนิด Imaginary ไว้รองรับ ช่วยให้ทำงานเกี่ยวกับจำนวนเชิงซ้อนได้สะดวก ลักษณะรูปแบบของตัวเลขจำนวนเชิงซ้อนจะประกอบด้วยค่า Real และค่า Imaginary

1.2 + 4j

เรียกค่า Real เรียกค่า Imaginary

รูปแสดงองค์ประกอบของตัวเลขจำนวนเชิงซ้อน Imaginary

ตามรูป ทดลองพิมพ์ตัวเลขเชิงซ้อนตามตัวอย่างต่อไปนี้

```
>>> a = 70.7 + 3.76j
>>> print(a)
(70.7+3.76j)
>>> print(a.real,a.imag)
70.7 3.76
>>> b = 30.6 + 5.12j
>>> print(a+b)
(101.30000000000001+8.879999999999999j)
>>> print(type(a))
<class 'complex'>
>>>
```

ตรวจชนิดตัวแปรใน a

ตัวแปรชนิดตัวเลขทศนิยม Decimal

โดยปกติทั่วไปถ้าเรากำหนดตัวเลขทศนิยม 3.1428 ให้กับตัวแปร สมมุติว่าชื่อตัวแปร a หรือ a = 3.1428 ในภาษา Python จะเป็นการสร้างและกำหนดชนิดตัวเลขทศนิยมแบบ Floating ให้กับตัวแปร a เมื่อไรก็ตามที่นำตัวแปรไปคำนวณแล้วเกิดค่าตัวเลขเศษขึ้นมาใหม่ วิธีการปัดเศษภายใต้ตัวแปรชนิด Floating จะผิดเพี้ยนไปทีละนิด โดยเฉพาะเมื่อมีจำนวนหลักของเศษทศนิยมมากๆ(ในตัวเลขทศนิยมชนิด Floating จำนวนของเศษทศนิยมอยู่ที่ประมาณ 17 หลัก) ดังนั้นเมื่อมีการนำค่าตัวเลขทศนิยมชนิด Floating ไปใช้งาน สมมุตินำไปใช้ร่วมกับสมการอะไรสักสมการหนึ่ง จะต้องมีการแก้ไขชดเชยตัวเลขทศนิยมที่มีโอกาสคลาดเคลื่อนด้วย ลองมาดูตัวเลขทศนิยมชนิด Decimal กันบ้าง ตัวเลขทศนิยมชนิดนี้มีกรรมวิธีการปัดเศษแบบใหม่ที่ให้ผลลัพธ์คลาดเคลื่อนน้อยกว่าชนิดตัวเลขทศนิยม Floating แต่การรองรับจำนวนตัวเลขเศษของชนิด Decimal ก็มีประมาณ 28 ตัว

ข้อจำกัดของตัวเลขทศนิยมชนิด Decimal ก็คงเป็นเรื่องของความเร็วในการคำนวณเมื่อเทียบกับชนิด Floating อีกเรื่องที่สำคัญในการใช้งานตัวเลขทศนิยมชนิด Decimal ก็คือการอ้างอิงโมดูล Decimal ก่อนใช้งานเสมอ

ต่อไปนี้เป็นการทดลองพิมพ์ตัวอย่างเกี่ยวกับตัวเลขทศนิยม Decimal

```
>>> import decimal
>>> a = 3.1428
>>> print(a)
3.1428
>>> print(type(a))
<class 'float'>
>>> b = decimal.Decimal('3.1428')
>>> print(b)
3.1428
>>> print(type(b))
<class 'decimal.Decimal'>
>>>
```

อ้างอิงการใช้ตัวเลขชนิด Decimal

พิมพ์ค่าตัวเลขในตัวแปร a

พิมพ์ชนิดค่าตัวเลขในตัวแปร a

ตัวแปรชนิดข้อความ(String)

ตัวแปรชนิดข้อความมีความจำเป็นต่อทุกภาษารวมทั้งภาษา Python ด้วย ส่วนใหญ่รูปแบบการเขียนและประกาศตัวแปรชนิดนี้จะคล้ายกับภาษาอื่น จะแตกต่างก็คงเป็นที่เครื่องหมาย เช่น ภาษา Visual Basic ใช้เครื่องหมายอัญประกาศ(Double Quote), ภาษา C หรือ C++ จะใช้เครื่องหมายฝนทอง(Single Quote) สำหรับภาษา Python สามารถใช้ได้ทั้ง 2 ภาษา(เครื่องหมายอัญประกาศหรือฝนทอง เช่น A = 'Blue Sky' หรือ A = "Blue Sky" เป็นต้น) Python กำหนดการใช้รหัสข้อความแบบ UTF-8 Unicode เพื่อรองรับข้อความที่เป็นภาษาอื่นทั่วโลก รวมทั้งภาษาอังกฤษที่เป็นพื้นฐานของการใช้ปกติ เหตุผลที่ Python เลือกรหัส Unicode เป็นหลักแทนรหัส ASCII ที่เรามักคุ้นเคยบน Microsoft Windows เนื่องจากการป้อนข้อความจากการใช้งานแต่ละประเทศ จะใช้ภาษาท้องถิ่นของประเทศนั้นๆ จำนวนพยัญชนะของแต่ละภาษาจะมีมากน้อยไม่เท่ากัน ยิ่งถ้ามีพยัญชนะมากๆ เช่น ภาษาจีน การใช้รหัส ASCII จะไม่สามารถรองรับพยัญชนะทั้งหมดได้ ทางออกก็คือต้องใช้รหัสใหม่ที่รองรับพยัญชนะได้มากขึ้น ก็คงหนีไม่พ้นรหัส Unicode เพื่อหลีกเลี่ยงปัญหาการป้อนข้อความที่มีความหลากหลายและจำนวนของพยัญชนะของแต่ละภาษาที่จะนำมาใช้กับข้อความของภาษา Python จึงได้กำหนดรหัส Unicode สำหรับใช้งานกับข้อความเป็นหลัก อีกเรื่องหนึ่งก็คือ Python สามารถทำงานได้บนหลายระบบปฏิบัติการ รวมทั้งสามารถนำมาสร้างเป็นเว็บแอพพลิเคชันที่รองรับกลุ่มภาษาที่หลากหลาย การใช้รหัส Unicode กับข้อความจะช่วยลดปัญหาเรื่องการใช้งานลง

สำหรับรูปแบบการใช้ข้อความร่วมกับตัวแปรหรือฟังก์ชันภายใน พอสรุปลักษณะการเขียนโปรแกรมของภาษา Python ได้ 5 รูปแบบ ดังนี้ (ทดลองพิมพ์คำสั่งลงใน Python IDLE)

1. **การใช้งานข้อความร่วมกับตัวแปรและฟังก์ชันภายใน**

การใช้ข้อความร่วมกับตัวแปร

```
>>> a = 'Apple'
>>> b = 'Banana'
>>> print(a,b)
Apple Banana
>>> print(type(a))
<class 'str'>
```

พิมพ์ค่าในตัวแปร a, b

ตรวจชนิดตัวแปรใน a

การใช้ข้อความร่วมกับฟังก์ชันภายในที่ชื่อ print()

```
>>> print('Python')
Python
>>> print("Python")
Python
>>>
```

2. การใช้ข้อความ + String Escape ร่วมกับตัวแปรและฟังก์ชันภายใน

กลุ่ม String Escape		
\newline	\f	\uhhhh
\\	\n	\Uhhhhhhhh
\’	\N{name}	\v
\”	\ooo	\xhh
\a	\r	
\b	\t	

ตารางแสดงกลุ่ม String Escape

การใช้ String Escape ก็เพื่อนำรหัส ASCII มาเสริมรวมเข้ากับกลุ่มของข้อความ ซึ่งรหัส ASCII ที่คุ้นเคย เช่น รหัส ASCII10 สำหรับ Linefeed หรือดูจากตารางที่ 3.5 ก็คือ การใช้รูปแบบ \n ถ้านำมาเขียนร่วมกับกลุ่มข้อความของ Python จะต้องใช้ภายในเครื่องหมายอัญประกาศ (Double Quote) หรือฝนทอง(Single Quote) ตามตัวอย่างต่อไปนี้(ทดลองพิมพ์ตัวอย่างใน Python IDLE เพื่อดูผลลัพธ์)

ตัวอย่างที่ 1 กลุ่มข้อความในตัวแปร a ที่ไม่ได้ใช้ String Escape

```
>>> a = '1.Product 2.Price'
>>> print(a)
1.Product 2.Price
>>>
```

ตัวอย่างที่ 2 กลุ่มข้อความในตัวแปร a มีการใช้ String Escape เกี่ยวกับการขึ้นบรรทัดใหม่(\n)

```
>>> a = '1.Product \n 2.Price'
>>> print(a)
1.Product
 2.Price
>>>
```

ตัวอย่างที่ 3 การใช้ String Escape ที่เกี่ยวกับการขึ้นบรรทัดใหม่ \n ร่วมกับฟังก์ชันภายในที่ชื่อ print()

```
>>> a = '1.Product \n 2.Price \n 3.Qualtity00'
>>> print(a)
1.Product
 2.Price
 3.Qualtity00
>>>
```

3. การใช้งานกลุ่มข้อความ + ค่าอาร์กิวเมนต์ (Argument) ร่วมกับฟังก์ชันภายใน

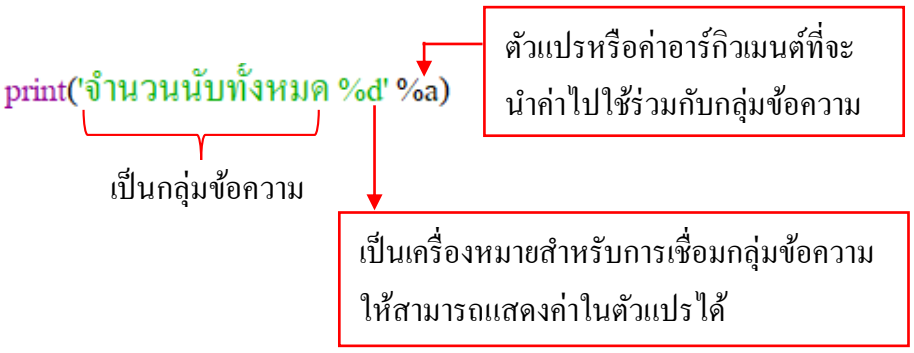
บางกรณีของการใช้กลุ่มข้อความร่วมกับฟังก์ชันภายในที่สามารถอ่านค่าจากตัวแปร แล้วนำค่าที่ได้มาใช้ร่วมกับกลุ่มข้อความได้ ตามตัวอย่างต่อไปนี้

```
บรรทัดที่ 1 >>> a = 20
บรรทัดที่ 2 >>> b,c = 40107.65,'วันอังคาร'
บรรทัดที่ 3 >>> print(a)
            20
บรรทัดที่ 4 >>> print('จำนวนนับทั้งหมด')
            จำนวนนับทั้งหมด
บรรทัดที่ 5 >>> print('จำนวนนับทั้งหมด %d' %a)
            จำนวนนับทั้งหมด 20
            >>>
```

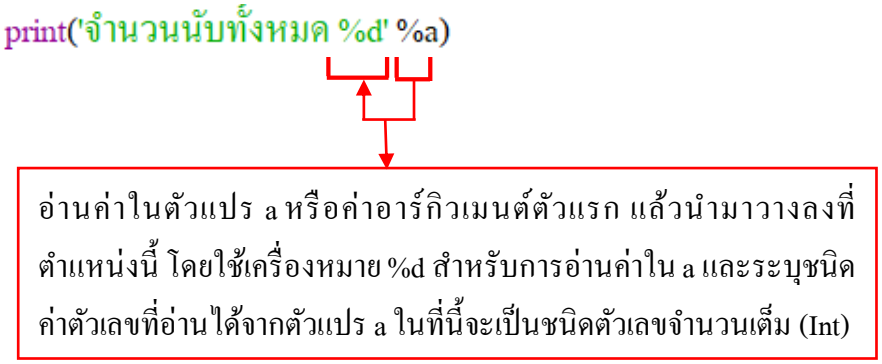
พิมพ์ค่าในตัวแปร a

พิมพ์ข้อความอย่างเดียว

<u>บรรทัดที่ 5</u> เป็นรูปแบบการใช้ข้อความผสมกับตัวแปรภายใต้การทำงานของฟังก์ชันภายในที่ชื่อ print



รูปแสดงรูปแบบการเขียนคำสั่งฟังก์ชัน print ข้อความผสมกับค่าในตัวแปร



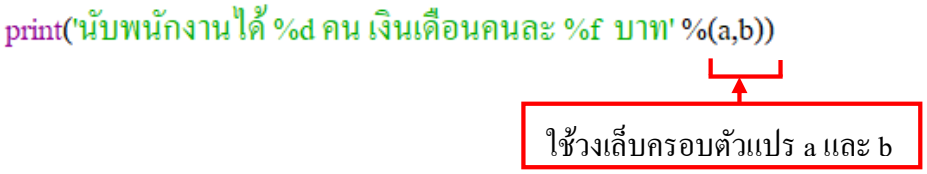
รูปแสดงตำแหน่งการวางค่าที่อ่านได้จากตัวแปร

เครื่องหมายสัญลักษณ์	ชนิดตัวแปร
%s	String
%d	Int, Decimal
%f	Float

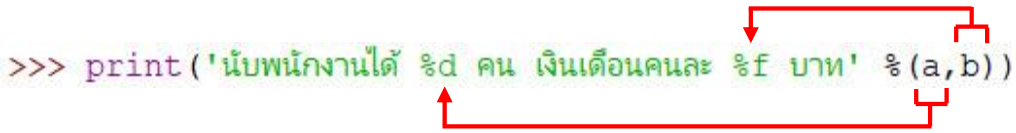
ตารางแสดงเครื่องหมายสัญลักษณ์สำหรับ
การระบุตำแหน่งในกลุ่มข้อความและระบุชนิดค่าที่อยู่ในตัวแปร

บรรทัดที่ 6 `>>> print('นับพนักงานได้ %d คน เงินเดือนคนละ %f บาท' %(a,b))`

ในบรรทัดที่ 6 เป็นการแสดงค่าตัวแปรหรืออาร์กิวเมนต์มากกว่า 1 ตัวแปร คือ a และ b



รูปแสดงการใช้ข้อความผสมกับค่าของตัวแปรที่มีมากกว่าหนึ่งตัวแปร



รูปแสดงลำดับการอ่านค่าในตัวแปรหรืออาร์กิวเมนต์ a และ b ตามลำดับ
และนำไปใช้งานร่วมกับกลุ่มข้อความตามลำดับด้วยเช่นกัน

4. การใช้งานข้อความ + String Escape + ค่าอาร์กิวเมนต์

รูปแบบนี้เป็นการนำความสามารถการทำงานของรูปแบบที่ 2 และ 3 มาใช้ร่วมกับกลุ่มข้อความตาม

ตัวอย่าง

บรรทัดที่ 1
```
>>> a = 50
```

บรรทัดที่ 2
```
>>> b , c = 5000, 'วันจันทร์'
```

บรรทัดที่ 3
```
>>> print('ซื้อทีวี %d ราคา %f ส่งของวัน %s' %(a,b,c))
ซื้อทีวี 50 ราคา 5000.000000 ส่งของวัน วันจันทร์
```

บรรทัดที่ 4
```
>>> print('ซื้อทีวี %d \n ราคา %f \n ส่งของวัน %s' %(a,b,c))
ซื้อทีวี 50
ราคา 5000.000000
ส่งของวัน วันจันทร์
>>>
```

<u>บรรทัดที่ 4</u> เป็นการผสมการใช้ String Escape โดยการใช้เครื่องหมาย \n สำหรับขึ้นบรรทัดใหม่ ร่วมกับการระบุตำแหน่งผ่านเครื่องหมายสัญลักษณ์ %d ตัวแรกที่มีในกลุ่มข้อความซึ่งแทนค่าตัวแปร a หรือค่าอาร์กิวเมนต์ a และเครื่องหมายสัญลักษณ์ %f ซึ่งแทนค่าตัวเลขและชนิดของตัวแปร b ตามลำดับ

5. การใช้งานข้อความ + String Escape + ฟังก์ชันภายใน + ค่าอาร์กิวเมนต์

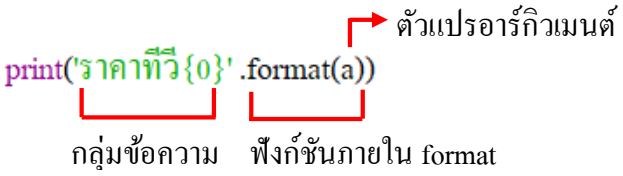
ค่าอาร์กิวเมนต์ที่เป็นกลุ่มชนิดตัวเลข เมื่อนำมาใช้ร่วมกับกลุ่มข้อความ บางกรณีก็ต้องการจัดรูปแบบเพื่อให้ง่ายต่อการใช้งาน ยกตัวอย่างตัวเลข 10000.7687 ต้องการจัดรูปแบบให้เป็น 10,000.77 เพื่อนำไปแสดงผลร่วมกับกลุ่มข้อความ โดยจะแทรกเครื่องหมาย , หรือคอมม่า(comma) ให้กับตัวเลขทุก 3 หลัก และปัดเศษให้เหลือ 2 หลัก ด้วยการใช้ฟังก์ชันภายในที่ชื่อว่า format() คราวนี้ลองมาดูตัวอย่างการใช้งาน

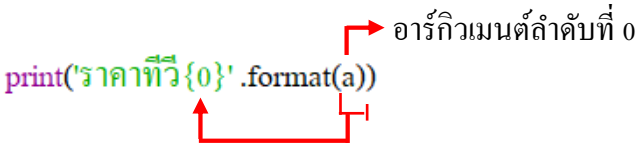
```
บรรทัดที่ 1 >>> a=10000.7687
บรรทัดที่ 2 >>> b,c=3.1428,'monday'
บรรทัดที่ 3 >>> print(a)
            10000.7687
บรรทัดที่ 4 >>> print('ราคาทีวี')
            ราคาทีวี
บรรทัดที่ 5 >>> print('ราคาทีวี{0}'.format(a))
            ราคาทีวี10000.7687
บรรทัดที่ 6 >>> print('ราคาทีวี{0}ผลิต{1}'.format(a,b))
            ราคาทีวี10000.7687ผลิต3.1428
บรรทัดที่ 7 >>>
```

พิมพ์ค่าในตัวแปร a

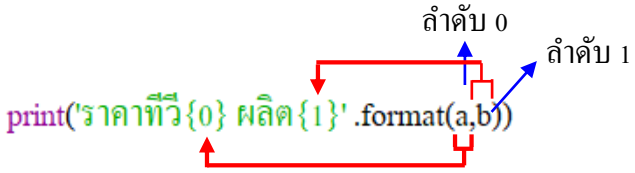
พิมพ์กลุ่มข้อความ

<u>บรรทัดที่ 6</u> เป็นการจัดรูปแบบตัวเลขในตัวแปร a ผ่านฟังก์ชันภายใน format โดยมีการนำไปใช้ร่วมกับกลุ่มข้อความเพื่อระบุตำแหน่งที่จะแทรกค่าตัวเลขในข้อความด้วยการใช้เครื่องหมาย {} สำหรับการแทรกลงในกลุ่มข้อความ





การระบุตำแหน่งค่าตัวเลขลงในกลุ่มข้อความ



ตัวแปร a เป็นลำดับอาร์กิวเมนต์ลำดับที่ 0 และนำไปแทรกลงในข้อความตรงเครื่องหมาย {0} หรือ {ลำดับของตัวแปรอาร์กิวเมนต์ ในฟังก์ชันภายใน format} ในกลุ่มข้อความ

พอจะเห็นรูปแบบและวิธีการเขียนลักษณะการอ้างอิงลำดับของตัวแปรอาร์กิวเมนต์ การใช้เครื่องหมายระบุตัวเลขที่ของอาร์กิวเมนต์สำหรับแทรกค่าตัวแปรผ่านเครื่องหมายปีกกา {} ในกลุ่มข้อความต่อไปนี้จะเป็นตัวอย่างเชิงลึกสำหรับการใช้ฟังก์ชันภายในที่ชื่อ format ร่วมกับค่าตัวแปรอาร์กิวเมนต์และการอ้างอิงในการแทรกค่าตัวแปรลงในข้อความ แบ่งออกเป็น 2 ส่วน

1. วิธีอ้างอิงการแทรกลำดับเลขที่ค่าตัวแปรอาร์กิวเมนต์ระหว่างฟังก์ชันภายใน format และกลุ่มข้อความ

การอ้างอิงการแทรกลำดับเลขที่ค่าตัวแปรอาร์กิวเมนต์กับกลุ่มข้อความ แบ่งออกเป็น 5 วิธีย่อยตามตัวอย่างต่อไปนี้

ตัวอย่าง 1.1 การอ้างอิงการแทรกลำดับเลขที่ค่าตัวแปรอาร์กิวเมนต์ด้วยหมายเลขลำดับ

```
>>> a = 10
>>> b,c=10000,'วันพุธ'
>>> print('ซื้อโทรศัพท์มือถือ {0} เครื่อง เป็นเงิน {1} จัดส่งวัน {2}' .format(a,b,c))
ซื้อโทรศัพท์มือถือ 10 เครื่อง เป็นเงิน 10000 จัดส่งวัน วันพุธ
>>> print('จัดส่งวัน {2} จำนวนทีวี {0} ยอดเงิน {1}' .format(a,b,c))
จัดส่งวัน วันพุธ จำนวนทีวี 10 ยอดเงิน 10000
>>>
```

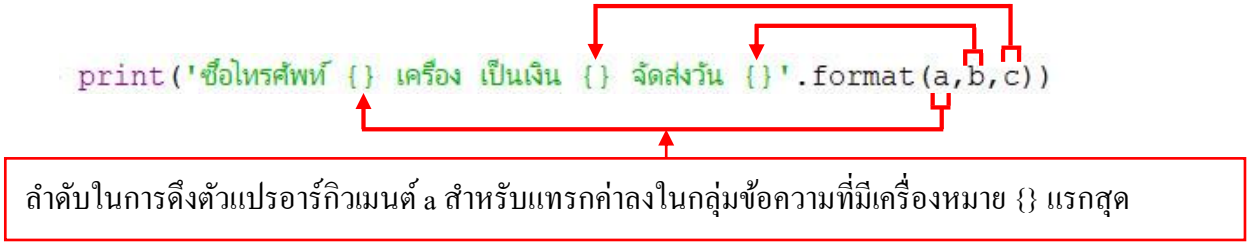
ตัวแปรอาร์กิวเมนต์ที่ชื่อ a จะมีลำดับอ้างอิงคือ 0 ตัวแปร b มีหมายเลขอ้างอิงลำดับที่ 1 และตัวแปร c จะเป็นหมายเลขอ้างอิงลำดับ 2

ประโยชน์ของการมีหมายเลขลำดับเลขที่ของตัวแปรอาร์กิวเมนต์ คือเวลาอ้างอิงเลขที่ลำดับขณะแทรกในกลุ่มข้อความ ซึ่งมีความยืดหยุ่นต่อการเรียกใช้งานตามเลขที่ลำดับที่รวมได้ตามต้องการ

ตัวอย่าง 1.2 การอ้างอิงการแทรกตัวแปรอาร์กิวเมนต์ด้วยการเรียงลำดับตามตัวแปรอาร์กิวเมนต์จากลำดับแรกไปลำดับสุดท้าย

```
บรรทัดที่ 1  >>> a = 10
บรรทัดที่ 2  >>> b,c=10000,'วันพุธ'
บรรทัดที่ 3  >>> print('ซื้อโทรศัพท์มือถือ {0} เครื่อง เป็นเงิน {1} จัดส่งวัน {2}' .format(a,b,c))
             ซื้อโทรศัพท์มือถือ 10 เครื่อง เป็นเงิน 10000 จัดส่งวัน วันพุธ
```

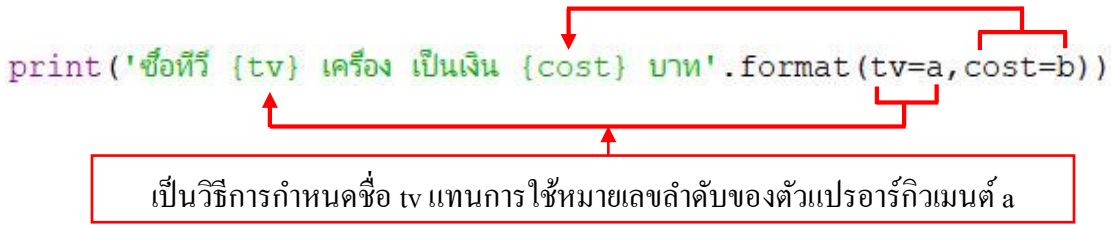
จากบรรทัดที่ 3 ลักษณะการใช้เครื่องหมาย {} ร่วมกับกลุ่มข้อความจะไม่รวมเลขที่ลำดับของตัวแปรอาร์กิวเมนต์ a,b,c ดังนั้นการดึงค่าที่อยู่ในตัวแปรอาร์กิวเมนต์ก็จะดึงค่าตามลำดับการเรียงชื่อตัวแปรอาร์กิวเมนต์ภายใต้ฟังก์ชันภายในที่ชื่อว่า format



ตัวอย่าง 1.3 การอ้างอิงแทรกค่าของตัวแปรอาร์กิวเมนต์ด้วยการกำหนดชื่อ(Field Name) แทนเลขที่ลำดับของตัวแปรอาร์กิวเมนต์

```
บรรทัดที่ 1  >>> a = 10
บรรทัดที่ 2  >>> b = 100000
บรรทัดที่ 3  >>> print('ซื้อทีวี {TV} เครื่อง เป็นเงิน {cost} บาท' .format(TV=a,cost=b))
             ซื้อทีวี 10 เครื่อง เป็นเงิน 100000 บาท
             >>>
```

จากบรรทัดที่ 3 ในส่วนท้าย .format(tv=a, cost=b) จะเป็นรูปแบบการกำหนดค่าว่า tv ให้มีค่าเท่ากับตัวแปรอาร์กิวเมนต์ a เมื่อต้องการแทรกค่าของตัวแปร a ด้วยเครื่องหมาย {} ลงในกลุ่มข้อความ ต้องระบุคำว่า tv ร่วมกับเครื่องหมาย {} เพื่อที่จะทำการอ้างอิงดึงค่าจากตัวแปรอาร์กิวเมนต์ a มาแทรกลงในกลุ่มข้อความ

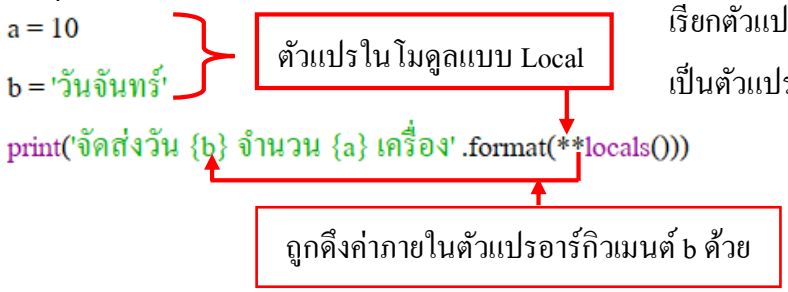


ตัวอย่าง 1.4 การอ้างอิงแทรกค่าของตัวแปรอาร์กิวเมนต์ด้วยการใช้ฟังก์ชันภายในที่ชื่อว่า locals()

```
บรรทัดที่ 1 >>> a = 10
บรรทัดที่ 2 >>> b = 'วันจันทร์'
บรรทัดที่ 3 >>> print('จัดส่งวัน {b} จำนวน {a} เครื่อง' .format(**locals()))
            จัดส่งวัน วันจันทร์ จำนวน 10 เครื่อง
            >>>
```

ตัวแปรภายในโมดูลนี้ซึ่งเป็นแบบ Local

เทคนิคการใช้ฟังก์ชัน local() ร่วมกับฟังก์ชัน format เป็นวิธีการอ้างอิงและทำการแทรกค่า(ตอนแทรกค่าลงในข้อความให้สังเกตที่เครื่องหมาย {} ในกลุ่มข้อความ) ภายในของตัวแปรอาร์กิวเมนต์โดยเป็นการอ้างจากชื่อของตัวแปรจริงๆ ภายในโมดูลที่เป็นแบบ Local ร่วมกับเครื่องหมาย {} ซึ่งจะเป็นการแทรกค่าลงตามตำแหน่งในกลุ่มข้อความ



เรียกตัวแปร Local ของโมดูลนี้มาเป็นตัวแปรอาร์กิวเมนต์โดยอัตโนมัติ

การอ้างอิงชื่อเดียวกับตัวแปรในโมดูล คือตัวแปร b สำหรับการแทรกค่าในตัวแปรอาร์กิวเมนต์ b ลงในกลุ่มข้อความตามตำแหน่งที่มีการใช้เครื่องหมาย {b} เทคนิคการใช้งานภายใต้ลักษณะการอ้างอิงค่าจากตัวแปร Local ของโมดูลแล้วนำมาเป็นค่าของตัวแปรอาร์กิวเมนต์ เพื่อที่จะนำไปแทรกค่าลงในข้อความ แบบนี้เรียกว่า Mapping Unpacking(อย่าลืมเครื่องหมาย ** นำหน้าฟังก์ชันภายใน local())

ตัวอย่าง 1.5 การอ้างอิงแทรกค่าของตัวแปรอาร์กิวเมนต์ประเภท Sequence ด้วยหมายเลขลำดับ

การนำตัวแปรอาร์กิวเมนต์ที่เป็นประเภท Sequence (คล้ายตัวแปรอาร์เรย์ในภาษาอื่นๆ) ซึ่งแบ่งย่อยได้ 3 ประเภท ได้แก่ Tuple, List และ Dictionary เป็นต้น รูปแบบของตัวแปรประเภท Sequence มีการเขียน ดังนี้

รูปแบบการเขียน	a	=	[	TV	,	10000 ,	'Monday']
วิธีการอ้างอิงค่าภายใน				a[0]		a[1]	a[2]

เมื่อนำไปใช้งานร่วมกับการผ่านค่าตัวแปรอาร์กิวเมนต์ จะมีรูปแบบการเขียนโปรแกรมใช้งานดังนี้

```
>>> a = ['TV','10000','Monday']
>>> b = 'ซื้อสินค้า {0[0]} เป็นเงิน {0[1]}'.format(a)
>>> print(b)
ซื้อสินค้า TV เป็นเงิน 10000
>>>
```

ตัวแปร a เป็นประเภท List Sequence โดยมีการระบุค่าภายในดังนี้ a[0] คือ TV , a[1] คือ 10000 , a[2] คือ Monday

จะเห็นว่ารูปแบบการแทรกค่าภายในตัวแปรอาร์กิวเมนต์ในกลุ่มข้อความ โดยปกติจะใช้เพียงเครื่องหมาย {0} ที่อ้างอิงเลขที่ลำดับตัวแปรอาร์กิวเมนต์ หมายเลข 0 หรือตัวแปรอาร์กิวเมนต์ a ซึ่งจะมีค่าที่เก็บภายในตัวแปรอาร์กิวเมนต์ a เป็นประเภท Sequence ที่สามารถจัดเก็บค่าภายในได้มากกว่าหนึ่งค่า การอ้างอิงแทรกค่าในกลุ่มข้อความจะต้องรวมให้ชัด

```
          [0]      [1]          [3]
a = ['TV' , '10000' , 'Monday']
b = 'ซื้อสินค้า {0[0]} เป็นเงิน {0[1]}' .format(a)
```

เลขลำดับอ้างอิงหมายเลข 0 ที่ติดกับเครื่องหมายปีกกา {0} จะบ่งบอกให้ทราบว่าเป็นตัวแปรอาร์กิวเมนต์ลำดับแรก หรือตัวแปรที่ชื่อว่า a แต่เผอิญตัวแปร a เป็นชนิด List Sequence ที่มีหมายเลขอ้างอิงย่อยสำหรับค่าภายในของตัวแปร a เป็น [0] คือ TV,[1] คือ 10000 ฯลฯ

การใช้เครื่องหมายแทรกก็ต้องทำการเพิ่มเติมเลขที่ลำดับย่อยภายในตัวแปรอาร์กิวเมนต์ของ a เข้าไป ดังนี้

```
          [0]      [1]          [3]
a = ['TV' , '10000' , 'Monday']
b = 'ซื้อสินค้า {0[0]} เป็นเงิน {0[1]}' .format(a)
```

อ่านค่าตามหมายเลขลำดับประเภท Sequence ของตัวแปร a โดยค่าที่ต้องการคือ TV ซึ่งตรงกับการอ้างอิงลำดับ [0]

หมายเลขอ้างอิงลำดับตัวแปรอาร์กิวเมนต์ a ใน format คือลำดับที่ {0}

2. การจัดค่าในตัวแปรอาร์กิวเมนต์ด้วยฟังก์ชันภายใน format

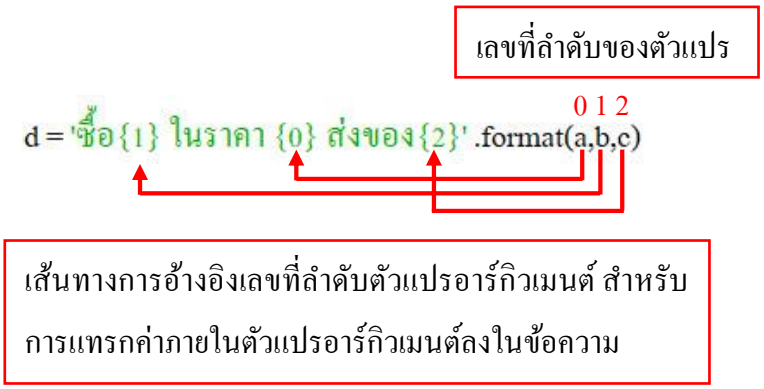
หลังจากคุ้นกับชื่อฟังก์ชันภายในที่ชื่อ format มาพอสมควร คราวนี้มาดูลักษณะการเขียนคำสั่งแสดงรูปแบบต่างๆ เพื่อให้ได้ผลลัพธ์ตามที่ต้องการ โดยส่วนใหญ่ฟังก์ชัน format จะเหมาะกับตัวแปรที่เป็นชนิดตัวเลขที่เป็นจำนวนเต็ม(Int), ทศนิยม(Float), ทศนิยมคงที่(Decimal) และชนิดข้อความ เป็นต้น

ต่อไปนี้เป็นรูปแบบการใช้งานฟังก์ชัน format กับตัวแปรชนิดต่างๆ ตามตัวอย่าง

ตัวอย่างที่ 2.1 เป็นการใช้คำสั่ง format กับตัวแปรสำหรับการแทรกค่าลงในกลุ่มข้อความด้วยการระบุเลขที่ลำดับของตัวแปรอาร์กิวเมนต์

```
บรรทัดที่ 1   >>> a,b,c = 10000, 'คอมพิวเตอร์', 'วันจันทร์'
บรรทัดที่ 2   >>> d = 'ซื้อ{1} ในราคา {0} ส่งของ{2}' .format(a,b,c)
บรรทัดที่ 3   >>> print(d)
              ซื้อคอมพิวเตอร์ ในราคา 10000 ส่งของวันจันทร์
              >>>
```

<u>จากบรรทัดที่ 2</u> ซึ่งนำฟังก์ชัน format ,ต่อท้ายกลุ่มข้อความ แล้วทำการแทรกค่าตัวแปรอาร์กิวเมนต์ด้วยเครื่องหมาย {} ในข้อความ โดยระบุเลขที่ลำดับของตัวแปรอาร์กิวเมนต์ในฟังก์ชัน



ตัวอย่างที่ 2.2 การจัดรูปแบบค่าในตัวแปรอาร์กิวเมนต์ระหว่างทำการแทรกแบบปรับแต่งการวางข้อมูลไปทางซ้าย,ขวา,ตรงกลาง,จำกัดจำนวนตัวอักษรที่ต้องการแทรก ฯลฯ

การเขียนคำสั่งฟังก์ชัน format แบบเพิ่มรูปแบบการจัดค่าข้อมูล

เลขที่ลำดับตัวแปรอาร์กิวเมนต์ : การจัดค่าข้อมูลภายในของตัวแปรอาร์กิวเมนต์ เพื่อทำการแทรกลงในข้อความ

ในตัวอย่างที่ 2.2 เมื่อมีการเพิ่มขีดความสามารถในการจัดรูปแบบข้อมูลให้กับฟังก์ชัน format การเขียนคำสั่งจะเพิ่มเติมเครื่องหมาย : (Colon) ต่อจากเลขที่ลำดับตัวแปรอาร์กิวเมนต์

```
>>> a = 'Television'
>>> b='{0}' .format(a)
>>> print(b)
Television
>>> b='{0:20}' .format(a)
>>> print(b)
Television
>>>
```

เป็นการจัดรูปแบบทั่วไปสำหรับการแทรก

เมื่อเพิ่มเครื่องหมาย :20 ต่อท้ายเลขที่ลำดับศูนย์ของตัวแปรอาร์กิวเมนต์ a ในฟังก์ชัน format จะเป็นการกำหนดจำนวนตัวอักษรอย่างน้อย 20 ตัวอักษรเสมอ เมื่อแทรกลงในข้อความ สำหรับกรณีค่าข้อมูลในตัวแปร a ตามตัวอย่างข้างล่าง ซึ่งมีข้อความ Television เมื่อนับจำนวนตัวอักษรตั้งแต่ตัวอักษร T ไปจนถึง n รวมทั้งหมด 10 ตัวอักษร เมื่อนำมาใช้กับเครื่องหมาย :20 จำนวนในตัวแปร a ที่มี 10 ตัวอักษร จึงน้อยกว่าจำนวนที่ต้องแทรกตามเครื่องหมาย :20

ภายใต้กระบวนการทำงานที่มีส่วนต่างกัน 9 ตัวอักษร จะมีการเพิ่มรหัสช่องว่างต่อท้ายภายในตัวแปร a อีก 9 ตัวอักษร เพื่อที่จะทำให้ครบตามจำนวน 20 ตัวอักษร

a =

1	2	3	4	5	6	7	8	9	10
T	e	l	e	v	i	s	i	o	n

ผลลัพธ์เมื่อใช้เครื่องหมาย :20 ร่วมกับค่าข้อมูลในตัวแปร a

a =

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	e	l	e	v	i	s	i	o	n										

เติมรหัสว่างเปล่าต่อท้ายค่าข้อมูลในตัวแปร a ให้ครบ 20 ตัวอักษร

ต่อไปเป็นการทดลองใช้ > , < , ^ ตามตัวอย่างคำสั่งต่อไปนี้

```
>>> b='{0:>20}' .format(a)
>>> print(b)
        Television
>>> b='{0:<20}' .format(a)
>>> print(b)
Television
>>> b='{0:^20}' .format(a)
>>> print(b)
    Television
>>>
```

กำหนดค่าที่แทรกให้ครบ 20 ตัวอักษร แล้วเลื่อนไปไว้ทางขวาสุด

กำหนดค่าที่แทรกให้ครบ 20 ตัวอักษร แล้วเลื่อนไปไว้ทางซ้ายสุด

กำหนดค่าที่แทรกให้ครบ 20 ตัวอักษร แล้วเลื่อนไปไว้ตรงกลาง

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
										T	e	l	e	v	i	s	i	o	n

เมื่อใช้เครื่องหมาย > ตัวอักษรจะเลื่อนทางขวาสุด

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
					T	e	l	e	v	i	s	i	o	n					

เมื่อใช้เครื่องหมาย ^ ตัวอักษรจะเลื่อนมาตรงกลาง

เทคนิคการ slicing และ Striding String สำหรับตัวแปรชนิดข้อความ

ในการเขียนภาษา Python จะมีการใช้เครื่องหมาย [] ร่วมกับตัวแปรชนิดต่างๆ ให้เห็นอยู่บ่อยครั้ง แต่ละเครื่องแบบที่นำเครื่องหมาย [] มาใช้ ก็ได้ผลลัพธ์ที่แตกต่างกัน ขึ้นอยู่กับชนิดตัวแปรหรือสถานการณ์ที่ใช้งานเครื่องหมาย [] ในขณะนั้น ตามผลลัพธ์ที่อยากได้ แต่ถ้าหากนำเครื่องหมาย [] มาใช้กับตัวแปรชนิดข้อความก็จะต้องมีคำศัพท์เชิงเทคนิคอยู่ 2 คำศัพท์ ได้แก่ slicing และ Striding String

โครงสร้างของข้อความที่จัดเก็บลงในตัวแปร ประกอบด้วย หมายเลขลำดับ(Index) ของตัวอักษรภายในกลุ่มข้อความเป็นหมายเลขกำกับการอ้างอิงถึงทุกๆ ตัวอักษร ซึ่งมีอยู่ด้วยกัน 2 ชนิด คือ 1.ชนิดหมายเลข

ลำดับ(Index) ที่มีหมายเลขทางด้านบวก(คือเริ่มจากหมายเลข 0,1,2,3,… จนถึงหมายเลขสุดท้ายเหมือนกับเริ่มจากตัวอักษรตัวแรกไปจนถึงตัวสุดท้าย) 2.ชนิดหมายเลขลำดับ(Index) ที่มีหมายเลขทางด้านลบ(คือเริ่มจากหมายเลข 0,-1,-2,-3,… จนถึงหมายเลขติดลบตัวสุดท้าย เริ่มจากตัวอักษรตัวสุดท้ายย้อนกลับไปตัวแรกสุด)

การใช้หมายเลขลำดับทางด้านลบ

A =

-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0
M	Y		N	A	M	E		I	S		J	O	H	N
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

การใช้หมายเลขลำดับทางด้านบวก

ตามโครงสร้างของข้อความที่จัดเก็บในตัวแปรของ Python รูปแบบการเขียนคำสั่งเพื่อทำการตัดคำหรือการ Slicing มีรูปแบบดังนี้

<u>รูปแบบที่ 1</u> การตัดคำด้วยการระบุตำแหน่งตัวอักษร

หมายเลข Index ทางด้านบวก

M	Y		N	A	M	E		I	S		J	O	H	N
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

วิธีการเขียนคำสั่งตัดคำ B = A[3]

ผลที่ได้จากการตัดคำเมื่อระบุ A[3] ให้กับตัวแปร B =

N
3

← หมายเลข Index ของข้อความในตัวแปร B

<u>รูปแบบที่ 2</u> การตัดคำด้วยการกำหนดจุดเริ่มต้นและการระบุความยาวจำนวนตัวอักษร

หมายเลข Index ทางด้านบวก

M	Y		N	A	M	E		I	S		J	O	H	N
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

วิธีการเขียนคำสั่งตัดคำ B = A[2:7]

ตำแหน่งหรือจุดเริ่มต้นของตัวอักษรที่ต้องการ

จำนวนตัวอักษรที่จะทำการตัดคำ โดยเริ่มนับจากตัวอักษรแรกสุดไปถึงจำนวน 7 ตัวอักษรแต่มีเงื่อนไขจากการกำหนดตำแหน่งหมายเลขลำดับที่ 2 สำหรับการนำค่าที่ทำการตัดไปจนถึงตำแหน่งสุดท้ายมาใส่ลงในตัวแปร B

ผลที่ได้จากการตัดคำเมื่อระบุ A[2:7] =

	N	A	M	E
2	3	4	5	6

<u>รูปแบบที่ 3</u> การตัดคำด้วยการกำหนดจุดเริ่มต้นและการระบุความยาวจำนวนตัวอักษร พร้อมด้วยการกำหนด Step ของตัวอักษรสำหรับการตัดคำ บางครั้งก็เรียกว่า Striding String

หมายเลข Index ทางด้านบวก

M	Y		N	A	M	E		I	S		J	O	H	N
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

จุดเริ่มต้น จุดสิ้นสุด

เขียนคำสั่ง B = A[0 : 7] = A[0 : 7 : 1]

ผลที่ได้จากการตัดคำ

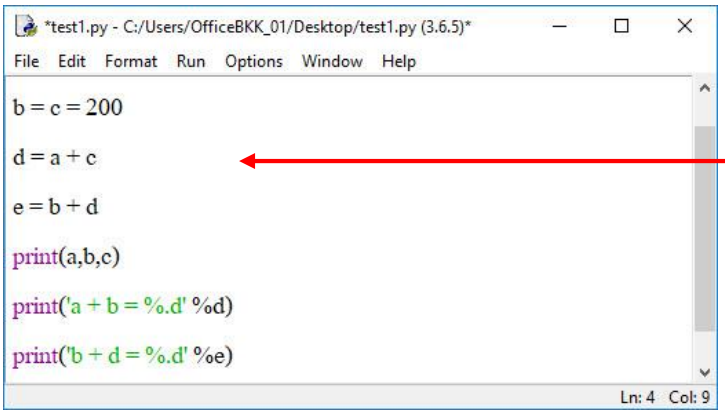
M	Y		N	A	M	E
0	1	2	3	4	5	6

→ จุดสิ้นสุดการ Step ของการตัดคำ

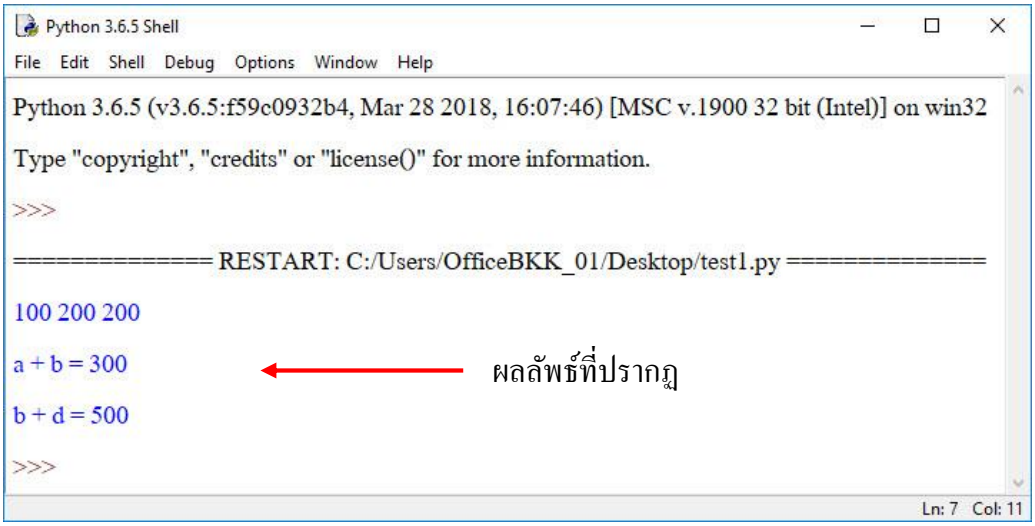
ปกติจะกำหนดการ Step ตัวอักษรหรือ Striding String ใช้ที่ 1

<u>สรุป</u> การใช้วิธีตัดคำในกลุ่มข้อความทั้ง 3 รูปแบบของภาษา Python เราเรียกว่า Slicing และลักษณะการ Step ของตัวอักษรบางครั้งเรียกว่า Striding String ซึ่งใน Python จะกำหนดไว้ทีละ 1 Step โดยอัตโนมัติ

- ทดลองโปรแกรมสำหรับการใช้ตัวแปรและชนิดตัวแปรแบบพื้นฐาน



พิมพ์คำสั่งต่อไปนี้



ผลลัพธ์ที่ปรากฏ

*** คำสั่ง print() ใช้สำหรับแสดงค่าทั่วไปหรือค่าในตัวแปร โดยมีรูปแบบการใช้คำสั่ง ดังนี้

print(ชื่อตัวแปร)

print('ข้อความ % = ชนิดตัวแปร' %(ตัวแปร))

แสดงข้อความผสมกับค่าตัวแปร

➢ การรีเซตค่าในตัวแปร

การสร้างตัวแปรในโปรแกรมบางครั้งก็มีความจำเป็นที่ต้องใส่ค่าอะไรสักอย่างเข้าไปในตัวแปรก่อนที่จะนำไปใช้งาน เพื่อให้เกิดความสมบูรณ์ตามรูปแบบการสร้างตัวแปรของ python ดังนั้นค่าที่เหมาะสมต่อการนำมาใช้งานร่วมกับการสร้างตัวแปร ควรจะเลือกค่าเริ่มต้นให้ตรงตามชนิดตัวแปรตามตัวอย่างต่อไปนี้

a = 0 ถ้าต้องการสร้างตัวแปร a สำหรับรอการใช้งานและทำงานเกี่ยวกับตัวเลข ควรกำหนดค่าให้กับตัวแปร a

a = " " ถ้าต้องการสร้างและรอการนำตัวแปรไปให้งาน

หรือ b = " " ควรกำหนดค่าข้อความแบบว่างหรือเรียกว่า Empty String(ต้องเขียนเครื่องหมาย ' ให้ติดกันเลย ห้ามกดปุ่ม Space Bar ขั้นกลางระหว่างเครื่องหมาย ' หรือ ")

c = None หากตัวแปร c เป็นชนิด Object ระหว่างรอการนำไปใช้งาน เราต้องการล้างค่า Object ที่ค้างอยู่ ด้วยการกำหนดค่า None กับตัวแปร

<u>การใช้ตัวแปรประเภท Sequence</u>

โดยปกติการใช้งานตัวแปรสำหรับเก็บค่าต่างๆ จะจัดเก็บได้ทีละ 1 ค่าต่อตัวแปร(ชนิดตัวแปรก็จะเปลี่ยนตามค่าที่จัดเก็บด้วย) เช่น a=100 เป็นต้น ถ้ามีกรณีพิเศษที่ต้องการให้ตัวแปรหนึ่งตัวจัดเก็บได้หลายค่าในเวลาเดียวกัน(คล้ายกับการใช้งานตัวแปรประเภทอาร์เรย์หรือ Array ในภาษาอื่น) Python ได้ทำการแบ่งรูปแบบตัวแปรประเภทนี้ไว้ 3 ประเภทย่อย ได้แก่

1. ตัวแปรประเภท Tuple
2.ตัวแปรประเภท List
3. ตัวแปรประเภท Dictionary

<u>ตัวอย่างที่ 1</u> การใช้ตัวแปรประเภท Tuple

```
a = ['TV',97,'DVD','VCD',12.6]
```

<u>ข้อควรสังเกต</u> ตัวแปรประเภท Tuple จะใช้เครื่องหมายวงเล็บ() และใช้เครื่องหมายคอมม่า(Comma)คั่นกลาง

ตัวอย่างที่ 2 การใช้ตัวแปรประเภท List สำหรับจัดเก็บข้อมูล

```
a = ['TV',97,'DVD','VCD',12.6]
```

ข้อควรสังเกต ตัวแปรประเภท List จะใช้เครื่องหมาย Slicing[] และคอมม่า(Comma)

ตัวอย่างที่ 3 การใช้ตัวแปรประเภท Dictionary สำหรับจัดเก็บค่าในตัวแปร

```
a = {100 : 'TV', 200:97, 300 : 'DVD',400 : 'VCD', 500 : 12.6}
```

ข้อควรสังเกต ตัวแปรประเภท Dictionary เครื่องหมายที่ใช้งานกับตัวแปรนี้คือ {} และคอมม่า (Comma)

เนื่องจากตัวแปรประเภท Tuple, List และ Dictionary สามารถเก็บค่าในตัวแปรได้มากกว่า 1 ค่า ตัวแปรประเภทเหล่านี้จึงมีโครงสร้างของการจัดเก็บข้อมูล และวิธีการกำหนดลำดับหมายเลขอ้างอิง(Index) ที่แน่นอน ตามตัวอย่างโครงสร้างต่อไปนี้

ตัวอย่าง ตัวแปร `a = {'TV','DVD',100,'fan'}`

ลำดับหมายเลขอ้างอิง →

'tv'	'dvd'	100	'fan '
0	1	2	3
A[0]	A[1]	A[2]	A[3]

หากต้องการใช้ค่า dvd ที่อยู่ในตัวแปร a จะต้องใช้หมายเลขอ้างอิงที่ 1

วิธีเขียนคำสั่ง

b = a[1] ค่าในตัวแปร b จะมีคำว่า dvd

กรณีหมายเลขอ้างอิงมีค่าติดลบ

ลำดับหมายเลขอ้างอิง →

'tv'	'dvd'	100	'fan '
0	-3	-2	-1
A[0]	A[-3]	A[-2]	A[-1]

การใช้หมายเลขอ้างอิงที่มีค่าติดลบ เป็นวิธีการเรียกค่าจากตำแหน่งสุดท้ายแล้วย้อนขึ้นไปหาค่าแรกสุด ตัวอย่างถ้าต้องการค่าที่เป็น dvd แบบหมายเลขอ้างอิงย้อนกลับหรือหมายเลขอ้างอิงติดลบ

b = a[-3]

การจัดลำดับหมายเลขอ้างอิงข้อมูลในตัวแปรประเภท Tuple และ List Sequence จะเริ่มต้นจากศูนย์และเพิ่มขึ้นไปเรื่อยๆ โดยอัตโนมัติ ตามค่าข้อมูลที่นำมาจัดเก็บ บางกรณีที่ไม่ต้องการหมายเลขอ้างอิงบอกข้อมูลอัตโนมัติที่ตัวแปร Sequence สร้างขึ้นเองในภาษา python ก็มีตัวแปร Dictionary Sequence เพื่อรองรับการกำหนดหมายเลขอ้างอิง พร้อมค่าข้อมูลจากผู้ใช้งานเอง

ตัวแปร Tuple a = ('TV', 'DVD', 5.67, 'fan')

ลำดับหมายเลขอ้างอิง →

'tv'	'dvd'	100	'fan '	ค่าข้อมูล
0	1	2	3	Positive Index
0	-3	-2	-1	Negative Index

การใช้งานตัวแปร Tuple b = a[1]

ผลลัพธ์ใน b คือ dvd

ตัวแปร List a = ('TV', 'DVD', 5.67, 'fan')

ลำดับหมายเลขอ้างอิง →

'tv'	'dvd'	100	'fan '	ค่าข้อมูล
0	1	2	3	Positive Index
0	-3	-2	-1	Negative Index

การใช้งานตัวแปร Tuple b = a[1]

ผลลัพธ์ใน b คือ dvd

ตัวแปร Dictionary a = {100 : 'tv', 200 : 'dvd', 300 : 5.67,400 : 'fan'}

ลำดับหมายเลขอ้างอิง →

'tv'	'dvd'	100	'fan '	ค่าข้อมูล
100	200	300	400	Index

การใช้งานตัวแปร Tuple b = a[200]

หรือ b = a.get(200)

ผลลัพธ์ใน b คือ dvd

แสดงการทำงานของลำดับหมายเลขอ้างอิงของตัวแปรประเภท Sequence

เทคนิคการใช้งานตัวแปรประเภท Sequence ในลักษณะ Nested

นอกจากความสามารถในการจัดการโครงสร้างของตัวแปรประเภท Sequence เพื่อรองรับการจัดเก็บข้อมูลแบบลักษณะทั่วไปแล้ว ยังมีการจัดเรียงโครงสร้างข้อมูลแบบซ้อนเชิงลึก หรือบางครั้งก็เรียกว่า Nested Sequence อีกด้วย

ตัวอย่างการสร้าง : การใช้ตัวแปรประเภท Sequence แบบเก็บค่าทั่วไป

```
Tuple a = ('TV','DVD',5.67,'FAN')
List b = ['TV','DVD',5.67,'FAN']
Dictionary c = {100 : 'TV', 200 : 'DVD', 300 : 5.67, 400 : 'FAN'}
```

ตัวอย่างการสร้าง : การใช้ตัวแปรประเภท Sequence แบบ Nested Sequence

```
a = ('TV',('SONY DVD', 'LG DVD', 'SOKEN DVD', 5.67, 'FAN')
b = ('TV',('SONY DVD', 'LG DVD', 'SOKEN DVD', 5.67, 'FAN')
c = {100 : 'TV', {200 : 'SONY DVD', 201 : 'LG DVD' 202 : 'SOKEN DVD'}, 5.67, 'FAN'}
```

สังเกต การจัดเก็บข้อมูลเชิงลึกในลำดับหมายเลขอ้างอิงที่ 1 ซึ่งจะมีรายการข้อมูล DVD หลายยี่ห้อ

ตัวอย่างวิธีการอ่านค่า : ตัวแปรประเภท Sequence แบบค่าทั่วไป

รูปแบบการเขียนคำสั่ง `a = ('TV',('DVD',5.67,'FAN')`

a =	'TV'	'DVD'	5.67	'FAN'
ลำดับหมายเลขอ้างอิงแบบ Forward	0	1	2	3

ตัวอย่างวิธีการอ่านค่า : ตัวแปรประเภท Nested แบบค่าทั่วไป

รูปแบบการเขียนคำสั่ง `a = ('TV',('SONY DVD', 'LG DVD', 'SOKEN DVD', 5.67, 'FAN')`

	0	1	2	3
a =	'TV'	'DVD' a[1]	5.67	'FAN'

ลำดับหมายเลขอ้างอิงขั้นที่ 1

ขั้นที่ 2 ภายใต้ตัวแปร a[1]

SONY DVD	LG DVD	FAN
0	1	2

b = a [1][1] ผลลัพธ์ใน b จะมีค่าเท่ากับ LG DVD

ตัวอย่างทดลองการ Nested ค่าในตัวแปรประเภท Sequence พิมพ์คำสั่งต่อไปนี้

```
>>> a = ('APPLE JOICE', ('ORANGE JOICE', 'LEMON JOICE') ,'MANGO JOICE')
>>> print(a[0])
APPLE JOICE
>>> print(a[1])
('ORANGE JOICE', 'LEMON JOICE')
>>> print(a[1][0])
ORANGE JOICE
>>> print(a[1][1])
LEMON JOICE
>>>
```

รูปแบบการเขียนตัวแปรประเภท Sequence

ลักษณะการเขียนตัวแปรประเภท Sequence ของภาษา Python ที่มีถึง 3 ประเภทย่อย ซึ่งที่บ่งบอกถึงประเภทย่อยนั่น จะใช้เครื่องหมายร่วมกับการกำหนดค่าต่างๆ ตามตารางที่ 3.7 ดังนี้

	ชื่อประเภทย่อย	สัญลักษณ์สำหรับการใช้งาน	ตัวอย่าง
1	Tuple	()	a = ('TV', 'DVD', 'VCD')
2	List	[]	a = ['TV', 'DVD', 'VCD']
3	Dictionary	{}	a = {100:'TV', 200'DVD'}

ตารางที่ 3.7 การใช้เครื่องหมายสำหรับการจัดเก็บข้อมูลลงในตัวแปร Sequence แต่ละประเภทย่อย

หลังจากเข้าใจการจัดเก็บค่าข้อมูลลงตัวแปร Sequence ตามประเภทต่างๆ ต่อไปนี้ การอ่านค่าข้อมูลในตัวแปร Sequence ในแต่ละประเภท ตามตารางที่ 3.8 ดังนี้

ชื่อประเภทย่อย	สัญลักษณ์สำหรับการใช้งาน	ตัวอย่าง
Tuple	a = ('TV', 'DVD', 'VCD')	b = a[0] ได้ค่า tv
List	a = ['TV', 'DVD', 'VCD']	b = a[0] ได้ค่า tv
Dictionary	a = {100:'TV', 200'DVD', 300: 'VCD'}	b = a[100] ได้ค่า tv

ตารางที่ 3.8 การใช้เครื่องหมาย [] สำหรับการอ่านค่าข้อมูลจากตัวแปรประเภทแบบ Sequence

การอ่านค่าในตัวแปรประเภท Sequence ของภาษา Python จะใช้เครื่องหมายการอ่านค่าเพียงเครื่องหมายเดียวคือ [] โดยระบุหมายเลขลำดับของค่าที่จัดเก็บตามลำดับโครงสร้างที่กล่าวไว้ตอนต้นหัวข้อนี้ ยกเว้นตัวแปรที่เป็น Dictionary ที่ใช้ค่า Key แทนหมายเลขลำดับค่าข้อมูล

ตัวอย่างคำสั่งการใช้ตัวแปร Sequence แต่ละประเภท พิมพ์ตามคำสั่งเพื่อดูผลลัพธ์

```
>>> a = ('TV','DVD','VCD',)
>>> print(a)
('TV', 'DVD', 'VCD')
>>> print(a[0], a[1], a[2])
TV DVD VCD
>>> print(type(a))
<class 'tuple'>
>>> b = ['APPLE', 'ORANGE', 'BANANA']
>>> print(b)
['APPLE', 'ORANGE', 'BANANA']
>>> print(b[0], b[1], b[2])
APPLE ORANGE BANANA
>>> print(type(b))
<class 'list'>
>>> c = {100 : 'CAR', 200 : 'VAN', 300 : 'MOTORCYCLE'}
>>> print(c)
{100: 'CAR', 200: 'VAN', 300: 'MOTORCYCLE'}
>>> print(c[100],c[200],c[300])
CAR VAN MOTORCYCLE
>>> print(type(c))
<class 'dict'>
>>>
```

กำหนดให้ตัวแปร a ซึ่งเป็นแบบ Tuple Sequence

กำหนดให้ค่าตัวแปร b เป็นแบบ List

กำหนดค่าลงในตัวแปร c เป็นแบบ Dictionary

ตัวแปรชนิด Tuple

ตัวแปรชนิดนี้จัดอยู่ในกลุ่มตัวแปรประเภท Sequence ที่สามารถเก็บค่าต่างๆ คล้ายกับตัวแปรอาร์เรย์ (Array) ของภาษาอื่นๆ ลงในตัวแปรตัวเดียว โดยค่าที่จัดเก็บจะจัดเรียงตามลำดับการใส่ค่าข้อมูลไว้ในตัวแปร และจะกำหนดหมายเลขลำดับ(Index) ให้กับค่าในโครงสร้างของตัวแปร แต่เนื่องจากตัวแปรชนิด Tuple จัดเป็นโครงสร้างแบบ Immutable(Immutable เป็นโครงสร้างที่เมื่อมีการกำหนดค่าข้อมูลลงในตัวแปรที่มีโครงสร้างแบบนี้เรียบร้อยแล้ว ขณะนำตัวแปรที่มีโครงสร้างแบบ Immutable ไปใช้ จะไม่สามารถเพิ่มหรือลบค่าข้อมูลที่จัดเก็บในโครงสร้างนี้ได้ ยกเว้นการแก้ไขค่าข้อมูลที่มีอยู่แล้วในตัวแปร)

```
>>> a = ('แอปเปิ้ล 5 ลูก', 'ส้มโอ 5 ลูก', 'ทุเรียน 1 ลูก')
>>> print(type(a))
<class 'tuple'>
>>> print(a)
('แอปเปิ้ล 5 ลูก', 'ส้มโอ 5 ลูก', 'ทุเรียน 1 ลูก')
>>> print(a[0])
แอปเปิ้ล 5 ลูก
>>> b = a[0]
>>> print(b)
แอปเปิ้ล 5 ลูก
>>> print(a[-2])
ส้มโอ 5 ลูก
>>> print(a[:1])
('แอปเปิ้ล 5 ลูก',)
>>> print(a[1:])
('ส้มโอ 5 ลูก', 'ทุเรียน 1 ลูก')
>>>
```

พิมพ์ค่าข้อมูลที่ระบุตามหมายเลขลำดับ(Index)

เป็นการให้หมายเลขลำดับ(Index) ทางด้านลบ

บรรทัดที่ 1 เป็นการกำหนดค่าข้อมูลให้ตัวแปร a เนื่องจากการใช้เครื่องหมาย() จะกำหนดโครงสร้างการจัดเก็บค่าข้อมูลที่เป็น Immutable โดยอัตโนมัติ

กรณีมีข้อมูลเพียงค่าเดียว แต่ต้องการกำหนดลงในตัวแปรให้เป็นแบบ Tuple จะมีเทคนิคการกำหนดค่าข้อมูลตามรูปแบบการเขียนดังนี้

```
a = ('แอปเปิ้ล 5 ลูก',)
```

ข้อสังเกต การใช้เครื่องหมาย , (Comma) ต่อท้ายก่อนวงเล็บปิด จะเป็นสิ่งที่บ่งบอกให้ Python รู้ว่าตัวแปร a จะต้องเป็นชนิด Tuple เท่านั้น เมื่อเรียกใช้งานก็ต้องเขียนคำสั่งดังนี้

b = a[0]

ถ้าลืมเครื่องหมาย , (Comma) ต่อท้ายจะเกิดอะไรขึ้นกับตัวแปร a คำตอบก็คือ ตัวแปร a จะเป็นตัวแปรปกติ เก็บค่าข้อมูลได้ทีละ 1 ค่า และเป็นข้อมูลชนิดแบบข้อความทั่วไป

```
>>> a = ('ปลาทู 2 เข่ง',)
>>> print(a[0])
ปลาทู 2 เข่ง
>>> b = ('ปลาเก๋า 1 ตัว')
>>> print(b)
ปลาเก๋า 1 ตัว
>>> print('a คือ %s  b คือ %s ' %(type(a), type(b)))
a คือ <class 'tuple'>  b คือ <class 'str'>
>>> print(a[0])
ปลาทู 2 เข่ง
>>> print(b[0])
ป
>>>
```

มีเครื่องหมาย Comma ตัวแปร a เป็นชนิด Tuple

ไม่มีเครื่องหมาย Comma เป็นตัวแปรแบบทั่วไป

จะได้ตัวอักษร ‘ป’ ซึ่งเป็นอักษรตัวแรก เนื่องจากตัวแปร b เป็นตัวแปรแบบ String ไม่ใช่ตัวแปรแบบ Tuple

ตัวแปรชนิด List

หนึ่งในตัวแปรประเภท Sequence ที่มีความยืดหยุ่นในการกำหนดค่าข้อมูลลงในตัวแปร และขณะนำตัวแปรชนิดนี้ไปใช้งานก็สามารถเพิ่ม,ลบ,แก้ไขค่าข้อมูลในตัวแปรได้ตลอดเวลา ด้วยโครงสร้างการจัดเก็บค่าของตัวแปร List เป็นแบบ Mutable(Mutable เป็นโครงสร้างที่เหมาะกับค่าข้อมูลที่มีการเปลี่ยนแปลงในลักษณะการเพิ่ม,ลบ,หรือแก้ไข ขณะนำไปใช้งาน)

a = ['ปลาทู', 1000, 'ปลาสลิด'], 'กิโลกรัม']

a =

-4	-3	-2	-1
'ปลาทู'	1000	'ปลาสลิด'	'กิโลกรัม'
0	1	2	3

หมายเลขลำดับ Index ทางด้านลบ

หมายเลขลำดับ Index ทางด้านบวก

รูปที่ 3.10 แสดงโครงสร้างการจัดค่าข้อมูลของตัวแปรชนิด List

ลักษณะการจัดเก็บค่าของตัวแปรชนิด List จะเรียงตามลำดับการนำค่าข้อมูลเข้าไปเก็บภายในตัวแปร และกำหนดหมายเลข Index สำหรับข้อมูลแต่ละค่าที่จัดเก็บ ตามโครงสร้างของตัวแปรในรูปที่ 3.10 เมื่อต้องการค่าในตัวแปรขณะใช้งานจะใช้วิธีอ้างหมายเลขลำดับ Index ของค่าข้อมูลนั้นๆ

ตัวอย่างการสร้างตัวแปรชนิด List และเรียกใช้งานค่าภายในตัวแปร

```
>>> a = ['ผักบุ้ง', 10, 'กะหล่ำปลี', 'คะน้า']
>>> print(type(a))
<class 'list'>
>>> print(a)
['ผักบุ้ง', 10, 'กะหล่ำปลี', 'คะน้า']
>>> print(a[0], a[1])
ผักบุ้ง 10
>>> a[0]='ผักบุ้งหมดแล้ว'
>>> print(a)
['ผักบุ้งหมดแล้ว', 10, 'กะหล่ำปลี', 'คะน้า']
>>> print(a[0])
ผักบุ้งหมดแล้ว
>>> print(a[:2])
['ผักบุ้งหมดแล้ว', 10]
>>> print(a[1:])
[10, 'กะหล่ำปลี', 'คะน้า']
>>>
```

กำหนดค่าข้อมูลเริ่มต้นลงในตัวแปร a ที่เป็นชนิด List

ทำการแก้ไขค่าในตัวแปร List

ในการเพิ่มหรือลบค่าข้อมูลภายในตัวแปร List จำเป็นต้องเรียกใช้คำสั่งพิเศษ โดยเรียกคำสั่งพิเศษนี้ว่า Method ต่อไปนี้เป็นตารางกลุ่ม Method ที่มีใช้ตัวแปร List

รูปแบบ	คำอธิบาย
A.ppend(x)	นำค่าในตัวแปร x ไปเพิ่มที่ตำแหน่งท้ายสุดของกลุ่มข้อมูลในตัวแปร a
A.count(x)	นับจำนวนกลุ่มข้อมูลที่อยู่ในตัวแปร a
A.extend(m) A+=m	นำค่าในตัวแปร m ไปเพิ่มที่ตำแหน่งท้ายสุดของกลุ่มข้อมูลในตัวแปร a
A.index(x,start,end)	อ่านค่าในตัวแปร a ตามตำแหน่งของ Index ที่ระบุไว้ในตัวแปร x
A.index(i,x)	นำค่าในตัวแปร x เพิ่มที่ตำแหน่งของ Index ตามที่ระบุในตัวแปร i
A.pop()	อ่านค่าที่ตำแหน่งท้ายสุดของตัวแปร a และลบค่าดังกล่าวออกจากตัวแปร a ด้วย
A.pop(i)	อ่านค่าที่ตำแหน่งท้ายสุดของตัวแปร i ระบุ Index ของตัวแปร a และลบค่าดังกล่าวออกจากตัวแปร a ด้วย
A.remove(x)	ลบค่าตามที่ระบุในตัวแปร x จากกลุ่มข้อมูลในตัวแปร a ถ้าหาค่าดังกล่าวไม่เจอ จะเกิดข้อผิดพลาดในกลุ่มของ Value Error ขึ้น
A. reverse()	เรียงค่ากลุ่มข้อมูลจากท้ายสุดไปตำแหน่งแรกสุด
A.sort(...)	เรียงค่ากลุ่มข้อมูลจากน้อยไปมาก

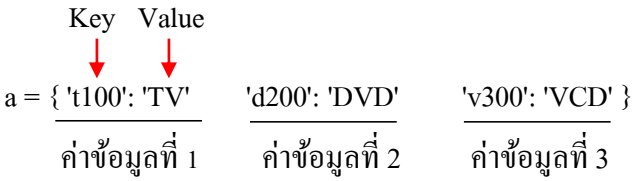
ตัวแปรชนิด Dictionary

สมมุติว่าเราต้องการเก็บค่าข้อมูลลงในตัวแปรประเภท Sequence โดยมีเงื่อนไขเกี่ยวกับค่าข้อมูลที่ประกอบด้วยรหัสของข้อมูลที่กำหนดพร้อมกับค่าข้อมูลที่ต้องการจัดเก็บ ตามตารางที่ 3.10 แสดงองค์ประกอบของค่าข้อมูล

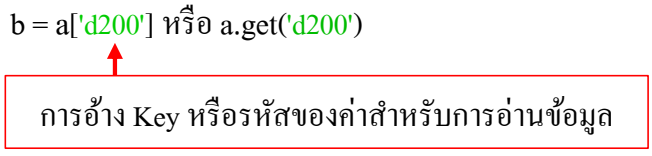
ลักษณะองค์ประกอบ	รหัสของข้อมูลที่กำหนดเอง : ค่าข้อมูลที่ต้องการจัดเก็บ
ชื่อทางเทคนิคใน Python	Key : Value
ตัวอย่างค่าข้อมูล	'T100' : 'TV'

ตารางที่ 3.10 แสดงองค์ประกอบค่าข้อมูลสำหรับใช้กับตัวแปรชนิด Dictionary

เมื่อถูกนำมาใช้ร่วมกับตัวแปรประเภท Sequence ด้วยค่าข้อมูล 1 ค่า แต่ประกอบด้วยรหัสและค่าข้อมูลรวมกัน นอกจากนี้ยังมีเงื่อนไขว่าการนำมาใช้งานหรืออ่านค่าข้อมูลภายในตัวแปรต้องใช้วิธีการระบุรหัสที่เราใส่ลงไปพร้อมกับค่าข้อมูลแทนการใช้หมายเลขลำดับ Index ปกติที่เราคุ้นเคยกับตัวแปร Sequence ชนิด Tuple หรือ List พอเจอเงื่อนไขข้อหลังนี้ ก็คงไม่มีทางเลือกที่จะต้องใช้ตัวแปร Sequence ชนิด Dictionary ที่เหมาะต่อการลองรับค่าข้อมูลที่มีโครงสร้างพิเศษและเรียกใช้งานด้วยรหัสที่มาพร้อมกับค่าข้อมูล สำหรับการเขียนคำสั่งของ Python ที่จะกำหนดตัวแปรให้เป็นชนิด Dictionary จะใช้เครื่องหมาย {} ร่วมกับค่าข้อมูล



แสดงการกำหนดค่าข้อมูลที่มีรหัสและค่าข้อมูลในตัวเองลงในตัวแปร a ซึ่งเป็นชนิด Dictionary



แสดงวิธีใช้งานตัวแปร a เมื่อต้องการค่าข้อมูลที่เก็บไว้ภายในตัวแปร

เนื่องจากตัวแปรชนิด Dictionary ใช้โครงสร้างค่าข้อมูลเป็นแบบ Mutable ที่มีคุณสมบัติในการเพิ่ม,ลบ,และแก้ไขค่าภายในตัวแปรโดยการใช้ Method ของตัวแปรชนิด Dictionary ช่วยในการทำงานตามคุณสมบัติต่างๆ ดังตาราง 3.11

รูปแบบ	คำอธิบาย
d.clear()	ลบกลุ่มข้อมูลภายในตัวแปร d ทั้งหมด
d.copy()	สำหรับการ copy กลุ่มข้อมูลในตัวแปร d ไปยังตัวแปรอีกตัวหนึ่ง
d.fromkeys(s,v)	ตรวจหาค่า key ของค่าข้อมูลที่ระบุในตัวแปร s โดยค่า key ที่ได้ เก็บไว้ในตัวแปร v
d.get(k)	อ่านค่าข้อมูลตามค่า key ที่ระบุในตัวแปร k
d.get(k,v)	อ่านค่าข้อมูลตามค่า key ที่ระบุในตัวแปร k
d.items()	อ่านค่าข้อมูลและค่า key จากกลุ่มข้อมูลภายในตัวแปร d ทั้งหมด
d.keys()	อ่านเฉพาะค่า key ของกลุ่มข้อมูลภายในตัวแปร
d.pop(k)	อ่านค่าที่ตำแหน่งที่ตัวแปร k ระบุ Index ของตัวแปร d และลบค่าดังกล่าวออกจากตัวแปร d ด้วย
d.pop(k,v)	อ่านค่าที่ตำแหน่งที่ตัวแปร k ระบุ Index ของตัวแปร d และลบค่าดังกล่าวออกจากตัวแปร d ด้วย
d.popitem()	อ่านค่าข้อมูลที่ตำแหน่งแรกสุด และลบค่าดังกล่าวออกจากตัวแปร d ด้วย
d.setdefault(k,v)	กำหนดค่าข้อมูลภายในตัวแปร d เป็นค่า default สำหรับการอ่านค่าจาก g.get()
d.update(a)	นำค่าในตัวแปร a ที่เป็นชนิด Dictionary เช่นเดียวกับตัวแปร d ไปแก้ไขข้อมูลในตัวแปร d
d.values()	อ่านเฉพาะค่าของกลุ่มข้อมูลภายในตัวแปร d ทั้งหมด

ตารางที่ 3.11 Method สำหรับตัวแปรชนิด Dictionary

ต่อไปนี้เป็นการทดลองตัวอย่างคำสั่งของตัวแปร Dictionary

```
>>> a = {'t100' : 'ส้มตำ', 'k200' : 'หมูกะทะ', 'p300' : 'ข้าวผัด'}
>>> print(type(a))
<class 'dict'>
>>> print(a)
{'t100': 'ส้มตำ', 'k200': 'หมูกะทะ', 'p300': 'ข้าวผัด'}
>>> print(a['p300'])
ข้าวผัด
>>> print(a.get('k200'))
หมูกะทะ
>>> a['k200'] = 'หมูกะทะหมดแล้ว'
>>> print(a['k200'])
หมูกะทะหมดแล้ว
>>>
```

เป็นการใช้งานตัวแปรโดยการแก้ไขค่าภายในผ่านรหัสปัจจุบัน

ตัวอย่าง การใช้ตัวแปร Dictionary ร่วมกับกลุ่ม Method

```
>>> a = {100 : 'ทะเล', 200 : 'ภูเขา', 300 : 'ป่าไม้'}
>>> print(a)
{100: 'ทะเล', 200: 'ภูเขา', 300: 'ป่าไม้'}
>>> b = {400 : 'น้ำตก'}
>>> a.update(b)
>>> print(a)
{100: 'ทะเล', 200: 'ภูเขา', 300: 'ป่าไม้', 400: 'น้ำตก'}
>>> a.pop(100)
'ทะเล'
>>> print(a)
{200: 'ภูเขา', 300: 'ป่าไม้', 400: 'น้ำตก'}
>>> a[100] = 'ทะเลอันดามัน'
>>> print(a)
{200: 'ภูเขา', 300: 'ป่าไม้', 400: 'น้ำตก', 100: 'ทะเลอันดามัน'}
>>> print(a.keys())
dict_keys([200, 300, 400, 100])
>>> print(a.values())
dict_values(['ภูเขา', 'ป่าไม้', 'น้ำตก', 'ทะเลอันดามัน'])
>>> a.clear()
>>> print(a)
{}
>>>
```

ประกาศค่าข้อมูลใหม่ในตัวแปร b

นำค่าข้อมูลจากตัวแปร b เพิ่มเข้าไปในตัวแปร a

ลบค่าข้อมูลที่ใช้รหัส(Key) ที่มีค่า 100

แก้ไขค่าในตัวแปร a

ลบทุกค่าข้อมูลในตัวแปร a

<u>สรุป</u> ข้อแตกต่างของตัวแปรชนิด Tuple, List และ Dictionary สำหรับการเลือกนำไปใช้งานให้เหมาะสมตามลักษณะของค่าข้อมูลที่นำมาใช้งานร่วมกับตัวแปรทั้ง 3 ชนิด ตามตารางที่ 3.12 ดังนี้

ชนิดตัวแปร	ใช้หมายเลข Index	กำหนดคีย์สำหรับข้อมูล	เพิ่มข้อมูลสำหรับใช้งานตัวแปร	แก้ไขค่าข้อมูลขณะใช้งาน	ลบค่าข้อมูลขณะใช้งาน	ใช้โครงสร้างการเก็บข้อมูล	ความเร็วในการทำงาน	ชนิดตัวแปรที่ใช้ได้
Tuple	✓			✓		Immutable	เร็วที่สุด	ทุกชนิด
List	✓		✓	✓	✓	Mutable	กลาง	ทุกชนิด
Dictionary		✓	✓	✓	✓	Mutable	กลาง	ทุกชนิด

ตาราง แสดงคุณสมบัติของตัวแปรชนิด Tuple, List และ Dictionary

➢ การใช้โอเปอเรเตอร์ (Operator) กับตัวแปร

การพัฒนาโปรแกรม เช่น a+b,c ^2 เป็นต้น รูปแบบการเขียนคำสั่งจะประกอบด้วยตัวแปรชนิดต่างๆ นำมาคำนวณร่วมกัน สำหรับนักพัฒนาโปรแกรมทั้งหลายจะเห็นว่าเครื่องหมายที่เกี่ยวข้องกับการคำนวณที่เป็นพื้นฐาน ได้แก่

รูปแบบการเขียน	ความหมาย (กำหนด A=10,B=5)	ตัวอย่าง
A+B	ทำการบวกค่าในตัวแปร A และ B	10 + 5 = 15
A-B	นำค่าในตัวแปร A ลบกับตัวแปร B	10 – 5 = 5
A*B	คูณค่าในตัวแปร A และ B เข้าด้วยกัน	10 * 5 = 50
A/B หรือ A // B	หารค่าตัวแปร A ด้วย B	10/5 = 2
A ^ B	ใช้ xor ระหว่างตัวแปร A และ B	10 ^ 5 = 2
A % B	หาค่าเศษจากการหารตัวแปร A ด้วย B	10 % 5 = 0
A|B	ทำการ or ระหว่างค่าในตัวแปร A และ B	10 | 5 = 5
A ** B	ยกกำลังค่าในตัวแปร A ยกกำลังด้วย B	10 ** 5 = 100000
A & B	ทำการ and ระหว่างค่าของ A และ B	10 & 5 = 0
A >> B	เลื่อนค่าบิตในตัวแปร A ไปทางขวาด้วย B	10 >> 5 = 0
A << B	เลื่อนค่าบิตในตัวแปร A ไปทางซ้ายด้วย B	10 << 5 = 320
~A	เป็นการ Not กลับค่าในตัวแปร A	~ 5
-A	ค่าในตัวแปร A ติดลบ	-10

เรียกเครื่องหมายเหล่านั้นว่า โอเปอเรเตอร์(Operator) ลักษณะการนำเครื่องหมายโอเปอเรเตอร์ไปใช้งาน จะแบ่งตามชนิดของและตัวแปรและสถานการณ์ ซึ่งบางครั้งก็เป็นเครื่องหมายโอเปอเรเตอร์เดียวกันแต่ลักษณะแตกต่างกัน ยกตัวอย่างคำสั่งของ Python ต่อไปนี้

```
>>> a = 10
>>> b = 20
>>> c = a | b
>>> print(c)
30
>>> x = {'แอปเปิ้ล','ส้มโอ','เงาะ','มะม่วง'}
>>> y = {'เนื้อหมู','เนื้อวัว','เนื้อปลา','กุ้งสด'}
>>> z = x | y
>>> print(z)
{'เนื้อวัว', 'เนื้อหมู', 'เงาะ', 'แอปเปิ้ล', 'มะม่วง', 'เนื้อปลา', 'กุ้งสด', 'ส้มโอ'}
>>>
```

กำหนดค่า 10 และ 20 ลงในตัวแปรที่เป็นชนิด Int

ทำการ or ค่าภายในในระดับบิตของตัวแปรทั้ง a,b ด้วยเครื่องหมาย |

แสดงผลลัพธ์ที่เกิดจากการ or ค่าของตัวแปร a และ b

ทำการเซตค่าในกลุ่มข้อมูลของตัวแปร a และ b ด้วยการ Union ด้วยเครื่องหมาย |

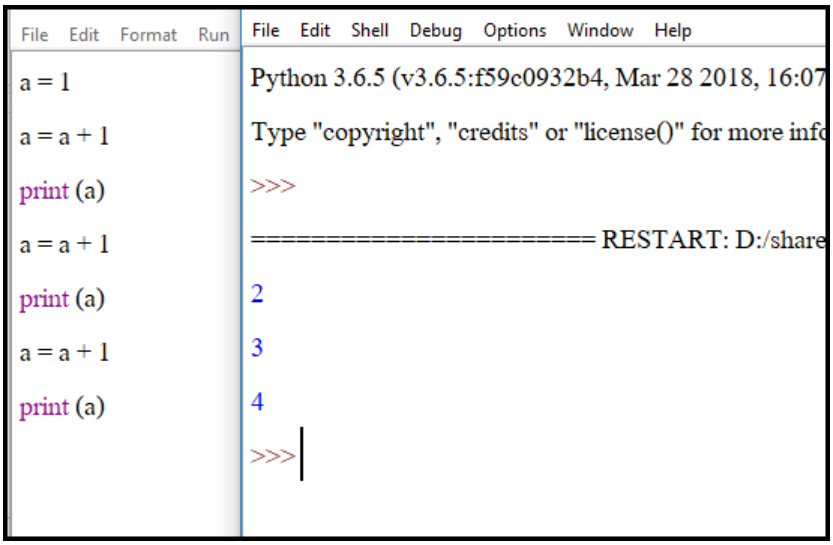
จะเห็นว่า กลุ่มตัวแปร a,b,c มีการใช้เครื่องหมายโอเปอเรเตอร์ | เพื่อทำการ or ข้อมูลในระดับบิต (Bit) ผลลัพธ์ที่ได้ในตัวแปร c ก็เกิดจากการ or ค่าของตัวแปร a และ b คราวนี้มาดูกลุ่มตัวแปร x,y,z ที่ใช้เครื่องหมายโอเปอเรเตอร์ | เช่นเดียวกับกลุ่ม a,b,c ที่อธิบายไปก่อนหน้านี้ เพียงแต่ว่าการใช้เครื่องหมายโอเปอเรเตอร์ | กับตัวแปร x,y,z ที่เป็นตัวแปรประเภท Sequence ชนิด List (สามารถจัดเก็บค่าข้อมูลได้มากกว่า 1 ค่า ภายในตัวแปร) จะอยู่ในรูปของการทำยูเนียนเซตทางคณิตศาสตร์ (Union Set) ด้วยการนำกลุ่มค่าข้อมูลในตัวแปร x มาทำการยูเนียนค่ากับตัวแปร y หรือ x u y โดยเก็บผลลัพธ์ที่ได้ลงในตัวแปร z จากทั้ง 2 กรณีของกลุ่มตัวแปร a,b,c และกลุ่ม x,y,z ที่ใช้เครื่องหมายโอเปอเรเตอร์เดียวกัน แต่ลักษณะการทำงานแตกต่างกัน เพื่อความง่ายต่อการศึกษาการใช้เครื่องหมายโอเปอเรเตอร์ของภาษา Python จะขอแบ่งออกเป็นหัวข้อย่อย ดังนี้

1. การใช้โอเปอเรเตอร์กับตัวแปรชนิดตัวเลข (Numeric And Iterable Operator)
2. การใช้โอเปอเรเตอร์กับตัวเลขจำนวนเต็มในระดับบิต (Integer Bitwise Operator)
3. เซตของข้อมูลกับการใช้โอเปอเรเตอร์ (Set Operator)

การใช้โอเปอเรเตอร์กับตัวแปรชนิดตัวเลข (Numeric And Iterable Operator)

การคำนวณทางคณิตศาสตร์นับเป็นพื้นฐานประกอบการเขียนคำสั่งร่วมกับตัวแปรหรือค่าตัวเลขโดยตรง ส่วนใหญ่จะใช้โอเปอเรเตอร์ด้านการคำนวณ เช่น บวก (+), ลบ(-),คูณ(*),หาร(/),ยกกำลัง(**) ฯลฯ (กรณีต้องการทำสูตรสมการที่ซับซ้อนมากขึ้น เช่น การทำ Log, ตรีโกณมิติ ซึ่งไม่มีโอเปอเรเตอร์รองรับการคำนวณแบบนี้โดยตรง แต่จะใช้ผ่านฟังก์ชันภายในที่เกี่ยวกับการคำนวณมาตรฐานทั่วไป เพียงแต่เมื่อใดก็ตามที่จะใช้ฟังก์ชันภายในเหล่านี้ก็อย่าลืมอ้างอิงถึงโมดูลที่ชื่อว่า Math หรือการ Import Math ก่อนเรียกใช้งานฟังก์ชันเสมอ) ต่อไปนี้เป็นตัวอย่างเกี่ยวกับโอเปอเรเตอร์ด้านการคำนวณดังตัวอย่างต่อไปนี้

<u>ตัวอย่างที่ 1</u>

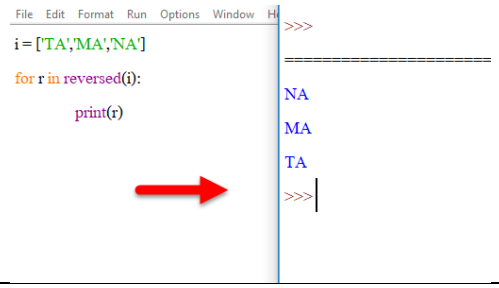
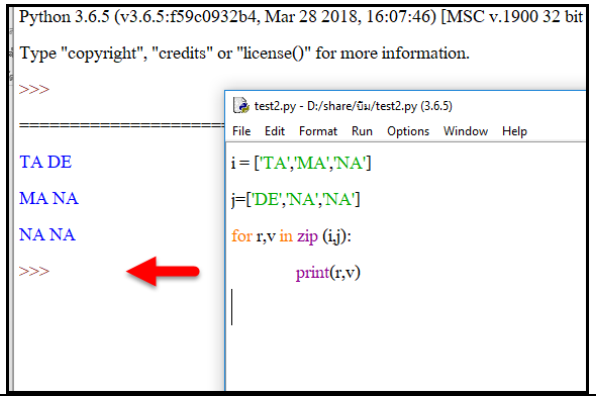


<u>ตัวอย่างที่ 1</u> จะเห็นการใช้เครื่องหมายโอเปอเรเตอร์ที่มีรูปแบบการเขียนคำสั่งที่แตกต่างกัน โดยที่มีกระบวนการคำนวณเหมือนกันตามตารางด้านล่าง แสดงรูปแบบวิธีใช้เครื่องหมายโอเปอเรเตอร์กับการคำนวณ

Numeric Operator	Iterable Operator
A = A + 1	A += 1
A = A * 1	A *= 1
A = A / 1	A / = 1
A = A - 1	A -= 1
A = A ** 2	A **= 2
A =A %3	A %= 3

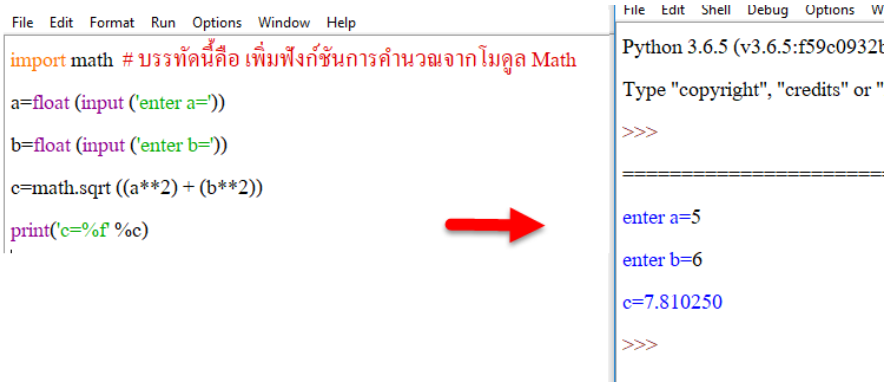
ตารางแสดงรูปแบบการใช้งานเครื่องหมายโอเปอเรเตอร์สำหรับการคำนวณ

รูปแบบ	คำอธิบาย
s + t	ถ้าหากตัวแปร s และ t มีค่าข้อมูลที่เป็นชนิดข้อความหรือ Sequence จะเป็นการเชื่อมต่อระหว่างกลุ่มข้อมูลเข้าด้วยกัน
s * n	ถ้าค่าข้อมูลในตัวแปรเป็นชนิดข้อความหรือ Sequence จะเป็นการเชื่อมต่อด้วยค่าข้อมูลของตัวแปรซ้ำๆ กันตามจำนวนของตัวแปร n
x in i	นำค่าในตัวแปร I สืบค้นภายในค่าข้อมูลตัวแปร x ซึ่งผลลัพธ์จะให้ค่าเจอ(True) หรือไม่เจอ (Flase)
all(i)	กลุ่มคำข้อมูลทุกๆ คำในตัวแปร i จำเป็นต้องมีตรรกะเป็นจริง (True) ผลลัพธ์ของฟังก์ชันจะได้ True เป็นคำตอบ (เหมือนคำสั่ง if ใช้คู่กับ and) ตัวอย่าง i=10 j=3 all([i>2,j>2]) ผลลัพธ์ True
any(i)	กลุ่มคำสั่งข้อมูลทุกๆ ค่าในตัวแปร i ค่าใดค่าหนึ่งมีตรรกะเป็นจริง (True) ผลลัพธ์ของฟังก์ชันจะได้ True เป็นคำตอบ (เหมือนคำสั่ง if ใช้คู่กับ or) ตัวอย่าง i=30 j=30 any([i>2,j>10]) ผลลัพธ์ True

รูปแบบ	คำอธิบาย
enumerate(i,start)	แสดงลำดับ Index คู่กับกลุ่มค่าข้อมูลในตัวแปร i ตัวอย่าง i = [‘L’,’O’,’M’] for r,v in enumerate(i): print(r,v) ผลลัพธ์ 0 L 1 O 2 M
len(x)	อ่านจำนวนตัวอักษรในตัวแปร x หรือจำนวนกลุ่มค่าตัวข้อมูลในตัวแปร x ที่เป็นชนิด Sequence
max(i,key)	บอกค่า key ที่มากที่สุดในตัวแปร i ที่เป็นชนิด Sequence
min(i,key)	บอกค่า key ที่น้อยที่สุดในตัวแปร i ที่เป็นชนิด Sequence
rang(start,stop,step)	ทำการนับจำนวนตัวเลขโดยเริ่มจาก Start และสิ้นสุดที่ Stop ซึ่งผลลัพธ์ที่ได้จะแสดงค่าออกมา
reversed(i)	อ่านค่ากลุ่มข้อมูลในตัวแปร i ที่เป็นชนิด Sequence จากท้ายสุดมาหน้าสุด ตัวอย่าง 
Sorted(i, key, reverse)	เรียงค่าข้อมูลในตัวแปร i
sum(i,start)	หาค่าผลรวมในตัวแปร i ที่เป็นชนิด Sequence
****(a,…,b)	กำหนดลำดับระหว่างค่าข้อมูลที่เกิดจากกลุ่มตัวแปร a ถึง b 

ตารางแสดงรูปแบบการใช้โอเปอเรเตอร์และฟังก์ชันสำหรับข้อมูลชนิด Sequence

ตัวอย่างที่ 2 ทดลองพิมพ์คำสั่งการใช้สมการพีทาโกรัส รูป004 สำหรับการคำนวณหาความยาวของด้านตรงข้ามมุมฉาก โดยการป้อนค่า a และ b ผ่านคีย์บอร์ด และทำการแสดงผลลัพธ์



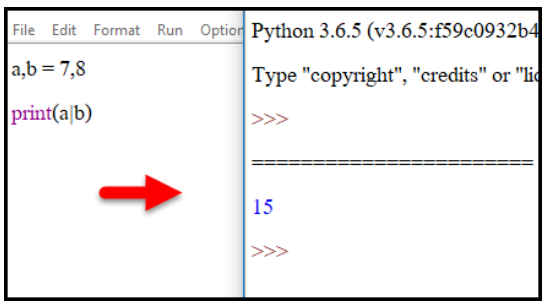
การใช้โอเปอเรเตอร์กับตัวเลขจำนวนเต็มในระดับบิต (Integer Bitwise Operator)

เครื่องหมายโอเปอเรเตอร์ในระดับบิตมักนิยมนำมาใช้กับเลขฐานสอง (Binary) สำหรับการคำนวณทางด้านพีชคณิตบูลีน (Boolean) ซึ่งประกอบด้วยการ and, or, xor และ Not (Invert) เป็นต้น เมื่อนำมาเขียนเป็นคำสั่งร่วมกับภาษา Python จะใช้เป็นเครื่องหมายโอเปอเรเตอร์กับตัวแปรที่เป็นชนิดตัวเลขจำนวนเต็มดังตาราง

รูปแบบ	คำอธิบาย
i | j	จะได้ผลลัพธ์ที่เกิดจากตรรกะการ or จากค่าในตัวแปร i และ j ที่เป็นชนิด Int
i ^ j	จะได้ผลลัพธ์ที่เกิดจากตรรกะการ xor จากค่าในตัวแปร i และ j ที่เป็นชนิด Int
i & j	จะได้ผลลัพธ์ที่เกิดจากตรรกะการ and จากค่าในตัวแปร i และ j ที่เป็นชนิด Int
i << j	ผลลัพธ์จากการเลื่อนค่า bit ไปทางซ้าย 1 ค่า คล้ายกับการคำนวณจากสมการ i*(2**j))
i >> j	ผลลัพธ์จากการเลื่อนค่า bit ไปทางขวา 1 ค่า คล้ายกับการคำนวณจากสมการ i*(2**j))
~ i	จะได้ผลลัพธ์ที่เกิดจากตรรกะการ Not

ตารางแสดงการใช้โอเปอเรเตอร์กับเลขจำนวนเต็มในระดับบิต

เป็นการกำหนดรูปแบบการคำนวณทางพีชคณิตบูลีนของ Python เพื่อให้เกิดความเข้าใจต่อการใช้งานโอเปอเรเตอร์ทางพีชคณิตบูลีน ดังตัวอย่างต่อไปนี้



A = 5	0	1	0	1	แปลง 5 เป็นเลขฐาน 2
B = 3	0	0	1	1	แปลง 3 เป็นเลขฐาน 2
A|B	0	1	1	1	ทำการ or ค่าในแต่ละบิตระหว่าง A กับ B

```
File  Edit  Format  Run  Options
a,b = 5,3
print (a|b)
print (a,b)
a |=b
print (a)
a,b = 5,3
print (a&b)
print (a,b)
a &= b
print (a)
print (a,b)
a=2
print (a>>1)
```

```
Python 3.6.5 (v3.6.5:f59c0
Type "copyright", "credits"
>>>
==========
7
5 3
7
1
5 3
1
1 3
1
>>>
```

<u>อธิบายคำสั่ง</u>

a,b = 5,3

print (a|b)

print (a,b)

a |=b <--- ทำการ or ค่าบิตของ a กับ b แล้วเก็บผลลัพธ์ที่ได้ไว้ใน a สังเกตการณ์ใช้เครื่องหมายโอเปอเรเตอร์ or คู่กับเครื่องหมายเท่ากับ (=)

print (a)

a,b = 5,3

print (a&b) <--- ทำการ and ค่าบิตระหว่างตัวแปร a และ b

print (a,b)

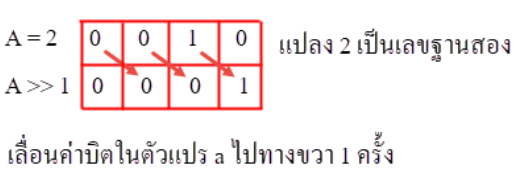
a &= b <--- ทำการ and ค่าบิตของตัวแปร a,b แล้วเก็บผลลัพธ์ที่ได้ไว้ในตัวแปร a

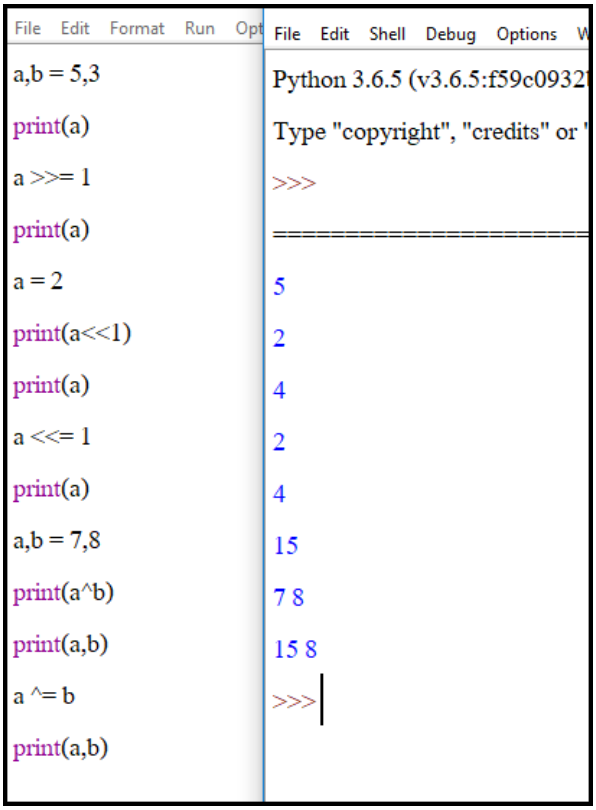
print (a)

print (a,b)

a = 2 <--- กำหนดค่า a=2 หรือ 0010 ฐานสอง

print (a>>1) <--- ใช้โอเปอเรเตอร์ >> สำหรับเลื่อนค่าบิตไปทางขวาตามจำนวนบิตที่ระบุ เช่น a>>1 แสดงถึงการเลื่อนค่าบิตไปทางขวา 1 ครั้ง





<u>อธิบายคำสั่ง</u>

a,b = 5,3

print(a)

a >>= 1 <--- เมื่อเพิ่มเครื่องหมายเท่ากับต่อท้ายเครื่องหมายโอเปอเรเตอร์ >> เพื่อต้องการเก็บผลลัพธ์ที่เกิดจากเลื่อนค่าบิตไว้ในตัวแปร a ดังนั้นค่าในตัวแปรจะเปลี่ยนไปจากค่าเดิม

print(a)

a = 2

print(a<<1) <--- ทำการเลื่อนค่าบิตในตัวแปร a ไปทางซ้ายมือ 1 ครั้ง โดยไม่เก็บผลลัพธ์ที่เกิดจากการเลื่อนค่าบิต

print(a)

a <<= 1 <--- เมื่อเลื่อนค่าบิตไปทางซ้าย 1 ครั้ง และให้เก็บผลลัพธ์ที่เกิดจากการเลื่อนไว้ในตัวแปร a

print(a)

a,b = 7,8

print(a^b) <--- ทำการ xor ค่าบิตในตัวแปร a ด้วยค่าในตัวแปร b

print(a,b)

a ^= b <--- หลังทำการ xor ในตัวแปร a กับ b จะเก็บผลลัพธ์ที่ได้ลงในตัวแปร a

print(a,b) <--- ค่าตัวแปร a จะเปลี่ยนไปจากเดิม

เซตของข้อมูลกับการใช้โอเปอเรเตอร์ (Set Operator)

ใน Python มีตัวแปรประเภท Sequence ที่ประกอบด้วย Tuple,List และ Dictionary ที่มีความสามารถในการจัดเก็บค่าข้อมูลชนิดต่างๆ เป็นกลุ่มเป็นก้อนลงในตัวแปรประเภท Sequence ได้

รูปแบบ	คำอธิบาย
s.add(x)	เพิ่มค่าข้อมูลในตัวแปร s ด้วยค่าของตัวแปร x
s.clear()	ลบกลุ่มค่าข้อมูลทั้งหมดของตัวแปร s
s.copy()	ใช้สำหรับการ copy กลุ่มค่าข้อมูลทั้งหมดของตัวแปร s ให้กับตัวแปรอื่นๆ
s.difference(t) s-t	เป็นฟังก์ชันที่แสดงกลุ่มค่าข้อมูลที่มีความแตกต่างของตัวแปร s และ t โดยแสดงผลลัพธ์ส่วนที่แตกต่างของตัวแปร s เท่านั้น
s. difference_update(t) s-=t	เป็นฟังก์ชันที่แสดงกลุ่มค่าข้อมูลที่มีความแตกต่างของตัวแปร s และ t โดยแสดงผลลัพธ์ส่วนที่แตกต่างของตัวแปร s เท่านั้นและจัดเก็บค่าผลลัพธ์ลงในตัวแปร s
s.discard(x)	ลบค่าข้อมูลในตัวแปร s ด้วยค่าข้อมูลของตัวแปร x
s.intersection(t) s&t	เป็นฟังก์ชันที่แสดงกลุ่มค่าข้อมูลที่เหมือนกันของตัวแปร s และ t โดยแสดงผลลัพธ์ส่วนที่เหมือนกันของตัวแปร s เท่านั้น
s. intersection_update(t) s&=t	เป็นฟังก์ชันที่แสดงกลุ่มค่าข้อมูลที่เหมือนกันของตัวแปร s และ t โดยแสดงผลลัพธ์ส่วนที่เหมือนกันของตัวแปร s เท่านั้นและจัดเก็บค่าผลลัพธ์ลงในตัวแปร s
s.sclisjoint(t)	แสดงผลลัพธ์เป็น True เมื่อตรวจค่ากลุ่มข้อมูลในตัวแปร s และ t ไม่มีกลุ่มข้อมูลใช้ร่วมกัน
s.issubset(t) s<=t	แสดงผลลัพธ์เป็น True เมื่อตรวจค่ากลุ่มข้อมูลในตัวแปร s และ t ที่เป็นลักษณะ subset
s.issuperset(t) s>=t	แสดงผลลัพธ์เป็น True เมื่อตรวจค่ากลุ่มข้อมูลในตัวแปร s และ t ที่เป็นลักษณะ superset
s.poop()	ลบค่าข้อมูลที่ตำแหน่งท้ายสุดของตัวแปร s
s.remove(x)	ลบค่าข้อมูลในตัวแปร s ด้วยค่าข้อมูลของตัวแปร x ถ้าไม่เจอจะเกิดข้อผิดพลาดในกลุ่มของ Key Error
s.symmetric_difference(t) s^t	เป็นฟังก์ชันที่แสดงกลุ่มค่าข้อมูลที่มีความแตกต่างของตัวแปร s และ t
s.symmetric_difference_ update (t) s^=t	เป็นฟังก์ชันที่แสดงกลุ่มค่าข้อมูลที่มีความแตกต่างของตัวแปร s และ t โดยจัดเก็บค่าผลลัพธ์ลงในตัวแปร s

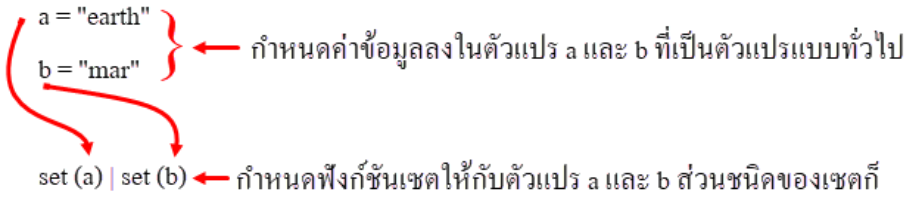
รูปแบบ	คำอธิบาย
s.union(t) s|t	เป็นฟังก์ชันที่แสดงกลุ่มค่าข้อมูลที่มีทั้งหมดของตัวแปร s และ t
s.update(t) s|=t	เป็นฟังก์ชันที่แสดงกลุ่มค่าข้อมูลที่มีทั้งหมดของตัวแปร s และ t โดยจัดเก็บค่าผลลัพธ์ลงในตัวแปร s

ตารางแสดงรูปแบบการใช้โอเปอเรเตอร์และฟังก์ชันสำหรับข้อมูลชนิด Set

แต่ถ้าต้องการนำกลุ่มข้อมูลเหล่านั้นมาทำงานแบบเซต ภาษา Python ก็มีฟังก์ชันเซต (Set Function) สำหรับทำงานกับค่าในตัวแปรประเภท Sequence โดยที่ฟังก์ชันเซตแบ่งออกเป็น 4 ชนิดย่อย ได้แก่

1. ฟังก์ชันยูเนียน (Union) ใช้เครื่องหมายโอเปอเรเตอร์คือ |
2. ฟังก์ชันอินเตอร์เซ็กชัน (Intersection) ใช้เครื่องหมายโอเปอเรเตอร์คือ &
3. ฟังก์ชัน Difference ใช้เครื่องหมายโอเปอเรเตอร์ คือ -
4. ฟังก์ชัน Symmetric Difference ใช้เครื่องหมายโอเปอเรเตอร์คือ ^

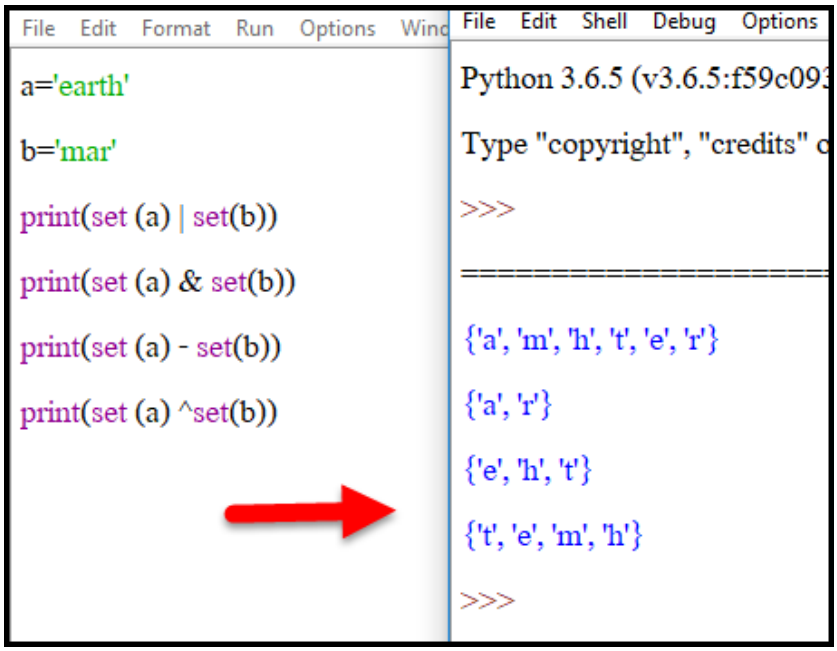
รูปแบบการเขียนคำสั่งสำหรับการใช้ฟังก์ชันเซตกับตัวแปร Python จะมีลักษณะตามรูปตัวอย่างด้านล่างจะแสดงผลลัพธ์ที่เกิดจากการใช้ฟังก์ชันเซตแต่ละชนิดร่วมกับตัวแปร



รูปแสดงรูปแบบและการเขียนฟังก์ชันเซตร่วมกับตัวแปร

ชนิดของฟังก์ชันเซต	รูปแบบการเขียน	ผลลัพธ์
Uninon	set('earth') | set('mar')	'a','e', 'h', 'm', 'r', 't'
intersection	set('earth') & set('mar')	'a', 'r'
Difference	set('earth') – set('mar')	'h','e', 't'
Symmetric Difference	set('earth') ^ set('mar')	'h', 'm', 'r', 't'

ตารางแสดงการใช้งานฟังก์ชันเซตแต่ละชนิด



<u>อธิบายคำสั่ง</u>

a='earth'

b='mar'

print(set (a) | set(b)) <--- ฟังก์ชันเซตชนิดยูเนียน

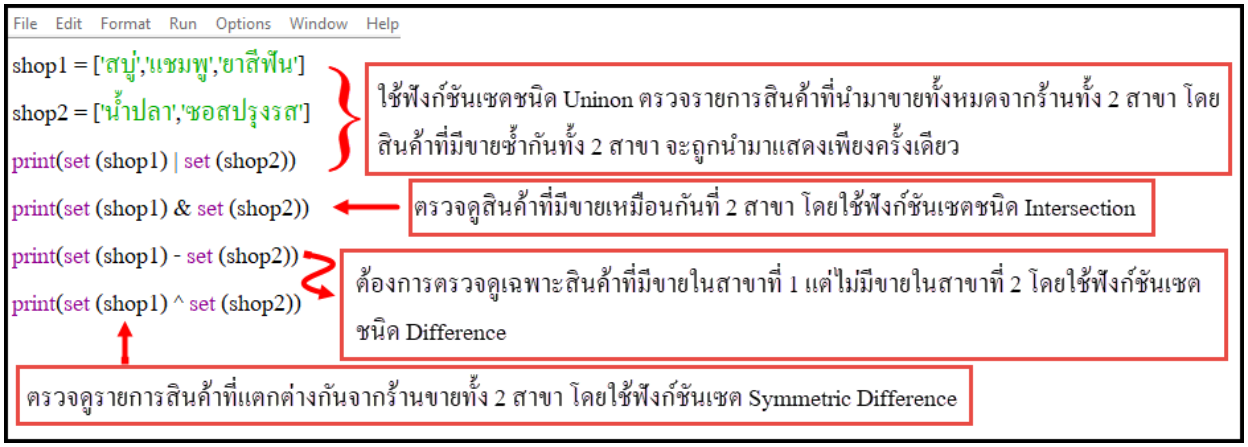
print(set (a) & set(b)) <--- ฟังก์ชันอินเตอร์เซ็กชัน

print(set (a) - set(b)) <--- ฟังก์ชันชนิด Difference

print(set (a) ^set(b)) <--- ฟังก์ชันเซตชนิด Symmetric Difference

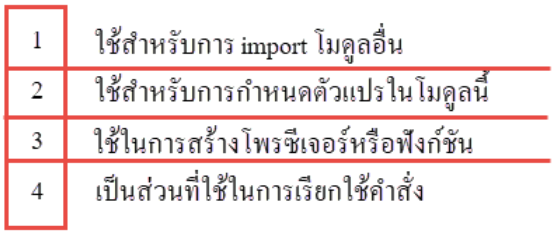
<u>ตัวอย่างที่ 2</u> การใช้ฟังก์ชันเซตกับตัวแปรประเภท Sequence

สมมุติว่ามีร้านขายของ 2 สาขา ที่มีสินค้าหน้าร้านบางรายการเหมือนกัน และบางรายการก็ต่างกัน คราวนี้ลองมาใช้ฟังก์ชันเซตชนิดต่างๆ ช่วยงานเกี่ยวกับสินค้า ดังนี้



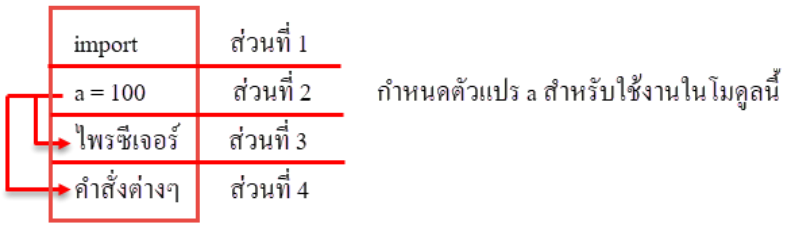
➢ การกำหนดขอบเขตการใช้งานตัวแปรในโมดูล

การเขียนโปรแกรมสำหรับพัฒนางานต่างๆ ด้วยภา Python มักนิยมเขียนกลุ่มคำสั่งต่างๆ ลงในโมดูล (ทำการเปิดโมดูลใหม่ โดยการเลือกเมนู File แล้วตามด้วยการเลือกเมนู New บนจอภาพจะปรากฏหน้าจอใหม่สำหรับเขียนโปรแกรมขึ้นมา สำหรับโครงสร้างโมดูลของ Python ประกอบด้วยอะไรบ้าง ได้อธิบายไว้ในบทที่ 2) สิ่งหนึ่งที่ต้องให้ความสนใจกับการใช้งานตัวแปรในโมดูลที่เขียนคือ การกำหนดขอบเขตการใช้งานตัวแปรภายในโมดูล เนื่องจากโครงสร้างในโมดูลจะแบ่งออกเป็น 4 ส่วน ตามรูปด้านล่าง



รูปแสดงโครงสร้างของโมดูลสำหรับเขียนคำสั่งต่างๆ

ส่วนที่2,3 และ 4 ของโครงสร้างโมดูล มักเป็นส่วนที่เกี่ยวกับการใช้งานตัวแปรของโปรแกรม โดยส่วนใหญ่เรามักประกาศตัวแปรในส่วนที่ 2 ซึ่งจะมีขอบเขตที่ตัวแปรจะถูกเรียกนำไปใช้งานได้ในส่วนที่ 3 และ 4 ของโมดูล



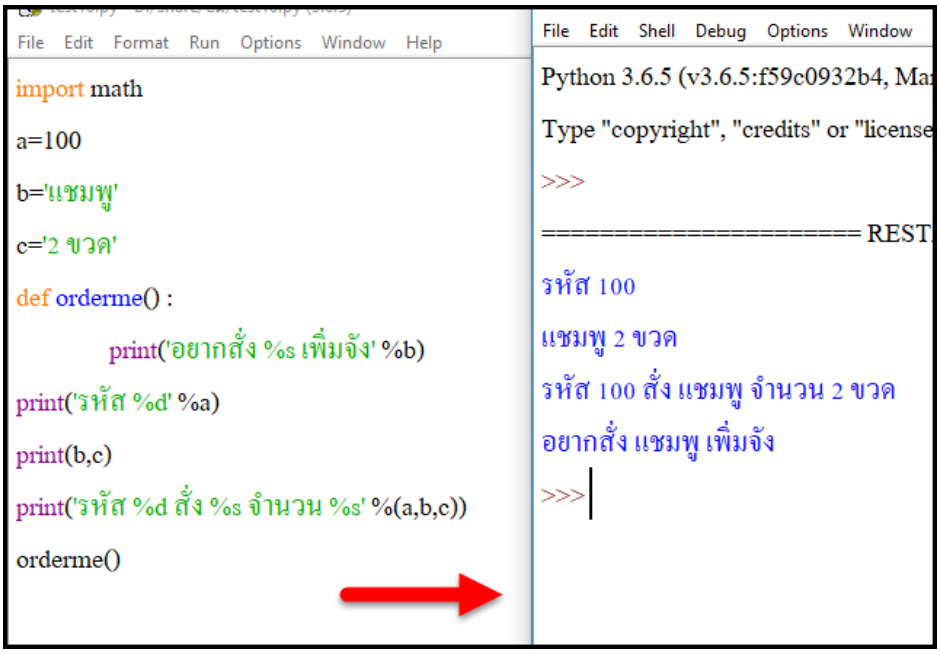
รูปแสดงขอบเขตการกำหนดและเรียกใช้งานของตัวแปรภายในโมดูลของ Python

เขียนคำสั่งตามตัวอย่าง (เปิดโมดูลใหม่ Python Shell โดยการเลือกเมนู File เลือกเมนูย่อยที่ชื่อ New)

บรรทัดที่ 1	Import math	ในส่วนที่ 1 อ้างอิงโมดูล math จากภายนอกมาใช้ในโมดูลนี้
บรรทัดที่ 2 บรรทัดที่ 3 บรรทัดที่ 4	a = 100 b = ‘แชมพู’ c = ‘2 ขวด’	สำหรับส่วนที่ 2 ได้กำหนดตัวแปร a,b,c เพื่อใช้งานในโมดูลนี้
บรรทัดที่ 5 บรรทัดที่ 6	def orderme(): print(‘อยากสั่ง %s เพิ่มจัง’ %b)	เป็นส่วนที่ 3 สำหรับฟังก์ชันที่ชื่อ orderme ซึ่งมีการเรียกใช้ตัวแปรจากส่วนที่ 2 คือ b มาใช้ร่วมกับคำสั่งอื่นๆ ภายในฟังก์ชัน

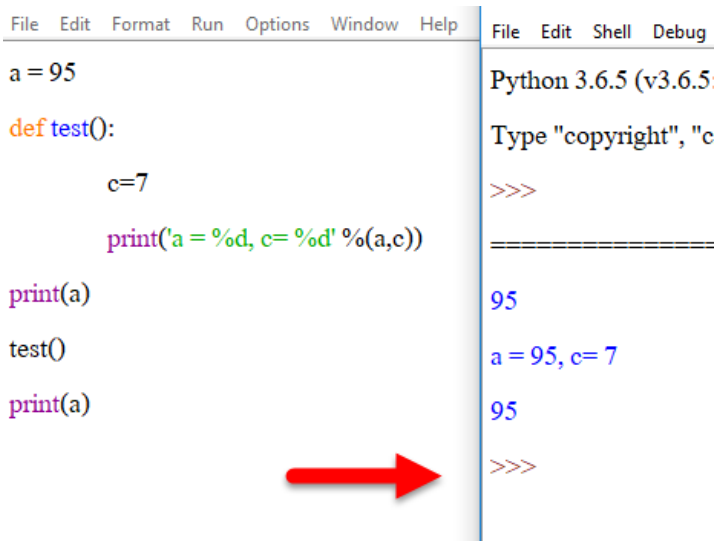
บรรทัดที่ 7 บรรทัดที่ 8 บรรทัดที่ 9 บรรทัดที่ 10	print(‘รหัส %d’ %a) print(b,c) print(‘รหัส %d สั่ง %s จำนวน %s’ %(a,b,c)) orderme()	ส่วนที่ 4 ซึ่งเป็นจุดเริ่มต้นการทำงานของโปรแกรม และเรียกใช้งานตัวแปร a,b,c จากส่วนที่ 2

ตัวอย่างการเขียนคำสั่ง



จากตัวอย่างจะเห็นได้ถึงวิธีกำหนดตัวแปรในโมดูล และวิธีเรียกใช้ตัวแปรที่กำหนดขึ้นในส่วนต่างๆ ของโมดูล ขอบเขตการใช้งานตัวแปร ในบางกรณีจำเป็นต้องระบุคีย์บอร์ด (Keyword) ที่ชื่อว่า Global เพิ่มเติมลงไปในโพรซีเจอร์หรือฟังก์ชัน อาจจะมีการกำหนดตัวแปรแบบ Local เพื่อใช้งานเฉพาะภายในโพรซีเจอร์หรือฟังก์ชันเท่านั้น สังเกตได้จากตัวอย่างด้านล่าง

<u>ตัวอย่างที่ 1</u> แสดงวิธีกำหนดและใช้งานตัวแปรแบบ Local ในฟังก์ชัน test



อธิบายคำสั่ง

a = 95 <--- กำหนดตัวแปร a ในส่วนที่ 2 ของโมดูล

def test(): <--- สร้างฟังก์ชัน test ในส่วนที่ 3 ของโมดูล

```
    c=7
    print('a = %d, c= %d' %(a,c))
```

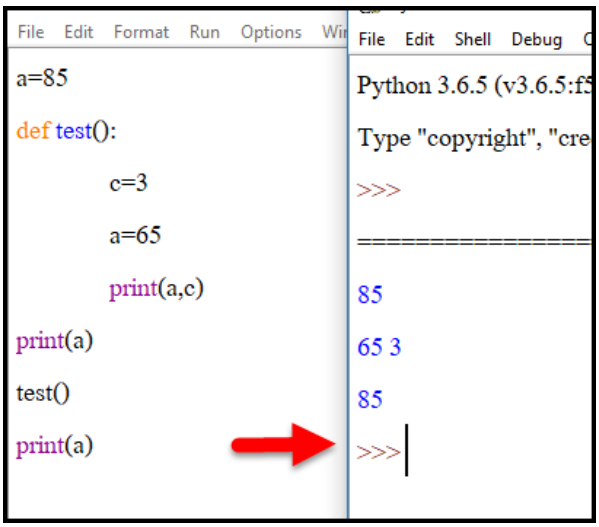
print(a) <--- เป็นจุดเริ่มต้นเขียนคำสั่งต่างๆ ในส่วนที่ 4 ของโมดูล โดยบรรทัดนี้จะพิมพ์ค่าในตัวแปร a ที่กำหนดในส่วนที่ 2

test() <--- เรียกฟังก์ชัน test มาใช้งาน

print(a) <--- พิมพ์ค่าตัวแปร a ที่กำหนดในส่วนที่ 2

จากตัวอย่างที่ 1 จะเห็นว่าในฟังก์ชัน test มีการกำหนดตัวแปรภายในที่ชื่อ c สำหรับใช้งานเฉพาะในฟังก์ชันเท่านั้น และในฟังก์ชันนี้ก็ยังใช้งานตัวแปร a ของโมดูลนี้ โดยการพิมพ์ค่าตัวแปร a ด้วยคำสั่ง print

ตัวอย่างที่ 2 แสดงการกำหนดตัวแปรแบบ Local ในฟังก์ชัน test ที่มีชื่อตัวแปร a ของโมดูล และ ตัวแปรชื่อ a ของฟังก์ชัน test เหมือนกัน เพื่อดูผลลัพธ์ที่เกิดขึ้น



อธิบายคำสั่ง

a=85 <--- กำหนดตัวแปร a ในส่วนที่ 2 ของโมดูล

def test(): <--- สร้างฟังก์ชัน test ในส่วนที่ 3 ของโมดูล

c=3 <---

a=65

กำหนดตัวแปรภายในที่ชื่อ a และ c ที่ใช้เฉพาะในฟังก์ชันนี้เท่านั้น ซึ่งตัวแปรภายใน a ที่กำหนดในฟังก์ชันนี้ จะไม่ใช่ตัวเดียวกันกับตัวแปร a ของ

print(a,c) <--- พิมพ์ค่าตัวแปร a และ c ซึ่งตัวแปรทั้ง 2 ตัวจะใช้งานเฉพาะฟังก์ชันนี้เท่านั้น

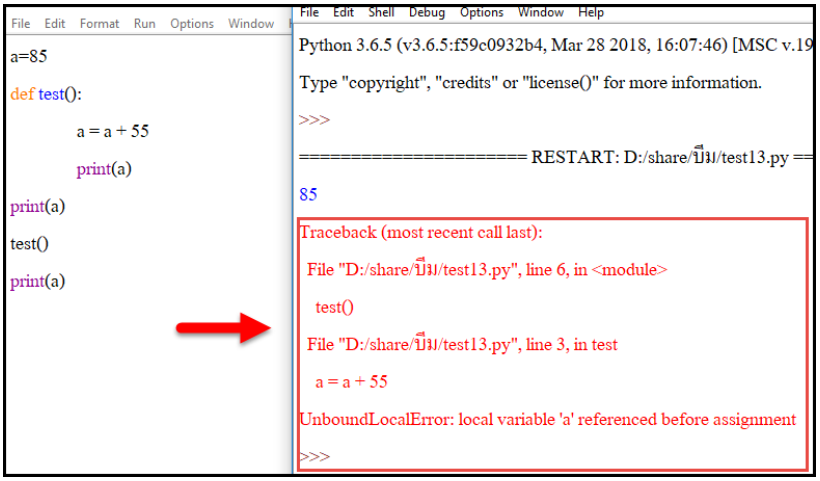
print(a) <--- จุดเริ่มต้นของคำสั่งในโปรแกรมนี้ จะทำการพิมพ์ค่าในตัวแปร a ของโมดูล

test() <--- เรียกฟังก์ชันในส่วนที่ 3 ของโมดูลนี้

print(a)

จากตัวอย่างที่ 2 การกำหนดตัวแปรที่ชื่อ a จะมี 2 ตัวแปร โดยตัวแปร a ตัวแรกจะถูกกำหนดในส่วนที่ 2 ของโมดูล ซึ่งมีขอบเขตการใช้งานตัวแปรตัวนี้ได้ทุกที่ในโมดูลนี้ ตัวแปร a ตัวที่สองถูกกำหนดภายในฟังก์ชัน test ได้มีการตั้งชื่อตัวแปรซ้ำกับตัวแปร a ตัวแรก ภายในฟังก์ชัน test เมื่อมีการเรียกใช้ตัวแปรที่ชื่อ a ฟังก์ชัน test จะเรียกใช้ตัวแปรภายใน a ตัวที่สองเสมอ

<u>ตัวอย่างที่ 3</u> แสดงการใช้ตัวแปร a ของโมดูล ร่วมกับฟังก์ชัน test โดยใช้คีย์เวิร์ด Global



รูปแสดงข้อความผิดพลาดที่เกิดจากการนำตัวแปร a ของโมดูลมาใช้คำนวณภายในฟังก์ชัน test

<u>อธิบายคำสั่ง</u>

a=85 <--- กำหนดตัวแปร a ในส่วนที่ 2 ของโมดูลนี้

def test(): <--- สร้างฟังก์ชัน test ในส่วนที่ 3 ของโมดูลนี้

a = a + 55 <--- เรียกใช้ตัวแปร a ของโมดูลมาทำการคำนวณในฟังก์ชัน test

print(a)

print(a) <--- จุดเริ่มต้นของคำสั่งในโมดูลนี้ ซึ่งอยู่ในส่วนที่ 4

test()

print(a)

ผลลัพธ์จากการทำงานของโปรแกรมในตัวอย่างที่ 3 จะเกิด Error ดังที่ปรากฏในรูปด้านบน ในฟังก์ชัน test บรรทัดที่เป็นการ

บทที่ 4
การตรวจสอบเงื่อนไข

ลองนึกภาพการใช้งานโปรแกรมสมัครสมาชิกร้านกาแฟ ที่ในช่องอายุของสมาชิกมีความจำเป็นต้องกรอกจำนวนตัวเลขเท่านั้น เพื่อนำไปจัดเก็บลงบันทึกเข้าสู่ระบบฐานข้อมูล ดูแล้วก็น่าจะเป็นการใช้งานที่ราบรื่นตามขั้นตอนการใช้งานโปรแกรมทั่วไป แต่เผอิญผู้ใช้งานมันถือไปนิด หนึ่งป้อนเป็นค่า a7 ลงในช่องอายุ ลูกค้าก็รีบๆ เพลงให้ผู้ใช้งานรีบทำงานให้เสร็จผู้ใช้งานจึงไม่ได้ตรวจสอบความถูกต้องของข้อมูลที่กรอกไว้บนจอภาพก่อนทำการบันทึกลงในฐานข้อมูล คิดดูว่าอะไรจะเกิดขึ้นกับข้อมูลในช่องอายุ ที่ต้องเป็นตัวเลข ถ้าโปรแกรมเมอร์เขียนโปรแกรมไว้ดีก็จะมีการตรวจสอบข้อมูลตามช่องกรอกข้อมูลว่าถูกต้องหรือเปล่า แล้วแจ้งเตือนผู้ใช้งานให้แก้ไขให้ถูกต้องจะเห็นว่าพื้นฐาน 1 ในการเขียนโปรแกรมไม่ว่าจะเขียนด้วยภาษาใดหรือแม้แต่ภาษา python ก็จะต้องมีคำสั่งตรวจสอบเงื่อนไขเพื่อนำมาใช้แก้ไขปัญหาตามที่ยกตัวอย่างโปรแกรมร้านกาแฟกับการป้อนอายุเป็นต้นซึ่งส่วนใหญ่คำสั่งตรวจเงื่อนไขวง ได้ (python conditional branching) มักจะใช้เป็นคำสั่ง If (If statement) โดยจะมีวิธีการเขียนตามรูปแบบดังนี้

If เงื่อนไข :

{ ชุดคำสั่ง
ทำเมื่อตรงกับเงื่อนไข

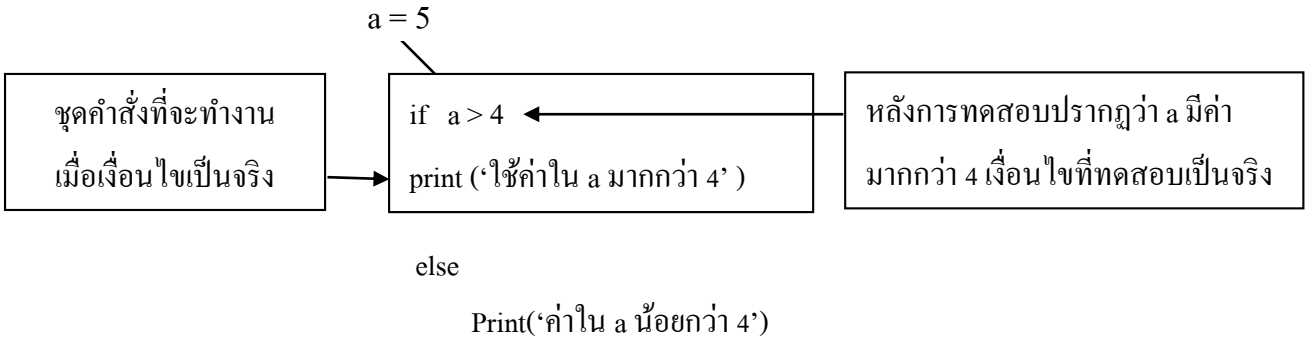
else :

{ ชุดคำสั่ง
ทำกรณีที่ไม่ตรงกับเงื่อนไข

คำว่า “เงื่อนไข” ที่เห็นในรูปแบบของคำสั่ง if (ซึ่งส่วนใหญ่ Python จะเรียกว่า Comparson) ข้างต้น พูดง่ายๆ ก็คือการเปรียบเทียบสิ่งที่ต้องการตรวจสอบนั่นเอง เช่น ค่าในช่องอายุ == ตัวเลขอย่างเดียว (จากตัวอย่างที่เล่าเรื่องในตอนต้นของบทนี้ เมื่อต้องการสร้างเงื่อนไขสำหรับการตรวจสอบช่องอายุ) หรือต้องการตรวจค่าในตัวแปร a มีค่าตัวเลขเกิน 1500 หรือเปล่า วิธีเขียนเงื่อนไขเพื่อตรวจสอบคือ a > 1500 ลงในคำสั่ง if หลังจากการทดสอบตามเงื่อนไขที่ระบุ คำสั่ง if จะบอกผลลัพธ์ที่ได้เป็น 2 ลักษณะ คือ

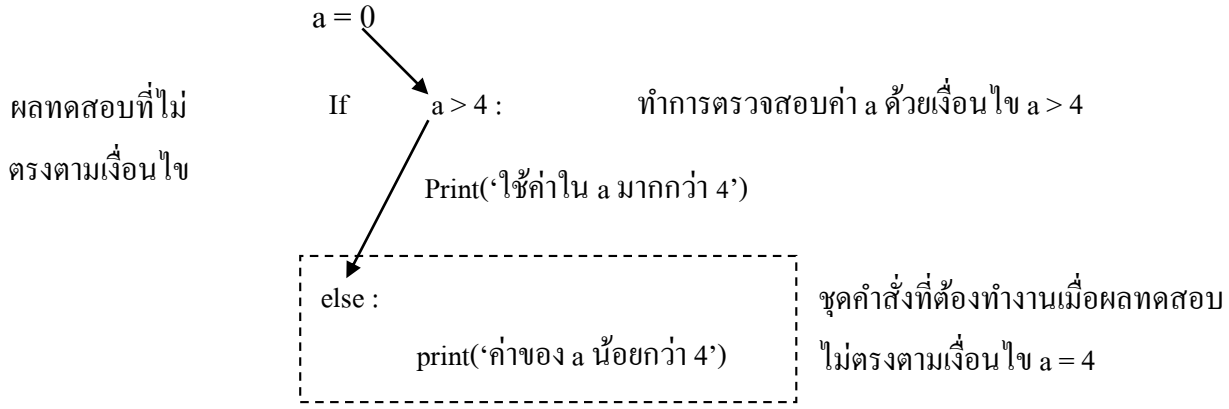
1. ผลลัพธ์เป็นไปตามเงื่อนไขหรือค่าที่เป็นจริง (True)
2. ผลลัพธ์ไม่ตรงกับเงื่อนไขที่ตรวจสอบหรือได้ผลเป็นเท็จ (False)

ผลลัพธ์เป็นไปตามเงื่อนไขหรือค่าที่เป็นจริง (True)



แสดงการทดสอบเงื่อนไขของคำสั่ง if และชุดคำสั่งที่ต้องทำงานเมื่อเงื่อนไขเป็นจริง

ผลลัพธ์ไม่ตรงกับเงื่อนไขที่ตรวจสอบหรือได้ผลเป็นเท็จ (False)



แสดงเส้นทางการทดสอบเงื่อนไขที่เป็นเท็จ และเริ่มกระบวนการทำงานในชุดคำสั่งหลังคำสั่ง else ซึ่งเป็นพื้นที่ของกลุ่มคำสั่งเมื่อการทดสอบเป็นเท็จ

เมื่อมองดูรูปแบบการสร้างเงื่อนไขเช่น a > 4 จะมีการใช้เครื่องหมายทางด้าน Operator สำหรับการเปรียบเทียบ (comparison มาประกอบร่วมกับค่าข้อมูลที่ต้องการเปรียบเทียบในที่นี้คือตัวแปรและตัวเลข 4 สำหรับการตรวจสอบเงื่อนไขว่าค่าในตัวแปร a มากกว่าค่าตัวเลข 4 หรือไม่) ลักษณะของการเลือกใช้เครื่องหมาย Operator สำหรับการเปรียบเทียบเงื่อนไขแบ่งออกเป็น 4 รูปแบบดังนี้

1. เครื่องหมายสำหรับเปรียบเทียบหาค่าข้อมูลที่เท่ากัน (Equality comparison operation) เช่น a== 10, B== c ฯลฯ จะใช้เครื่องหมายเท่ากับ (แต่เขียน 2 ครั้งติดกันตามตัวอย่าง)ในการสร้างเงื่อนไข

2. เครื่องหมายในการเปรียบเทียบค่าข้อมูลที่ไม่เท่ากัน (Not Equality comparison operation) จะใช้ ! = รวมกับค่าข้อมูลที่ต้องการเปรียบเทียบเช่น a!=10 (ค่าข้อมูลในตัวแปร a ไม่เท่ากับค่าตัวเลข 10 ใช่หรือไม่ เป็นต้น

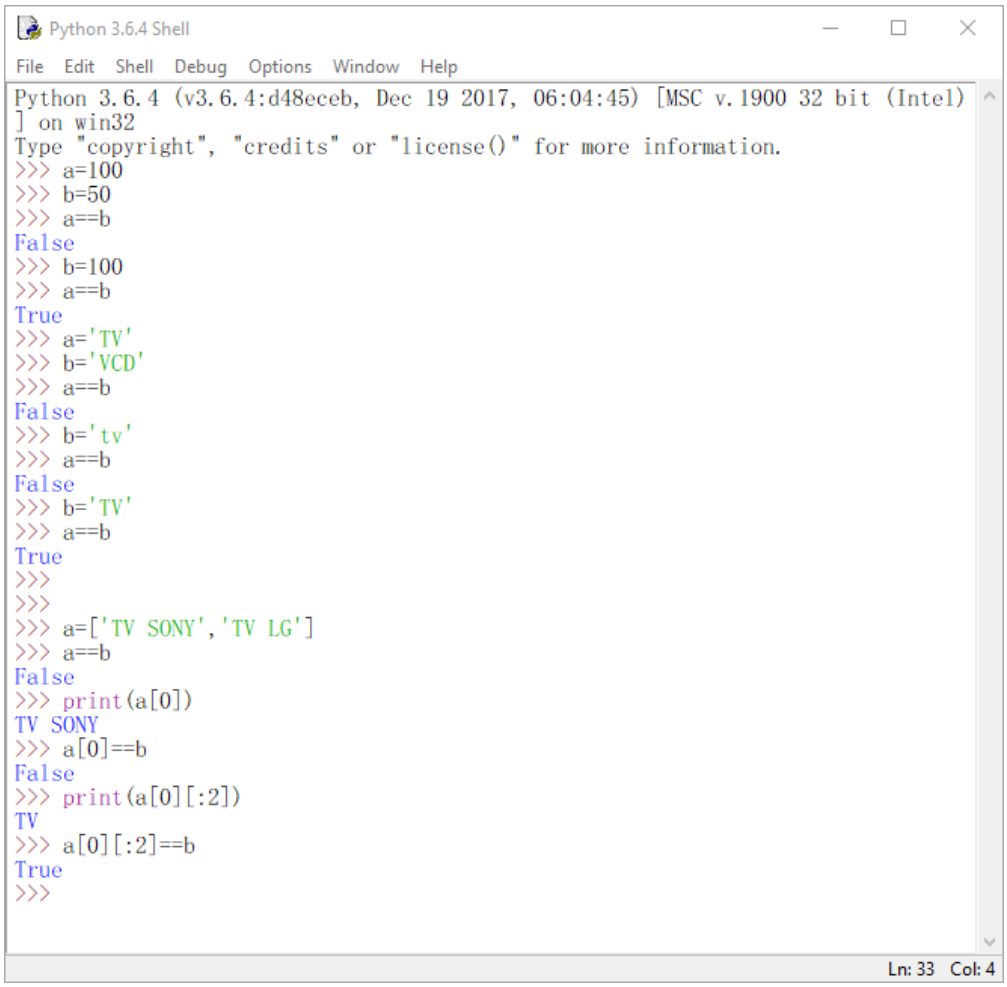
3. เครื่องหมายเปรียบเทียบค่าข้อมูลมากกว่าหรือน้อยกว่า (Greater Than and Less Than Comparison Operation) ใช้เครื่องหมาย > (มากกว่า), < (น้อยกว่า), >= (มากกว่าหรือเท่ากับ), <= (น้อยกว่าหรือเท่ากับ)

4. Not กับการทำหน้าที่ในการเปรียบเทียบ (Not Comparition operation) โอเปอเรเตอร์นี้เหมาะกับการนำมาใช้งานกับค่าข้อมูลที่เป็น True หรือ False ด้วยการทำงานของ โอเปอเรเตอร์ not จะเปลี่ยนค่าข้อมูลไปในทางตรงกันข้ามเช่นถ้าค่าในตัวแปร A มีค่าเป็น True เมื่อนำ โอเปอเรเตอร์ not มาใช้กับตัวแปร a หรือ not a ผลลัพธ์จะได้ค่า false ด้วยเหตุนี้ถ้าข้อมูลที่จะนำมาใช้กับ โอเปอเรเตอร์ not ต้องมี2สถานะเสมอคือ True และfalse

จะเห็นได้ว่าลักษณะการใช้โอเปอเรเตอร์การเปรียบเทียบของภาษา Python จะมีการใช้งานเหมือนภาษาโปรแกรมอื่นๆตามมาตรฐานทั่วไปจึงง่ายต่อการเรียนรู้รูปแบบการเขียนและการทำงาน

ตารางสรุปการใช้เครื่องหมายโอเปอเรเตอร์เปรียบเทียบ

โอเปอเรเตอร์	ความมาย	ตัวอย่าง	ผลลัพธ์
==	เท่ากับ	A == 10	จะได้ค่า True ก็ต่อเมื่อตัวแปร A มีค่าเป็น 10 เท่านั้น
>	มากกว่า	A > 3	ถ้าค่าข้อมูลในตัวแปร A มีค่าตั้งแต่ 4 ขึ้นไปจะได้ผลลัพธ์เป็น True
>=	มากกว่าหรือเท่ากับ	A >=3	ผลลัพธ์จะเป็น True ตั้งแต่ค่าในตัวแปรมีค่าเป็น 3 ขึ้นไป
<	น้อยกว่า	A < 2	จะเป็น True เมื่อค่าข้อมูลในตัวแปร A มีค่าเป็น 1 หรือน้อยกว่า 1 เสมอ
<=	น้อยกว่าหรือเท่ากับ	A <= 2	เมื่อไรค่าข้อมูลในตัวแปร A มีค่าเป็น 2
!=	ไม่เท่ากับ	A! = 10	จะได้ผลลัพธ์เป็น True ตราบใดที่ค่าข้อมูลใน A ไม่เท่ากับ 10
not	ตรวจค่าตรงกันข้ามระหว่าง True กับ False	Not A	



ทดลองเขียนคำสั่งและสังเกตผลลัพธ์ที่ได้จากการทำงานตามตัวอย่างต่อไปนี้ใน Python Shell

ตัวอย่างที่ 1 โอเปอเรเตอร์เปรียบเทียบสำหรับทดสอบค่าที่เท่ากัน

```
>>> a=100
>>> b=50
>>> a==b    ◄── ตรวจค่าในตัวแปร a และ b เท่ากันหรือไม่
>>> b=100   ◄── แก้ไขค่าใน b เป็น 100
>>> a==b    ◄── ตรวจค่าใดตัวแปรทั้ง 2 ตัวอีกครั้ง
>>> a='TV'
>>> b='VCD'
>>> a==b
>>> b='tv'
```

```
>>> a==b
>>> b='TV'
>>> a==b
>>> a=['TV SONY','TV LG']
>>> a==b
>>> print(a[0])
>>> a[0]==b
>>> print(a[0][:2])
>>> a[0][:2]==b
```

น่าแปลกที่ TV ตัวอักษรตัวใหญ่ในตัวแปร a ไม่เท่ากับ tv ของตัวแปร b เนื่องจากเป็นจากเป็นการเปรียบเทียบด้วยรหัส ASCll หรือ Unicode ของตัวอักษร

ตัดคำว่า TV จาก TV SONY ซึ่งได้จากตัวแปร a [0]

นำคำว่า TV ใน a[0] [:2] มาเปรียบเทียบกับตัวแปร b

ตัวอย่างที่ 2 โอเปอเรเตอร์เปรียบเทียบมากกว่าและน้อยกว่า

```
>>>a = input
```

(‘ป้อนค่าตัวเลขอะไรก็ได้’)

```
>>>a = int(a)
```

แปลงค่าในตัวแปรที่รับค่าจากฟังก์ชัน Input (ซึ่งจะเป็นข้อมูลชนิด String เสมอ) ให้เป็นตัวเลขชนิดจำนวนเต็มหรือ Integer ผ่านฟังก์ชันแปลงชนิดที่ชื่อ Int

```
>>>a > 50
```

ตรวจค่าในตัวแปร a มีค่ามากกว่า 50 หรือเปล่า

```
>>>a = ‘d’
>>>b = ‘D’
>>>a > b
```

ตรวจค่าในตัวแปร a และ b ด้วยโอเปอเรเตอร์ > (มากกว่า) กระบวนการเปรียบเทียบจะนำค่า ‘d’ ตัวอักษรตัวเล็ก และ ‘D’ ตัวอักษรตัวใหญ่ไปถอดเป็นรหัส ASCll หรือ Unicode แล้วนำรหัสมาเปรียบเทียบกัน

```
>>> a = ‘ปลาทู’
>>> b = ‘ปลาทู’
>>>a > = b
```

ภาษาไทยไม่ค่อยมีปัญหาเรื่องตัวอักษรเล็กหรือใหญ่

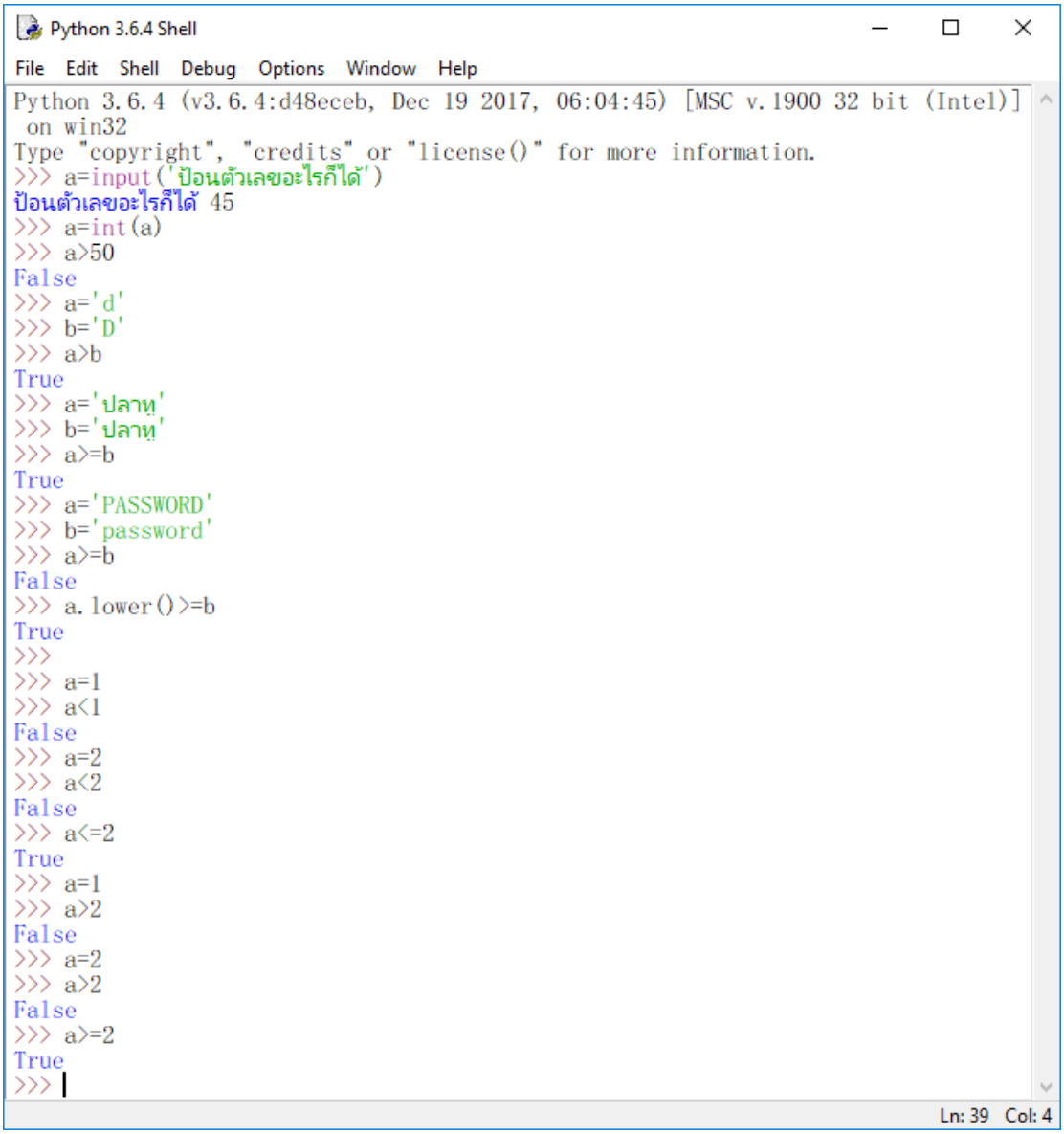
```
>>>a = ‘PASSWORD’
>>>b = ‘password’
>>>a >= b
>>>a.lower() >= b
```

ใช้ฟังก์ชัน Lower แปลงค่าตัวอักษรในตัวแปร a ให้เป็นตัวอักษรตัวเล็กทั้งหมด แล้วค่อยนำไปเปรียบเทียบกับตัวแปร b

```
>>>a = 1
>>>a < 2
```

ค่าในการเปรียบเทียบคือ 1 < 2

```
>>>a = 2
```



>>> a < 2 ⟵ ค่าที่ใช้ทดสอบคือ 2 < 2

>>> a <= 2 ⟵ ลองใหม่กับการใช้ <= ด้วยค่า 2<=2

>>> a = 1

>>> a > 2 ⟵ ทดสอบค่ามากกว่าด้วย 1 > 2

>>> a = 2

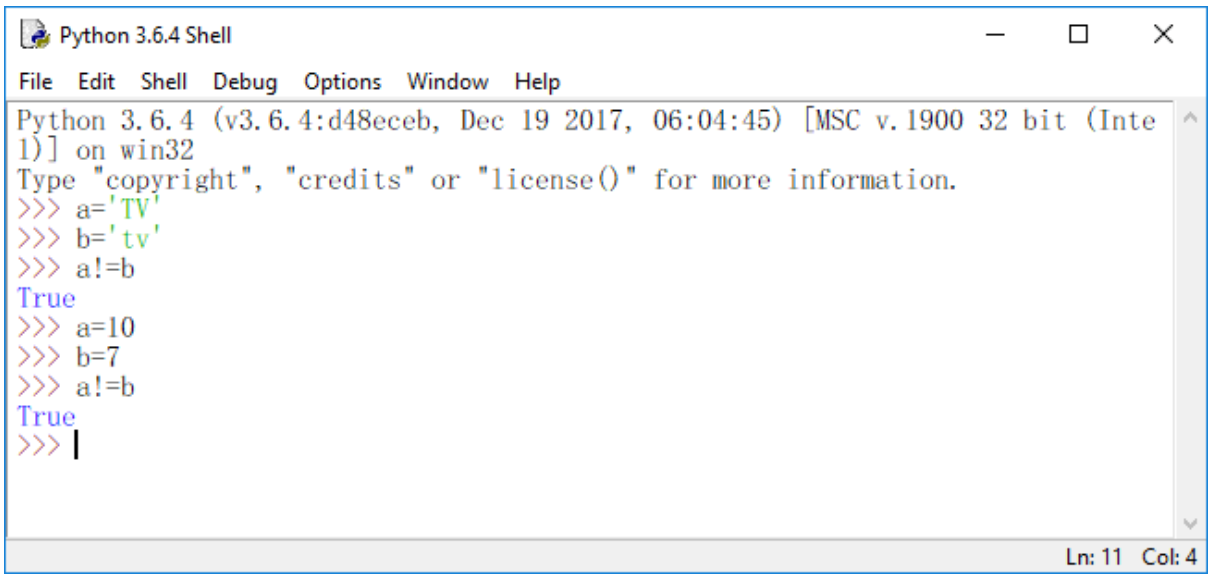
>>> a > 2 ⟵ ดูการทดสอบเมื่อ a มีค่าเป็น 2 ด้วย 2 > 2

>>> a >= 2 ⟵ เมื่อใช้ >= กับค่า 2 >= 2 อะไรจะเกิดขึ้น

ตัวอย่างที่ 3 โอเปอเรเตอร์เปรียบเทียบกับการทดสอบค่าที่ไม่เท่ากัน

```
>>> a='TV'
>>> b='tv'
>>> a!=b
```
⟵ TV ตัวอักษรพิมพ์ใหญ่ไม่เท่ากับ tv ตัวอักษรตัวพิมพ์เล็กใช่หรือไม่

```
>>> a=10
>>> b=7
>>> a!=b
```
⟵ 10 ไม่เท่ากับ 7 ใช่ไหม



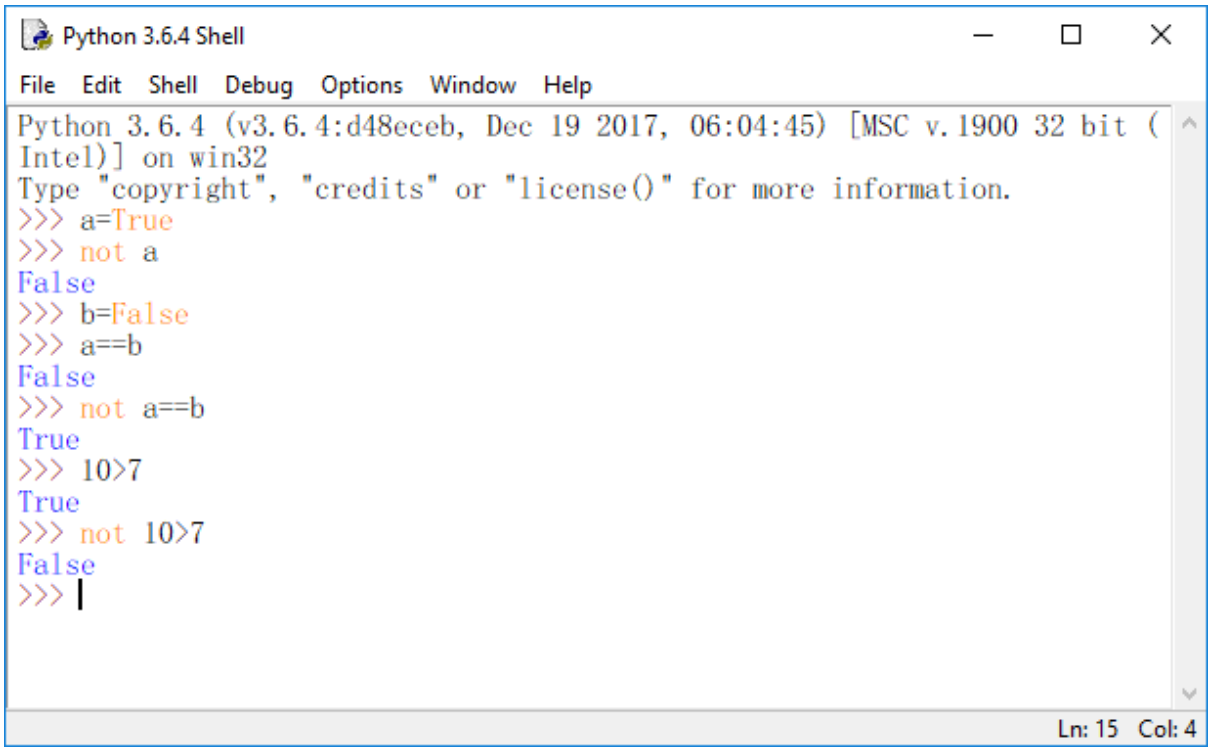
ตัวอย่างที่ 4 โอเปอเรเตอร์เปรียบเทียบกับ Not

```
>>> a=True
>>> not a
>>> b=False
>>> a==b
```
⟵ เปรียบเทียบค่าใน a และ b

```
>>> not a==b
```
⟵ ใช้ not a สำหรับกลับค่าใน a จาก True ให้เป็น False แล้วค่อยนำไปเปรียบเทียบกับ b

```
>>> 10>7
```
⟵ ผลลัพธ์จากการเปรียบเทียบจะได้ True เพราะเงื่อนไข 10>7 เป็นจริง

```
>>> not 10>7
```
⟵ กระบวนการจะเริ่มจากการเปรียบเทียบ 10 > 7 เพื่อให้ได้ผลลัพธ์จากการเปรียบเทียบก่อน ซึ่งในที่นี้จะได้ True แล้วทำการเพิ่ม Not สำหรับกลับค่าผลลัพธ์ที่ได้ให้เป็นค่าตรงกันข้าในที่นี้คือจาก True ไปเป็น False



หลังจากทำความเข้าใจการกำหนดเงื่อนไขด้วย Operator ด้านการเปรียบเทียบเมื่อต้องนำนั่นไข่มาประกอบการเขียนร่วมกับคำสั่ง If โดยผลลัพธ์ของคำสั่ง if มีมี 2 สถานะเมื่อตรวจสอบตรงตามเงื่อนไขจะได้สถานะเป็น True และได้สถานะเป็น false เมื่อการตรวจสอบไม่ตรงตามเงื่อนไขซึ่งจะมีการระบุการกระทำสำหรับแต่ละสถานะของผลลัพธ์ตามโครงสร้างของคำสั่ง If แต่งรูปแสดงเส้นทางการกระทำของผลลัพธ์ที่เกิดจากการตรวจสอบเงื่อนไข(ภายใต้โครงสร้างของคำสั่ง If จะมีวิธีกำหนดพื้นที่ย่อยโดยแบ่ง indent ของคำสั่งย่อย ซึ่งได้กล่าวไว้ในบทที่ 2 เกี่ยวกับการกำหนดกลุ่มคำสั่งย่อยภายใต้คำสั่งหลัก)

การเข้าถึงพื้นที่ของกลุ่มคำสั่งย่อยภายใต้คำสั่ง If ซึ่งจะแบ่งออกเป็น 2 พื้นที่คือพื้นที่ของกลุ่มคำสั่งย่อยเมื่อผลลัพธ์ของการตรวจสอบเงื่อนไขมีสถานะเป็นจริง (True) และพื้นที่ของกลุ่มคำสั่งย่อยที่จะทำงานได้ก็ต่อเมื่อผลการตรวจสอบเงื่อนไขไม่ตรงตามที่กำหนดของเงื่อนไขหรือได้สถานะเป็นเท็จ (False) คำสั่ง If

เมื่อต้องนำคำสั่ง If มาใช้ร่วมกับการเขียนโปรแกรม python จะมีรูปแบบที่นักพัฒนาโปรแกรมนิยมเขียนอยู่ 4 รูปแบบ ตามเหตุการณ์ของการตรวจสอบเงื่อนไขดังนี้ที่มันคิด

1. การใช้คำสั่ง If ตรวจสอบเงื่อนไขและทำงานกลุ่มคำสั่งย่อย เมื่อการตรวจสอบเงื่อนไขเป็นจริง (True) เท่านั้น

2. การใช้คำสั่ง If ร่วมกับ elif สำหรับตรวจสอบเงื่อนไขและทำกลุ่มคำสั่งย่อย เมื่อผลลัพธ์การตรวจสอบเป็นจริง (True) เท่านั้น

3. การใช้คำสั่ง If ที่มีลักษณะการตรวจสอบเงื่อนไขแบบสอนกัน (Nested)

4. การใช้คำสั่ง If ร่วมกับคำสั่ง else

เส้นทางการทำงานเมื่อการตรวจสอบเงื่อนไขเป็นจริง(True)

a = 10

if a = = 10:

พื้นที่ของกลุ่มคำสั่งย่อยจะทำงานเมื่อเงื่อนไขที่ตรวจสอบตรงตามเงื่อนไขหรือมีสถานะเป็น True

else :

พื้นที่ของกลุ่มคำสั่งย่อยจะทำงานก็ต่อเมื่อการตรวจสอบไม่ตรงตามเงื่อนไขที่กำหนด โดยมีสถานะเป็น False

print('พิมพ์ค่าของ a = %d' %a)

print('ค่าของ b= '%d' %b)

เส้นทางการทำงานของคำสั่ง If เมื่อการตรวจสอบเป็นเท็จ Fasle

a = 1

if a = = 10:

พื้นที่ของกลุ่มคำสั่งย่อยจะทำงานเมื่อเงื่อนไขที่ตรวจสอบตามเงื่อนไขหรือมีสถานะเป็น True

else :

พื้นที่ของกลุ่มคำสั่งย่อยจะทำงานก็ต่อเมื่อการตรวจสอบไม่ตรงตามเงื่อนไขที่กำหนดโดยมีสถานะเป็น false

print('พิมพ์ค่าของ a = %d' %a)

print('ค่าของ b= '%d' %b)

การใช้คำสั่ง if ตรวจสอบเงื่อนไข และทำงานกลุ่มคำสั่งย่อย เมื่อการตรวจสอบเงื่อนไขเป็นจริง (True) เท่านั้น

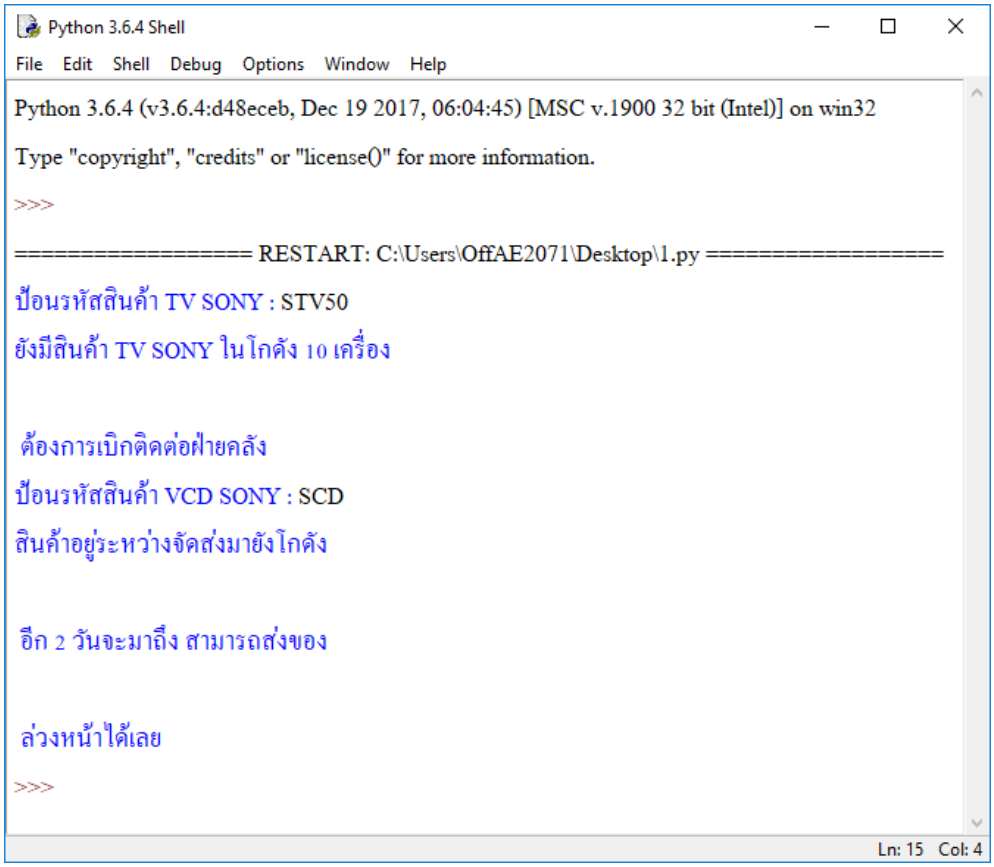
การใช้คำสั่ง If ตรวจสอบเงื่อนไข และทำงานกลุ่มคำสั่งย่อย เมื่อการตรวจสอบเงื่อนไขเป็นจริง (True) เท่านั้น

```
a = 10
if  a == 10:
```

พื้นที่กลุ่มคำสั่งย่อยเมื่อตรวจสอบเงื่อนไข ตรงตามที่กำหนดในคำสั่ง If หรือมีสถานะจริง (True)

ทดลองพิมพ์การใช้ if ตามรูปแบบที่ 1 ลงในโมดูลของ Python (ใน Python shell เลือกเมนู file คลิกเมนูย่อยที่ชื่อว่า New เพื่อเปิดโมดูลของ Python)

```
a = {'STV50' : 'TV SONY 50 นิ้ว', 'LG100' : 'TV LG', 'SCD' : 'VCD SONY', 'TDVD' : 'SAMSUNG DVD'}
b = input('ป้อนรหัสสินค้า TV SONY : ')
if a[b] == 'TV SONY 50 นิ้ว':
    print('ยังมีสินค้า TV SONY ในโกดัง 10 เครื่อง')
    print('\n ต้องการเบิกติดต่อฝ่ายคลัง')
b = input('ป้อนรหัสสินค้า VCD SONY : ')
if a[b] == 'VCD SONY':
    print('สินค้าอยู่ระหว่างจัดส่งมายังโกดัง')
    print('\n อีก 2 วันจะมาถึง สามารถส่งของ')
    print('\n ล่วงหน้าได้เลย')
```



การใช้คำสั่ง if ร่วมกับ elif สำหรับการตรวจสอบเงื่อนไขและการทำงานกลุ่มคำสั่งย่อย เมื่อผลลัพธ์การตรวจสอบเป็นจริง (True)

มีรูปแบบการเขียน ดังนี้

A = 10

If a == 10:

ทำคำสั่งย่อยในพื้นที่นี้เมื่อค่าในตัวแปร a มีค่าเท่ากับ 10

Elif a == 100:

ทำคำสั่งย่อยในพื้นที่นี้เมื่อค่าของตัวแปร a มีค่าเท่ากับ 100

Ekif a > 200:

ทำคำสั่งย่อยในพื้นที่นี้เมื่อค่าของตัวแปร a มีค่ามากกว่าหรือเท่ากับ 200

Elif a < = 0 :

ทำคำสั่งย่อยในพื้นที่นี้เมื่อค่าในตัวแป a น้อยกว่าหรือเท่ากับ 0

ถ้าหากมีโจทย์ที่ต้องการตรวจค่าในตัวแปร a โดยมีเงื่อนไขและการกระทำเมื่อการตรวจสอบเงื่อนไขเป็นจรองที่แตกต่างกัน ดังเช่นตัวอย่างข้างต้น การตรวจสอบค่าในตัวแปร a จะมีถึง 4 ครั้ง สั่งเกตจากการใช้คำสั่ง if และ elif ตรวจสอบเงื่อนไขและมีการทำงานของกล่มคำสั่งย่อยของใครของมัน เมื่อผลลัพธ์

การตรวจสอบมีสถานะเป็นจริง จำนวนของคำสั่ง if + elif ก็คือ จำนวนของความต้องการที่ตรวจสอบค่าในตัวแปร a นั่นเอง

เส้นทางลำดับการตรวจสอบค่าในตัวแปร a เมื่อผลการทดสอบเป็นเท็จทั้งหมด

```
a = 9
if a == 10:
   Print('a มีค่าเป็น 10 จริง')
elif a == 100:
   Print('a มีค่า')
elif a >= 200:
   print('มีค่ามากกว่า 200')
print('จบขั้นตอนการตรวจค่า a')
print('ทำคำสั่งอื่นๆ ต่อไป')
```

เส้นทางลำดับการตรวจสอบค่าในตัวแปร a เมื่อผลการทดสอบเป็นจริง

```
a = 100
if a == 10:
   print('a มีค่าเป็น 10')
elif a == 100:
   print('a มีค่า 100')
elif a >= 200:
   print('a มีค่ามากกว่า 200')
print('จบการทดสอบ')
print('ทำคำสั่งอื่นๆ ')
```

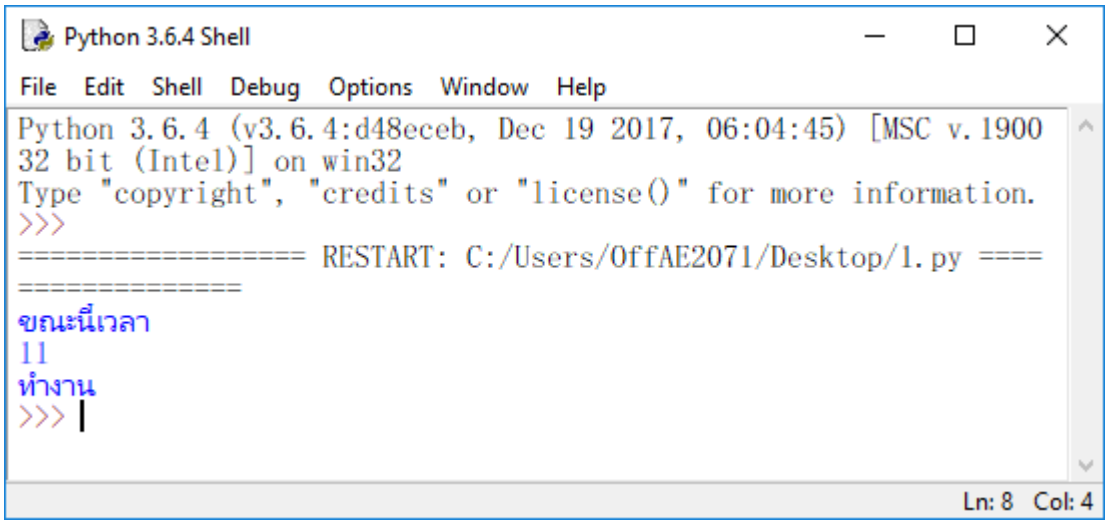
ด้านซ้ายแสดงลักษณะการทำงานของรูปแบบการเขียนคำสั่ง if + elif เมื่อผลลัพธ์การตรวจสอบเป็นเท็จทั้งหมด และทางด้านขวาแสดงการทำงานเมื่อการตรวจสอบค่าในตัวแปร a มีผลลัพธ์การตรวจสอบมีสถานะเป็นจริง และได้มีการกระทำของกลุ่มคำสั่งย่อยของคำสั่ง if หรือ elif นั้น

เพื่อให้ง่ายต่อการทำความเข้าใจจึงกำหนดลำดับการตรวจสอบในแต่ละคำสั่ง if และ elif ตามเงื่อนไขที่กำหนดไว้ โดยโครงสร้างการเขียนจะเริ่มต้นด้วยคำสั่งตรวจสอบแรกสุดคือ คำสั่ง if และเมื่อต้องเพิ่มคำสั่งตรวจสอบถัดไปให้ใช้คำสั่ง elif จึงจะถือว่าเป็นคำสั่งตรวจสอบตามรูปแบบในหัวข้อที่ 2 นี้

ทดลองการใช้คำสั่ง if + elif ลงในโมดูลของ Python (บนจอภาพ Python Shell เลือกเมนู file เลือกเมนูย่อย New เพื่อทำการเปิด Python โมดูล)

```
import time      ← ทำการอ้างอิงโมดูลมาตรฐานเกี่ยวกับฟังก์ชันเวลาหรือ time มาใช้ในโมดูลนี้
loadtime=time.localtime()  ← สร้างตัวแปรชื่อ loadtime เพื่ออ่านวัน เวลาจาระบบปฏิบัติการ
loadhour=loadtime.tm_hour  ← สร้างตัวแปรที่ชื่อ loadhour สำหรับอ่านชั่วโมงปัจจุบันมาใช้งาน
print('ขณะนี้เวลา')
print(loadhour)
if loadhour<7:  ← ตรวจสอบตารางเวลาสำหรับกิจกรรมประจำวัน
   print('ยังไม่ตื่น')
elif loadhour<9:
   print('อาบน้ำ ทานข้าว')
```

```
elif loadhour<16:
   print('ทำงาน')
elif loadhour<19:
   print('กลับบ้าน ทานข้าว')
elif loadhour<21:
   print('ดูทีวีรายการสุดโปรด')
```



การใช้คำสั่ง if ที่มีลักษณะการตรวจสอบเงื่อนไขแบบซ้อนกัน (Nested)

การเขียนคำสั่ง if แบบ Nested ฟังดูแล้วน่าจะเป็นการเขียนอะไรที่ต้องแปลกแตกต่างจากรูปแบบการเขียนคำสั่งในข้อที่ 1 และ 2 อย่างแน่นอน แต่ความเป็นจริงก็คือ การเขียนคำสั่งแบบ Nested เป็นการใช้คำสั่ง if ซ้อนภายในกลุ่มคำสั่งย่อยของคำสั่ง if แรกเท่านั้นเอง ประโยชน์ของการเขียนคำสั่ง if แบบ Nested ก็เพื่อที่จะมีการตรวจสอบเงื่อนไขย่อยลึกลงไป ยกตัวอย่าง หากคำในตัวแปร a มีข้อมูลเกี่ยวกับทีวีโซนี่ (TV Sony), ทีวีแอลจี(TV LG), ทีวีโตชิบา(TV Toshiba), ทีวีซัมซุง(TV Samsung), วีซีดีโซนี่(VCD Sony), วีซีดีแอลจี(VCD LG) เป็นต้น เราจะใช้คำสั่ง if สำหรับการตรวจสอบเงื่อนไข สำหรับการแบ่งประเภทของข้อมูลในตัวแปร a ออกเป็น 2 ประเภท หรือ 2 คำสั่ง if ประกอบด้วย ประเภททีวีและประเภทวีซีดี ตามรูปแบบการเขียนคำสั่ง ต่อไปนี้

```
a = ['tv sony','tv lg', 'tv toshiba', 'tv samsung', 'vcd sony', 'vcd lg']
if a[1][:2]== 'tv:':
```

พื้นที่กลุ่มคำสั่งย่อยจะทำงานก็ต่อเมื่อค่าในตัวแปร a[1][:2] เป็นประเภททีวีเท่านั้น

```
elif a[1][:3]=='vcd':
```

พื้นที่กลุ่มคำสั่งย่อยจะทำงานเมื่อผลการตรวจสอบค่าในตัวแปร a[1][:3] เป็นประเภทวีซีดีเท่านั้น

การใช้คำสั่ง if ตรวจสอบเงื่อนไขเพื่อแยกประเภทของข้อมูลในตัวแปร a นับเป็นการกรองข้อมูลชั้นที่ 1 ด้วยคำสั่ง if คราวนี้ลองมาเพิ่มการตรวจสอบเงื่อนไขย่อยด้วยคำสั่ง if เพิ่มลงในพื้นที่กลุ่มคำสั่งย่อยของแต่ละคำสั่ง if ในชั้นที่ 1 เพื่อตรวจสอบค่าข้อมูลเฉพาะ ยกตัวอย่าง ต้องการตรวจสอบหาค่าในตัวแปร a ที่ตรงกับคำว่า ‘tv lg’ โดยที่ค่าดังกล่าวจะตรงกับ a[1] (เนื่องด้วยตัวแปร a เป็นชนิด Sequence แบบ List ซึ่งต้องใช้หมายเลขอ้างอิงข้อมูลภายในของตัวแปร a อ่านรายละเอียดตัวแปรชนิดนี้ได้ในบทที่ 3) ซึ่งจะมีรูปแบบการเขียนคำสั่ง if เพิ่มเติมซ้อนลงในกลุ่มคำสั่งย่อยของคำสั่ง if ในชั้นที่ 1 ดังต่อไปนี้

```
a = ['tv sony','tv lg', 'tv toshiba', 'tv samsung', 'vcd sony', 'vcd lg']
if a[1][:2]== 'tv:':
    b = a[1]
    if b == 'tv sony':
        print('ค่าในคัวแปร b คือ tv sony')
    if b == 'tv lg':
        print('ค่าในตัวแปร b คือ tv lg')
    if b == 'tv toshiba':
        print('มีค่า tv toshiba ในตัวแปร b')
    if b == 'tv samsung':
        print('ในตัวแปร b มีค่า tv samsung ')
elif a[1][:3] == 'vcd':
    b = a[1]
    if b == 'vcd sony':
        print('ค่าในตัวแปร b มี vcd sony เก็บไว้ภายใน')
    if b == 'vcd lg':
        print('มีค่า vcd lg ในตัวแปร b')
```

ตรวจสอบเงื่อนไขด้วยคำสั่ง if ในชั้นที่ 1 เพื่อแยกประเภทค่าในตัวแปร a เป็นประเภททีวี

พื้นที่กลุ่มคำสั่งย่อยของการตรวจสอบเงื่อนไขประเภททีวี

พื้นที่กลุ่มคำสั่งย่อยของการตรวจสอบเงื่อนไขเมื่อตัวแปร a[1] เป็นประเภทวีซีดี

จากรูปแบบการเขียนคำสั่งที่เห็นในตัวอย่าง จะใช้คำสั่ง if ซ้อนกันเป็นชั้นๆ โดยการตรวจสอบเงื่อนไขจะมีลำดับจากคำสั่ง if ชั้นที่ 1 ซึ่งประกอบด้วย a[1][:2] == ‘vcd’ ภายใต้เงื่อนไขแรกคือ a[1][:2] == ‘tv’ หรือการตรวจข้อมูลสำหรับกลุ่มทีวี จะมีคำสั่ง if ย่อยเป็นชั้นที่ 2 อีก 4 คำสั่งสำหรับการตรวจสอบเฉพาะค่าข้อมูลของตัวแปรจริง เช่น ค่าใน b (ซึ่งรับค่าโอนถ่ายมาจากตัวแปร a[1]) มีค่าเท่ากับ ‘tv lg’ ใช่หรือไม่ ถ้าใช่ก็ทำคำสั่งย่อยของคำสั่ง if b == ‘tv lg’ คือทำคำสั่ง print(‘ค่าในตัวแปร b คือ tv lg’)

ตามรูปแบบการใช้คำสั่ง if ดังที่อธิบายไว้เป็นวิธีการใช้คำสั่ง if แบบ Nested สำหรับการตรวจสอบเงื่อนไขในเชิงลึก

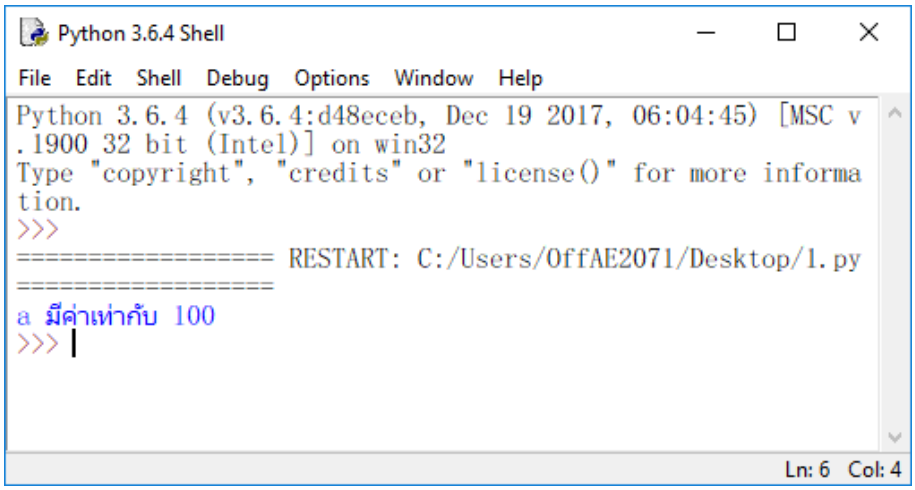
เส้นทางลำดับการตรวจสอบเงื่อนไขเชิงลึกด้วยคำสั่ง if แบบ Nested

```
a = ['tv sony', 'tv lg', 'tv lg', 'tv to shiba',...]
if a[1][:2] == 'tv':
    b = a[1]
    if b == 'tv sony':
        print('ค่าในตัวแปร b คือ tv sony')
    if b == 'tv lg':
        print('ค่าในตัวแปร b คือ tv lg')
    if b == 'tv toshiba':
        print('มีค่า toshiba ในตัวแปร b')
    if b == 'tv samsung':
        print('.....')
elif a[1][:3] == 'vcd':
    b = a[1]
    if b == 'vcd sony':
    print('........')
    if b == 'vcd lg':
    print('........')
print('จบการตรวจสอบ')
```

คำสั่ง if ชั้นที่ 1 ที่มีคำสั่ง if ย่อยอีก 4 คำสั่งภายในการตรวจสอบเงื่อนไขมีสถานะเป็นจริง

ทดลองพิมพ์การใช้คำสั่ง if แบบ Nested ลงในโมดูลของ Python

```
a = 100
if a < 40:
    if a == 25:
        print('a มีค่าเท่ากับ 25')
    if a == 30:
        print('a มีค่าเท่ากับ 30')
elif a > 80:
    if a == 50:
        print('a มีค่าเท่ากับ 50')
    if a == 100:
        print('a มีค่าเท่ากับ 100')
```



การใช้คำสั่ง If ร่วมกับคำสั่ง else

พื้นฐานการทำงานของคำสั่ง If มีหน้าที่ตรวจสอบเงื่อนไขตามที่กำหนด และต้องมีการตอบสนองต่อผลลัพธ์ของการตรวจสอบเงื่อนไขในพื้นที่กลุ่มคำสั่งย่อย เมื่องเงื่อนไขเป็นไปตามที่กำหนดหรือมีสถานะที่เป็นจริง (True) แต่ถ้าหากต้องการเพิ่มพื้นที่กลุ่มคำสั่งย่อยกรณีผลลัพธ์ของการตรวจสอบเงื่อนไขไม่ตรงตามที่กำหนดหรือมีสถานะเป็นเท็จ (false) รูปแบบการเขียนคำสั่งในกรณีนี้จำเป็นต้องเพิ่มคำสั่ง else ให้กับคำสั่ง If

```
a = 10
if  a == 10:
    พื้นที่ของกลุ่มคำสั่งย่อยจะตอบสนองเมื่อผลลัพธ์การตรวจสอบตรงที่กำหนดหรือมีสถานะเป็นจริง (True)
print('ทำต่อไปเรื่อยๆ')
```

แสดงรูปแบบการใช้คำสั่ง if สำหรับการตรวจสอบเงื่อนไขและตอบสนองกลุ่มคำสั่งย่อยเมื่อตรงตามที่กำหนดเพียงอย่างเดียว

```
a = 7
if  a == 10:
    พื้นที่ของกลุ่มคำสั่งย่อยเมื่อการตรวจสอบเงื่อนไขตรงตามเงื่อนไขหรือมีสถานะที่เป็นจริง (True)
else:
    พื้นที่ของกลุ่มคำสั่งย่อยกรณีผลลัพธ์ของการตรวจสอบไม่เป็นไปตามเงื่อนไขหรือมีสถานะที่เป็นเท็จ (False)
print('ทำต่อไปเรื่อยๆ')
```

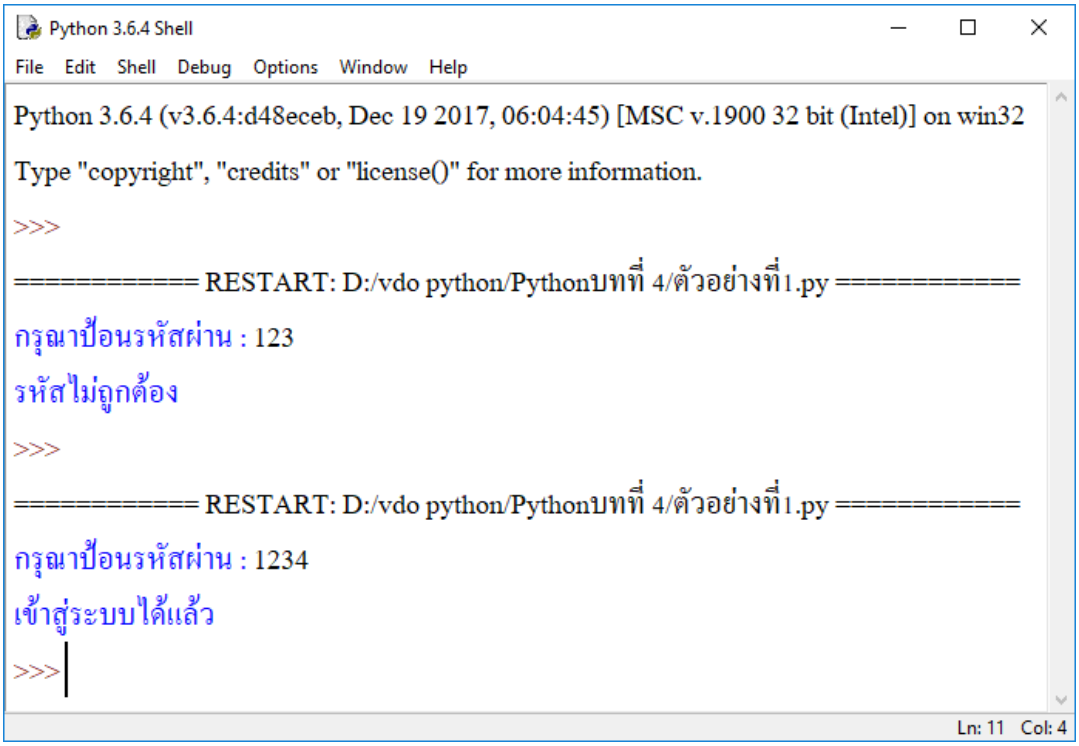
แสดงรูปแบบการใช้คำสั่ง if ร่วมกับคำสั่ง else สำหรับตอบสนองผลลัพธ์ที่เกิดจากการตรวจสอบเงื่อนไขและกระทำกับกลุ่มคำสั่งย่อยที่อยู่ในพื้นที่ตามสถานะที่เป็นจริงหรือเท็จ

แสดงรูปแบบการเขียนและลักษณะการทำงานของคำสั่ง if และ else

ทดลองพิมพ์คำสั่ง if และคำสั่ง else สำหรับการตรวจสอบเงื่อนไขลงในโมดูลของ Python

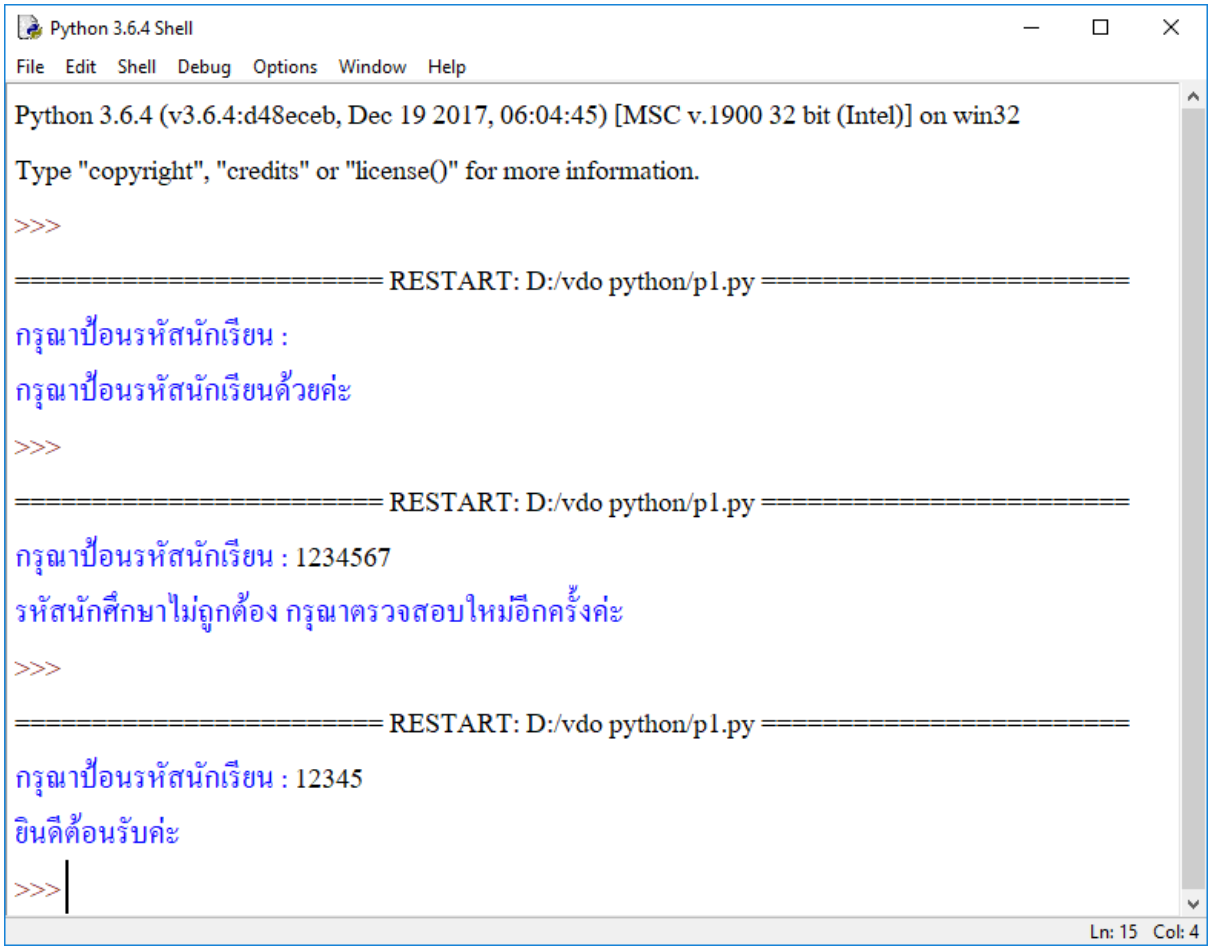
ตัวอย่างที่ 1

```
a=input('กรุณาป้อนรหัสผ่าน : ')
if a=='1234':
   print('เข้าสู่ระบบได้แล้ว')
else:
   print('รหัสไม่ถูกต้อง')
```



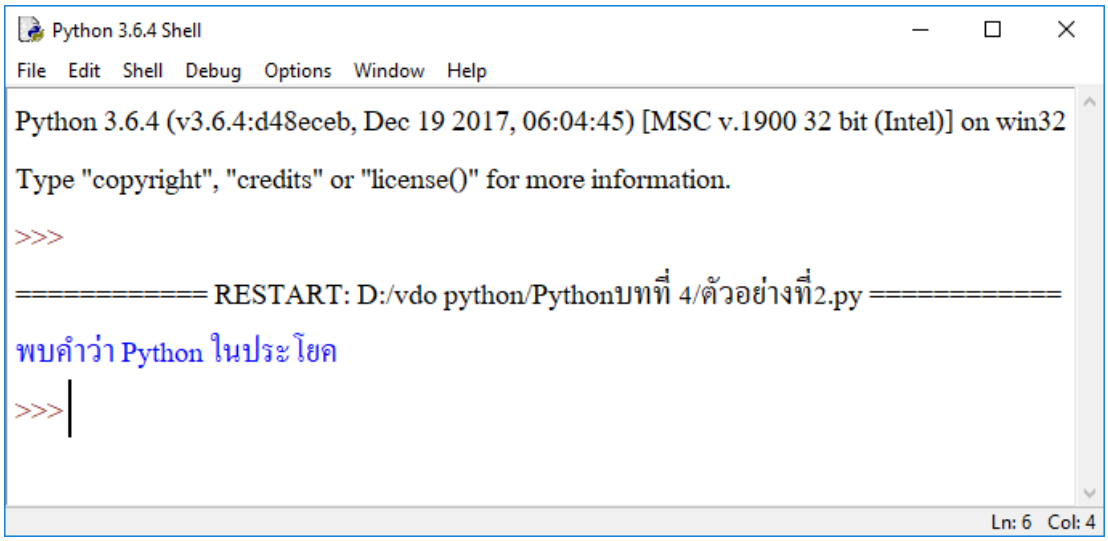
ตัวอย่างที่ 2

```
a=input('กรุณาป้อนรหัสนักเรียน : ')
if len(a) <= 0:
   print('กรุณาป้อนรหัสนักเรียนด้วยค่ะ')
elif len(a) > 5:
   print('รหัสนักศึกษาไม่ถูกต้อง กรุณาตรวจสอบใหม่อีกครั้งค่ะ')
else:
   print('ยินดีต้อนรับค่ะ')
```



ตัวอย่างที่ 3

```
a=' Python is a computer language '
if a.find('Python') != -1:
   print('พบคำว่า Python ในประโยค')
else:
   print('ไม่พบคำว่า Python')
```



การกำหนดเงื่อนไขมากกว่าหนึ่งเงื่อนไขให้กับคำสั่ง if

การกำหนดเงื่อนไขมากกว่า 1 เงื่อนไขให้กับคำสั่ง If

ถ้าหากต้องการสร้างเงื่อนไขตรวจค่าข้อมูลที่อยู่ในตัวแปร A ซึ่งเป็นข้อมูลชนิดตัวเลขจำนวนเต็มที่มีค่าตัวเลขอยู่ในช่วง 37 ถึง 76 จะมีวิธีการสร้างเงื่อนไขเพื่อนำไปตรวจสอบและคำสั่ง If ถึง 2 เงื่อนไขโดยเงื่อนไขแรกจะตรวจดูค่าตัวเลขในตัวแปร A ที่มีค่ามากกว่าหรือเท่ากับตัวเลข 37 (มีรูปแบบการเขียนเพื่อนำไปใช้ในงานคือ a >= 37 และเงื่อนไขที่ 2 จะตรวจค่าตัวเลขของตัวแปร A ที่น้อยกว่าหรือเท่ากับตัวเลข 76 (มีรูปแบบของการเขียนเงื่อนไขสำหรับนำไปใช้ในคำสั่ง If ดังนี้ a<=76) ซึ่งในการตรวจสอบด้วยคำสั่ง If จำเป็นต้องนำเงื่อนไขแลกมาผูกร่วมกับเงื่อนไขที่ 2 จึงจะทำให้การตรวจสอบเงื่อนไขได้ตามจุดที่ตั้งขึ้นสิ่งที่น่าสนใจคือการผูกเงื่อนไขทั้ง2เข้าด้วยกัน ให้ทำงานผสมผสานกันเป็นอย่างดีโดยอาศัยการใช้ Combining Operator ซึ่งประกอบด้วย and และ or

โอเปอเรเตอร์ที่ทำหน้าที่ผูกเงื่อนไขของคำสั่ง If เข้าด้วยกันมี 3 รูปแบบดังนี้

1. รูปแบบการเขียนคำสั่ง If ร่วมกับ and operator
2. รูปแบบการเขียนคำสั่ง If ร่วมกับ or operator
3. รูปแบบการเขียนคำสั่ง If ร่วมกับ and และ or operator

รูปแบบการเขียนคำสั่ง if ร่วมกับ and opreator

บรรทัดที่ 1 a = 5

บรรทัดที่ 2 if a >= 2 and a <= 7:

บรรทัดที่ 3 print(‘ตัวเลขอยู่ในช่วง 2 ถึง 7 เท่านั้น’)

บรรทัดที่ 4 print(‘ทำเรื่องอื่นต่อไป’)

จากการใช้โอเปอเรเตอร์ and ในบรรทัดที่ 2 แสดงให้เห็นการทดสอบเงื่อนไขแรกสุดคือ a >= 2 และเงื่อนไข a <= 7 โดยจะนำผลลัพธ์ของทั้งสองเงื่อนไขมาเทียบคุณสมบัติของ and operator (คุณสมบัติของการ and operator จะได้ผลลัพธ์เป็นจริง (True) ก็ต่อเมื่อผลลัพธ์ของทั้งสองเงื่อนไขเป็นจริง (True) เท่านั้น) จากตัวอย่างคำสั่งในบรรทัดที่ 2 ตัวแปร a มีค่า 5 เมื่อนำมาตรวจสอบกับเงื่อนไขที่ 1 คือ a >= 2 หรือแทนค่า 5 ที่ตัวแปร a ของเงื่อนไขแรกจะได้ 5 >= 2 แน่นอนผลลัพธ์ของเงื่อนไขนี้เป็นจรอง (True) เช่นเดียวกับเงื่อนไขที่ 2 คือ a <= 7 แทนค่า 5 ที่ตัวแปร a ในเงื่อนไขจะได้ 5 <= 7 ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบเงื่อนไขที่ 2 ก็เป็นจริง (True) เช่นกัน

เมื่อนำผลลัพธ์ของทั้งสอง 2 เงื่อนไขมาใช้ร่วมกับ and ภายใต้คำสั่ง if ส่งผลให้การตรวจสอบเงื่อนไขทั้งหมดเป็นไปตามที่กำหนด หรือมีสถานการณ์ตรวจสอบเป็นจริง(True) ในที่นี้คำสั่งถัดไปหลังการตรวจสอบเงื่อนไขก็คือ คำสั่ง print บรรทัดที่ 3 นั่นเอง

เทคนิคการเขียน and operator ร่วมกับคำสั่ง if ใน Python บางรูปแบบจะมีลักษณะดังต่อไปนี้

รูปแบบคำสั่ง if ร่วมกับ and ปกติ

If a >= 2 and a <= 7

รูปแบบคำสั่ง if ร่วมกับ and operator วิธีลัด

If 2 >= a <= 7:

รูปแบบการเขียนคำสั่ง if ร่วมกับ or operator

บรรทัดที่ 1 a,b = 5,10

บรรทัดที่ 2 if a == 5 or b == 2:

บรรทัดที่ 3 print(‘ค่าในตัวแปร a คือ 5 หรือค่าในตัวแปร b มีค่าเป็น 2 เท่านั้น’)

บรรทัดที่ 4 print(‘ทำคำสั่งอื่นต่อไป’)

โอเปอเรเตอร์ or จะถูกนำมาใช้ร่วมกับการตรวจสอบเงื่อนไขภายใต้คำสั่ง if ก็ต่อเมื่อต้องการให้ผลลัพธ์ของเงื่อนไขใดเงื่อนไขหนึ่งเป็นจริง (True) ซึ่งเท่านี้ก็ส่งผลให้กลุ่มคำสั่งย่อยที่จะทำงานเมื่อผลลัพธ์มีค่าเป็นจริง (True) เกิดการทำงาน ยกตัวอย่างตามบรรทัดที่ 2 ค่าในตัวแปร a คือ 5 และค่าของตัวแปร b ที่มีค่า 10 เมื่อนำค่าในตัวแปร a มาตรวจสอบเงื่อนไข จะมีสถานะเป็นจริง (True) ตามเงื่อนไขกำหนดไว้ในขณะที่การตรวจสอบเงื่อนไขที่ 2 ผลลัพธ์ที่ได้จากการตรวจสอบจะมีสถานะเป็นเท็จ (False) เนื่องจากค่าในตัวแปร b ไม่ตรงตามเงื่อนไขที่กำหนดไว้ เมื่อนำผลลัพธ์ของทั้ง 2 เงื่อนไขมาใช้กับโอเปอเรเตอร์ or ในบรรทัดที่ 2 นี้จะทำ

ให้การตรวจสอบของคำสั่ง if นี้มีสถานะเป็นจริงหรือ True (เนื่องจากผลลัพธ์การตรวจสอบเงื่อนไขแรกสุดมีสถานะเป็นจริง) ซึ่งจะทำคำสั่งต่อไปในบรรทัดที่ 3 ที่อยู่ในพื้นที่กลุ่มคำสั่งย่อยของคำสั่ง if

รูปแบบการเขียนคำสั่ง if ร่วมกับ and และ or operator

บรรทัดที่ 1 a, b = 5, 10

บรรทัดที่ 2 if a >= 2 and a <= 7 or b == 10:

บรรทัดที่ 3 print('ค่าของตัวแปร a อยู่ในช่วง 2 ถึง 7')

บรรทัดที่ 4 print('หรือค่าในตัวแปร b มีค่าเท่ากับ 10')

บรรทัดที่ 5 print('ทำคำสั่งอื่นต่อไป')

การใช้คำสั่ง If สำหรับการตรวจสอบเงื่อนไขที่จำนวนของเงื่อนไขอาจจะมีมากกว่า 1 เงื่อนไขจึงมีความจำเป็นที่จะต้องผูกเงื่อนไขเข้ากับการใช้ and หรือ or Operator ทั้งนี้ต้องคำนึงถึงคุณสมบัติการทำงานของ operator ที่นำมาใช้สำหรับการผูกเงื่อนไขบางกรณีจำนวนเงื่อนไขมีมากกว่า 2 เงื่อนไขจำนวนของ operator ที่จะนำมาผูกเงื่อนไขก็จะมีมากขึ้นอีกทั้งยังสามารถใช้ Operator ทั้งสองชนิดผสมผสานในการตรวจสอบภายใต้คำสั่ง is เดี่ยวตามตัวอย่างการใช้คำสั่งบรรทัดที่ 2 จะเห็นได้ว่ามีจำนวนเงื่อนไขถึง 3 เงื่อนไขภายใต้การตรวจสอบของคำสั่ง If โดยที่โจทก์ถูกกำหนดให้เงื่อนไขที่ 1(a >= 2) และเงื่อนไขที่ 2 (a <= 7) จำเป็นต้องผ่านตามกำหนดหรือมีผลสถานะการตรวจสอบเงื่อนไขเป็นจริง (True) ทั้งคู่ความเหมาะสมต่อการใช้ Operator ก็ต้องเลือก and operator มาผูกเงื่อนไขที่ 1 และ 2 เข้าด้วยกันจึงจะถูกต้องตามความต้องการของโจทย์ในหัวข้อนี้สำหรับเงื่อนไขที่ 3 (b == 10)ถ้าผลลัพธ์การตรวจสอบเงื่อนไขเป็นจริง(True) คำสั่ง if ก็จะสั่งให้ปุ่มคำสั่งย่อยในพื้นที่ที่สถานการณ์ตรวจสอบเป็นจริง(True) หรือตรงตามที่เงื่อนไขกำหนดให้ทำงานได้ทันที

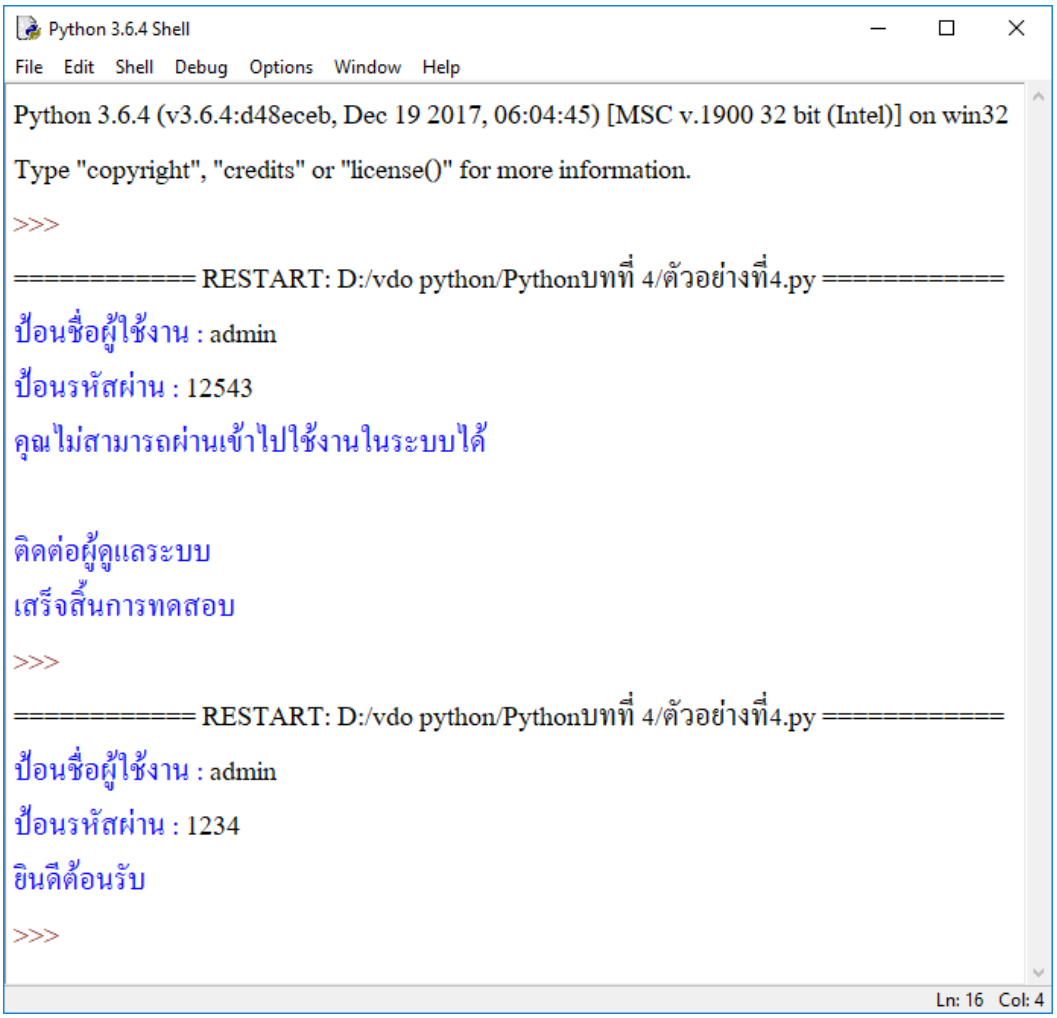
เป็นที่น่าสังเกตสำหรับการทำงานของเงื่อนไขที่ 3(b == 10) ที่มีความเป็นอิสระในการตรวจสอบของคำสั่ง If โดยไม่ต้องไปยุ่งกับเงื่อนไขอื่นๆทีคนกันในคำสั่ง If นี้ Combining Operator ควรจะใช้กับการทำงานของเงื่อนไขที่มีความเป็นอิสระแบบนี้ก็คือ or operator

พิมพ์คำสั่งต่อไปนี้สำหรับทดลองการใช้ Combining Operator ร่วมกับคำสั่ง if โดยเขียนคำสั่งลงในโมดูลของ python (ที่จอภาพของ Python Shell เลือกเมนู file และเลือกเมนูย่อยที่ชื่อ New เพื่อกำหนดเปิดโมดูล python)

ตัวอย่างที่ 1 การใช้ and operator

```
username = input('ป้อนชื่อผู้ใช้งาน : ')
password=input('ป้อนรหัสผ่าน')
if username is None :
   print('ป้อนชื่อด้วยค่ะ')
elif password is None:
   print('ป้อนรหัสเท่านั้น')
```

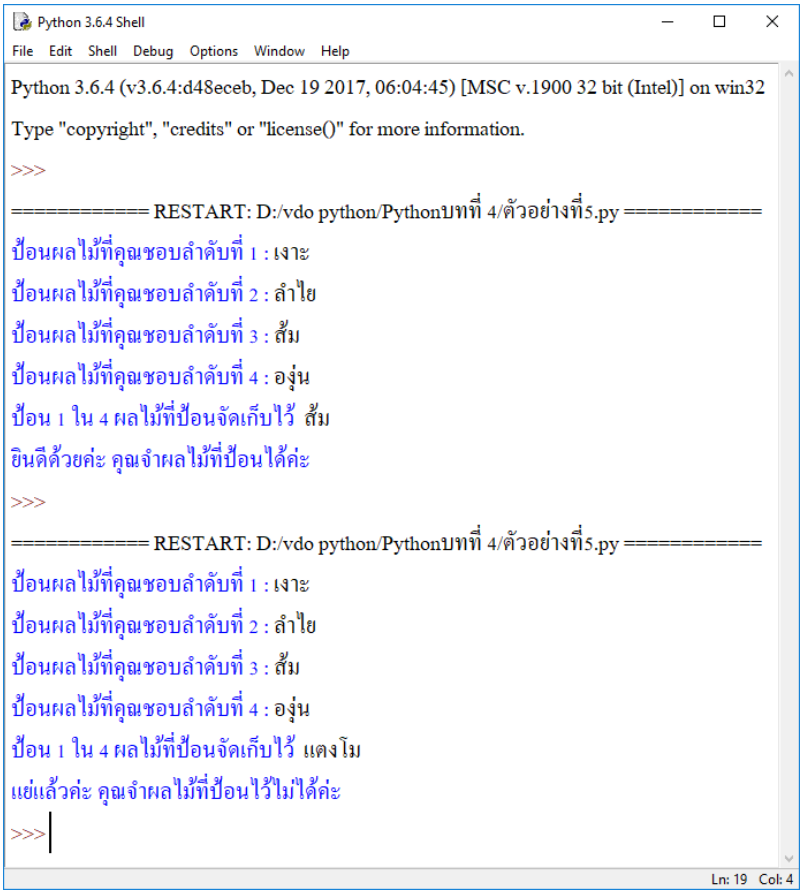
```
elif username == 'admin' and password == 'password':
    print('ยินดีต้อนรับ')
else:
    print('คุณไม่สามารถผ่านเข้าไปใช้งานในระบบได้')
    print('\nติดต่อผู้ดูแลระบบ')
    print('เสร็จสิ้นการทดสอบ')
```



การทดลองในตัวอย่างนี้เป็นการใช้โอเปอเรเตอร์ and สำหรับตรวจสอบค่าในตัวแปร username และ password เพื่อเข้าสู่การใช้งานในระบบตามลำดับ โดยที่การล็อกอินเข้าสู่ระบบจะสำเร็จก็ต่อเมื่อชื่อผู้ใช้งานเป็น admin และรหัสผ่านเป็นคำว่า 1234

ตัวอย่างที่ 2 การใช้ or operator

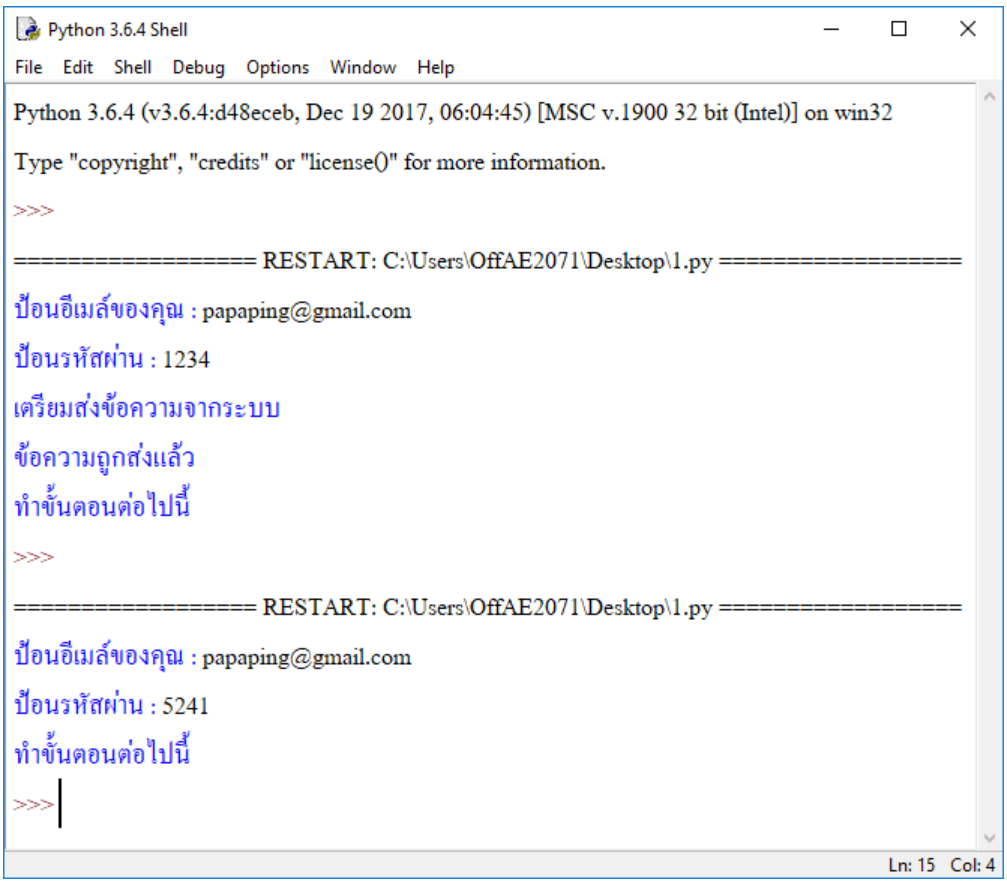
```
a = []
b = input('ป้อนผลไม้ที่คุณชอบลำดับที่ 1 : ')
a.append(b)
b = input('ป้อนผลไม้ที่คุณชอบลำดับที่ 2 : ')
a.append(b)
b = input('ป้อนผลไม้ที่คุณชอบลำดับที่ 3 : ')
a.append(b)
b = input('ป้อนผลไม้ที่คุณชอบลำดับที่ 4 : ')
a.append(b)
b = input('ป้อน 1 ใน 4 ผลไม้ที่ป้อนจัดเก็บไว้')
if b == a[0] or b == a[1] or b == a[2] or b == a[3]:
    print('ยินดีด้วยค่ะ คุณจำผลไม้ที่ป้อนได้ค่ะ')
else :
    print('แย่แล้วค่ะ คุณจำผลไม้ที่ป้อนไว้ไม่ได้ค่ะ')
```



ลองใช้โอเปอเรเตอร์ or สำหรับตรวจค่าผลไม้ที่ต้องการป้อนเก็บไว้ในตัวแปร a (เป็นตัวแปร Sequence แบบ List เป็นจำนวน 4 ชนิด จากนั้นก็ทำการทดสอบความจำโดยการเลือกพิมพ์ชื่อผลไม้ที่เคยป้อนเก็บไว้ 1 ชนิดจากทั้งหมด 4 ชนิด

ตัวอย่างที่ 3 การใช้ and และ or operator

```
a = input('ป้อนอีเมล์ของคุณ : ')
if '@' in a and '.' in a or a == 'admin' :
    b = input('ป้อนรหัสผ่าน : ')
    if b == '1234' :
        print('เตรียมส่งข้อความจากระบบ')
        print('ข้อความถูกส่งแล้ว')
else :
        print('อีเมล์คุณมีรูปแบบการเขียนที่ไม่ถูกต้อง')
        print('%s กรุณาตรวจใหม่อีกครั้ง' %a)
print('ทำขั้นตอนต่อไปนี้')
```



การทดลองสมมติให้ใช้อีเมล์ของผู้ใช้ login เข้าสู่ระบบโดยตรวจสอบรูปแบบของการเขียนอีเมล์จากค่าในตัวแปรเอด้วยการสังเกตเครื่องหมาย @ และ . ที่เป็นองค์ประกอบของชื่ออีเมล์ด้วยเงื่อนไขที่ 1 (‘@’ in a) และเงื่อนไขที่ 2 (‘.’ in a) เทคนิคการเขียนคำสั่งด้วยการใช้ in ซึ่งทำหน้าที่ค้นหาตัวอักษรเครื่องหมายหรือค่าที่ต้องการค้นหา ฯลฯ ผลลัพธ์จากการค้นหาด้วยคำสั่ง in ถ้าไม่เจอจะให้ค่าเท็จ (False) และให้ค่าที่เป็นจริง (True) เมื่อเจอสิ่งที่ต้องการค้นหาจากค่าในตัวแปรโจทย์ของการตรวจสอบรูปแบบของ e mail คือ เงื่อนไขที่ 1 (‘@’ in a) และเงื่อนไขที่ 2 (‘.’ in a) จำเป็นต้องผูกกันด้วย operator an เพราะต้องการผลลัพธ์ที่มีสถานะจริง (True) ทั้งคู่เสมอในขณะเดียวกันโจทก์ก็ยังตั้งเงื่อนไขพิเศษสำหรับการ login เข้าสู่ระบบในเงื่อนไขที่ 3 (a == ‘admin’) ที่มีความเป็นอิสระจากเงื่อนไขอื่นในคำสั่ง If นี้นั่นคือ Operator ที่เหมาะสมสำหรับเงื่อนไขที่ 3 ก็คือ or

บทที่ 5 คำสั่งการวนซ้ำ

ด้วยพื้นฐานของการเขียนโปรแกรม จะมีกลุ่มคำสั่งหลักอีกประเภทหนึ่งที่ทำงานร่วมกับกลุ่มคำสั่งย่อยโดยบังคับให้คำสั่งย่อยทั้งหมดทำงานตามจำนวนครั้งที่ระบุในคำสั่งหลัก เรามักจะเรียกคำสั่งหลักที่มีลักษณะการทำงานเช่นนี้ว่า คำสั่งการวนทำซ้ำ (Repetition) สำหรับภาษา Python กลุ่มคำสั่งการวนทำซ้ำจะประกอบด้วยคำสั่ง for และคำสั่ง while ก่อนที่จะอธิบายลงไปในรายละเอียดของทั้ง 2 คำสั่ง ลองมาดูประโยชน์ของการใช้คำสั่งการวนทำซ้ำ และการตัดสินใจที่จะนำคำสั่งการวนทำซ้ำมาใช้ระหว่างการพัฒนางาน (การทดลองคำสั่งจะทำการเขียนคำสั่งลงในโมเดล Python โดยไปที่เมนู File เลือกเมนูย่อย New สำหรับเปิดโมดูล)

ตัวอย่างที่ 1 เป็นคำสั่งรับข้อมูลแบบลุกทุ้งลูกทุ่ง

```
a = 1
c = []
print (‘กรุณาป้อนชื่อผลไม้ 6 ชื่อ’)
b = input(‘%d ป้อนชื่อผลไม้สุดโปรดของคุณ’ %a)
c.append(b)
a = a + 1
b = input(‘%d ป้อนชื่อผลไม้สุดโปรดของคุณ’ %a)
c.append(b)
a + = 1
b = input(‘%d ป้อนชื่อผลไม้สุดโปรดของคุณ’ %a)
c.append(b)
a + = 1
b = input(‘%d ป้อนชื่อผลไม้สุดโปรดของคุณ’ %a)
c.append(b)
a + = 1
b = input(‘%d ป้อนชื่อผลไม้สุดโปรดของคุณ’ %a)
c.append(b)
print (‘ผลไม้โปรดของคุณคือ %s’%c)
print(‘จบคำสั่ง’)
```

```
a=1
c=1
print('%d ป้อนชื่ออาหารสุดโปรดของคุณ '%a)
b= input('%d ป้อนชื่ออาหารสุดโปรดของคุณ '%a)
c.append(b)
a = a+1
b = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ ' %a)
c.append(b)
a += 1
b = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ ' %a)
c.append(b)
a += 1
b = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ ' %a)
c.append(b)
a += 1
b = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ ' %a)
c.append(b)
a += 1
b = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ ' %a)
c.append(b)
print(' อาหารโปรดของคุณคือ %s' %c)
print('จบคำสั่ง')
```

```
กรุณาป้อนชื่ออาหารหลัก 6 ชื่อ
1 ป้อนชื่ออาหารสุดโปรดของคุณ ปลาเผา
2 ป้อนชื่ออาหารสุดโปรดของคุณ คั่วกลิ้ง
3 ป้อนชื่ออาหารสุดโปรดของคุณ คอหมูย่าง
4 ป้อนชื่ออาหารสุดโปรดของคุณ ต้มขมเนื้อ
5 ป้อนชื่ออาหารสุดโปรดของคุณ เส้นใหญ่เย็นตาโฟต้มยำ
6 ป้อนชื่ออาหารสุดโปรดของคุณ หมูกรอบผัดขี้เมา
 อาหารโปรดของคุณคือ ['ปลาเผา', 'คั่วกลิ้ง', 'คอหมูย่าง', 'ต้มขมเนื้อ', 'เส้นใหญ่เย็นตาโฟต้มยำ', 'หมูกรอบผัดขี้เมา']
จบคำสั่ง
>>>
```

การทำงานของกลุ่มคำสั่งในตัวอย่างที่ 1 จะรอให้ผู้ใช้งานป้อนชื่ออาหารให้ครบ 6 ชื่อ เมื่อครบตามจำนวนรายชื่ออาหาร ก็จะทำการพิมพ์รายการชื่อเหล่านั้นเป็นขั้นตอนสุดท้ายของการทำงานในโปรแกรมนี้ ลองมาดูลักษณะการเขียนคำสั่งของตัวอย่างที่ 1 พบว่ามีการใช้คำสั่งการรับค่าจากการป้อนผ่านคีย์บอร์ด คือ คำสั่ง input และค่าข้อมูลที่ได้จะถูกเก็บลงในตัวแปร b ชั่วคราว (b = input ('%d ป้อนชื่ออาหารสุดโปรดของคุณ%a) ซึ่งค่าข้อมูลในตัวแปร b ก็คือชื่ออาหารโปรดที่ผู้ใช้จะต้องทำการป้อน 1 ชื่อ หลังจากได้ชื่ออาหารก็

จะนำไปเก็บรวมกันในตัวแปร c ที่เป็นชนิด Sequence แบบ List โดยการใช้คำสั่ง c.append(b) เพียงเท่านี้ก็ครบตามกระบวนการใช้งานสำหรับการป้อนชื่ออาหาร และจัดเก็บไว้ในโปรแกรม แต่โจทย์ต้องการชื่ออาหารทั้งหมด 6 รายชื่อ ด้วยวิธีการแบบเดิมคือ การป้อนชื่ออาหารตามกระบวนการที่ได้อธิบาย และคำสั่งที่นำมาใช้สำหรับการเขียนโปรแกรมก่อนหน้านี้ไปแล้ว ดังนั้นกลุ่มคำสั่งชุดที่ 2,3,4 ไปจนถึง 6 หรือมากกว่านี้ก็จะใช้เหมือนกันกลุ่มคำสั่งในชุดที่ 1 ซึ่งถ้าดูตามตัวอย่างที่ 1 จะเห็นกลุ่มคำสั่งที่ทำงานซ้ำๆ กัน โดยแบ่งออกเป็น 6 ชุด ประกอบด้วย คำสั่งต่อไปนี้

```
b = input('%d ป้อนชื่ออาหารสดโปรดของคุณ' %a)
c.append(b)
```

คราวนี้ลองมาดูประโยชน์ของการนำคำสั่งการวนทำซ้ำของภาษา Python มาดัดแปลงลงในกลุ่มคำสั่งที่ใช้ในตัวอย่างที่ 1 ให้สังเกตการณ์เปลี่ยนแปลงของคำสั่ง หลังจากการนำคำสั่งการวนซ้ำมาใช้งานร่วมกัน เนื่องจากคำสั่งการวนทำซ้ำของภาษา Python มี 2 คำสั่ง วิธีหรือรูปแบบการเขียนคำสั่งเพื่อใช้งานจะคล้ายกัน เพียงแต่แตกต่างกันที่วิธีการสิ้นสุดการวนทำซ้ำของกลุ่มคำสั่งย่อย ถ้าหากเป็นการทำซ้ำที่มีจุดเริ่มต้น สมมุติว่าเริ่มจากตัวเลข 1 และกำหนดตัวเลข 5 ให้เป็นจุดสิ้นสุด ซึ่งจะมีความชัดเจนในการกำหนดจุดเริ่มต้นและจุดสิ้นสุด คำสั่งการวนทำซ้ำก็ต้องใช้คำสั่ง for โดยการมีรูปแบบการเขียนตามตัวอย่างที่ 2 ยัง เหลือวิธีการสิ้นสุดการวนทำซ้ำจากคำสั่ง while ซึ่งอาศัยการกำหนดเงื่อนไข และตรวจสอบเงื่อนไขในแต่ละรอบของการวนทำซ้ำ เมื่อผลลัพธ์ของการตรวจสอบเงื่อนไขเป็นเท็จ (False) ก็จะถือว่าสิ้นสุดการวนทำซ้ำ ตัวอย่างที่ 3 จะแสดงวิธีการเขียนคำสั่ง while และกำหนดเงื่อนไขสำหรับการสิ้นสุดการวนทำซ้ำของกลุ่มคำสั่งย่อย

ตัวอย่างที่ 2 เป็นคำสั่งรับข้อมูลโดยใช้คำสั่งการวนทำซ้ำ for มาแทนการเขียนแบบลูกทุ้งลูกทุ่งใน ตัวอย่างที่ 1

```
บรรทัดที่ 1 c = []
บรรทัดที่ 2 for a in [1,2,3,4,5] :
บรรทัดที่ 3     b = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ' %a)
บรรทัดที่ 4     c.append(b)
บรรทัดที่ 5 print('อาหารโปรดของคุณคือ %s ' %c)
บรรทัดที่ 6 print('จบคำสั่ง')
```

สังเกตรูปแบบการเขียนคำสั่ง for ซึ่งจะเริ่มจากบรรทัดที่ 2 โดยมีกลุ่มคำสั่งย่อยที่จะทำการวนทำซ้ำคือ บรรทัดที่ 2 และ 3 ที่มีจุดเริ่มต้นจากตัวเลข 1 (แบบเรียงจากน้อยไปมาก หรือจะแก้จากมากไปน้อยก็พิมพ์ 5,4,3,2,1 ก็ได้ หรือชอบแบบผาดโผน เช่น 1,9,17,4,100 เป็นต้น) ในการวนทำซ้ำรอบแรกซึ่งจัดเก็บค่าตัวเลขไว้ที่ตัวแปร a เมื่อเริ่มการวนทำซ้ำในรอบที่ 2 หรือรอบถัดไปก็จะอ่านค่าตัวเลขที่เก็บเรียงไว้คือ 2,3,4,5 จึงจะ

สิ้นสุดการวนซ้ำของคำสั่ง for อีกเทคนิคหนึ่งที่นิยมนำมาใช้ร่วมกับการกำหนดตัวเลขเริ่มต้นไปจนถึงตัวเลขที่สิ้นสุด ด้วยการใช้ฟังก์ชัน range ที่มีรูปแบบการเขียนฟังก์ชัน ดังนี้

แบบที่ 1 range (1,5) มีค่าเท่ากับ [1,2,3,4,5] เหมือนที่ใช้ในบรรทัดที่ 2 เมื่อนำฟังก์ชัน range มาร่วมกับคำสั่ง for จะมีรูปแบบการเขียน ดังนี้

```
c = []
for a in range(1,5) :
    b = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ' %a)
    c.append(b)
print('อาหารโปรดของคุณคือ %s' %c)
print('จบคำสั่ง')
```

แบบที่ 2 range (5,1, -1) มีค่าเท่ากับ [5,4,3,2,1] เป็นการกำหนดตัวเลขจากมากไปน้อย จะให้ลดทีละ 1 ก็กำหนด -1 ในอาร์กิวเมนต์ตัวที่ 3 ซึ่งทำหน้าที่กำหนดการเพิ่มหรือลดค่าตัวเลขของฟังก์ชัน range ฟังก์ชันจะกำหนดค่าเป็น 1 ไว้เสมอ ซึ่งหมายถึงจะทำการเพิ่มทีละ 1 โดยอัตโนมัติ)

```
c = []
for a in range (5,1,-1) :
    b = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ' %a)
    c.append(b)
print('อาหารโปรดของคุณคือ %s' %c)
print('จบคำสั่ง')
```

แบบที่ 3 แสดงการใช้คำสั่ง while สำหรับกลุ่มคำสั่งย่อยในการวนทำซ้ำและสิ้นสุดด้วยการตรวจสอบเงื่อนไข

บรรทัดที่ 1 a = 1

บรรทัดที่ 2 c = []

บรรทัดที่ 3 while a < 5 :

บรรทัดที่ 4 b = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ' %a)

บรรทัดที่ 5 c.append(b)

บรรทัดที่ 6 a += 1

บรรทัดที่ 7 print('อาหารโปรดของคุณคือ %s' %c)

บรรทัดที่ 8 print('จบคำสั่ง')

ในบรรทัดที่ 3 เป็นรูปแบบการเขียนคำสั่งการวนซ้ำด้วยคำสั่ง while พร้อมเงื่อนไขที่กำหนดขึ้นเพื่อหยุดการวนทำซ้ำ ซึ่งต้องการผลลัพธ์ของการตรวจสอบเงื่อนไขที่เป็นเท็จ (False) เท่านั้น จากตัวอย่างจะเห็นว่าค่าในตัวแปร a จะเริ่มที่ 1 (กำหนดที่บรรทัดที่ 1) เมื่อนำค่าในตัวแปรมาตรวจสอบเงื่อนไข a < 5 ของคำสั่ง while ผลลัพธ์ที่ได้จากการตรวจสอบจะเป็นจริง (เนื่องจากค่าในตัวแปร a น้อยกว่า 5 ตามเงื่อนไข) กระบวนการวนทำซ้ำกับกลุ่มคำสั่งย่อยในบรรทัดที่ 4 ถึง 6 ก็จะเริ่มต้นทำงานแต่ละรอบจนกว่าจะจบตามเงื่อนไขของคำสั่ง while คราวนี้ลองมาดูตัวอย่างเพิ่มเติมกรณีมีเงื่อนไขที่สามารถจบการวนทำซ้ำมากกว่า 1 เงื่อนไข

ตัวอย่างที่ 4 แสดงการใช้คำสั่ง while ที่มีมากกว่า 1 เงื่อนไขสำหรับการวนทำซ้ำ

บรรทัดที่ 1 a = 1

บรรทัดที่ 2 c = []

บรรทัดที่ 3 while a < 5 or b != 'exit' :

บรรทัดที่ 4 b = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ' %a)

บรรทัดที่ 5 c.append(b)

บรรทัดที่ 6 a += 1

บรรทัดที่ 7 print('อาหารโปรดของคุณคือ %s' %c)

บรรทัดที่ 8 print('จบคำสั่ง')

จากตัวอย่างที่ 4 ในบรรทัดที่ 3 จะมีเงื่อนไข b <> 'exit' เพิ่มเติม ซึ่งมีประโยชน์สำหรับการหยุดการป้อนชื่ออาหารสุดโปรด โดยการพิมพ์คำว่า exit แทนชื่ออาหาร เท่านี้การวนทำซ้ำของคำสั่ง while ก็จะสิ้นสุดการทำงาน

พอเห็นตัวอย่างที่ 4 ทำให้นึกถึงเทคนิคการเขียนคำสั่งการวนทำซ้ำที่เป็นลักษณะการวนทำซ้ำแบบไม่มีวันสิ้นสุดหรือ Infinity Repetition โดยใช้คำสั่ง while ซึ่งเป็นที่นิยมใช้แพร่หลายกับการเขียนในภาษา Python ลองดูเทคนิคการกำหนดเงื่อนไขในลักษณะ Infinity Repetition ตามตัวอย่างต่อไปนี้

ตัวอย่างที่ 5 การใช้คำสั่ง while กับการกำหนดเงื่อนไขในลักษณะ Infinity Repetition (ข้อควรระวังของตัวอย่างนี้คือ คุณจะต้องพิมพ์ชื่ออาหารจนเมื่อยมือแน่ๆ เพราะคุณไม่สามารถออกจากคำสั่งการวนทำซ้ำได้)

บรรทัดที่ 1 a = 1

บรรทัดที่ 2 c = []

บรรทัดที่ 3 while True :

บรรทัดที่ 4 b = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ' %a)

บรรทัดที่ 5 c.append(b)

บรรทัดที่ 6 a += 1

บรรทัดที่ 7 print('อาหารโปรดของคุณคือ %s' %c)

บรรทัดที่ 8 print('จบคำสั่ง')

จากตัวอย่างที่ 5 บรรทัดที่ 3 คำสั่ง while ถูกกำหนดเงื่อนไขด้วยค่าประเภทบูลีน (Boolean) ที่เป็นค่าจริง (True) ซึ่งเป็นรูปแบบการเรียงเงื่อนไขที่สั้นมาก แต่ส่งผลให้การวนทำซ้ำจะกลายเป็นแบบ Infinity Repetition ไปเลย คือ จะทำคำสั่งกลุ่มย่อยในบรรทัดที่ 4 และ 5 ที่ทำหน้าที่รับชื่ออาหารโปรดของคุณและจัดเก็บไว้ในตัวแปร c ไปเรื่อยๆ วิธีหยุดก็ต้องหาทางหยุดด้วยตัวเอง เทคนิคใคร เทคนิคมัน ถ้าใครไม่รู้จริง กด Ctrl + C ก็แล้วกัน บาคนสงสัยกับเทคนิคการเขียนในลักษณะ Infinity Repetition จะนำไปใช้ประโยชน์อะไรในเชิงการใช้งาน แถมยังไม่สามารถหยุดโปรแกรมจากคำสั่งปกติได้อีก เรื่องแรกมาดูประโยชน์ของการใช้เทคนิคแบบนี้ ตามตัวอย่างที่ 5 จะเห็นว่ากรป้อนชื่ออาหารและการจัดค่าลงในตัวแปร c สามารถทำไปเรื่อยไม่ต้องอยู่ภายใต้ขอบเขตจำนวนรอบของค่าในตัวแปร a เหมือนในตัวอย่างที่ 4 เรื่องที่สองของเทคนิคการเขียนคำสั่งวนทำซ้ำแบบไม่มีวันสิ้นสุด คือ การหยุดการวนทำซ้ำจากคำสั่งที่ชื่อว่า break (เมื่อคำสั่ง break ทำงานจะเปลี่ยนสถานะเงื่อนไขของคำสั่ง while จาก True ให้เป็น False โดยอัตโนมัติซึ่งจะทำให้การวนทำซ้ำของคำสั่ง while หยุดกระบวนการทำงานทันที) สำหรับการใช้คำสั่ง break เพื่อหยุดการวนทำซ้ำของคำสั่ง while จะเขียนได้ตามตัวอย่างข้างล่างนี้

บรรทัดที่ 1 a = 1

บรรทัดที่ 2 c = []

บรรทัดที่ 3 while True :

บรรทัดที่ 4 d = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ' %a)

บรรทัดที่ 5 if d = = 'exit':

บรรทัดที่ 6 break

บรรทัดที่ 7 c.append(d)

บรรทัดที่ 8 a += 1

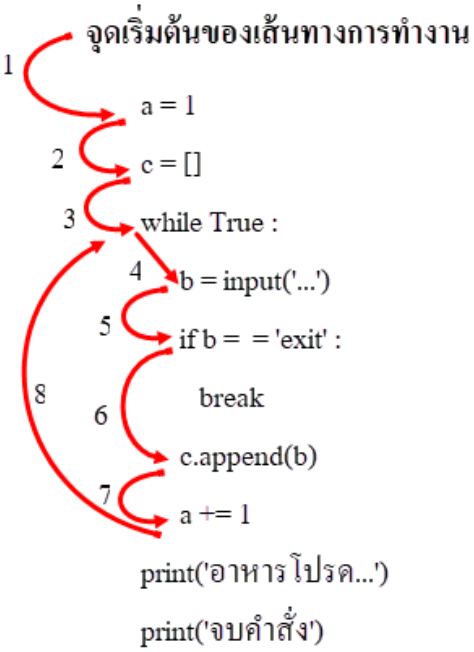
บรรทัดที่ 9 print('อาหารโปรดของคุณคือ %s' %c)

บรรทัดที่ 10 print('จบคำสั่ง')

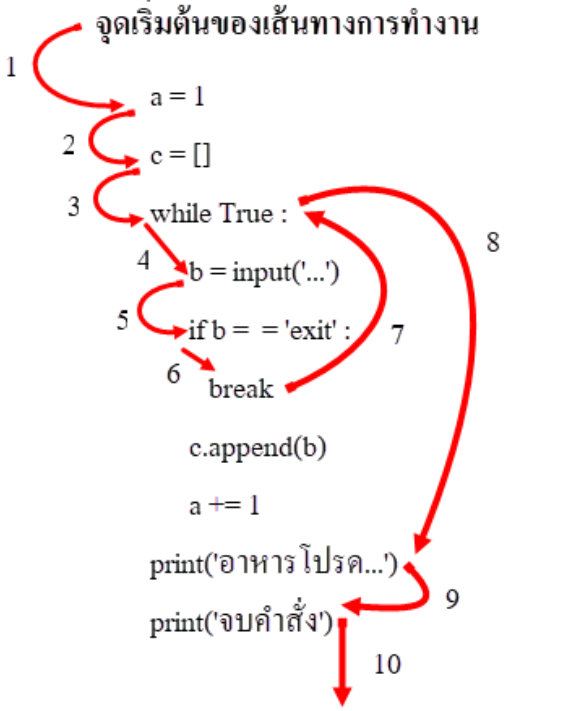
```
a=1
c=[]
while True :
    d = input('%d ป้อนชื่ออาหารสุดโปรดของคุณ '%a)
    if d == 'exit' :
        break
    c.append(d)
    a += 1
    print('อาหารโปรดของคุณคือ %s' %c)
    print('จบคำสั่ง')
```

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:\share\ดาว\test2.py ==========
1 ป้อนชื่ออาหารสุดโปรดของคุณ เส้นใหญ่เย็นตาโฟ
อาหารโปรดของคุณคือ ['เส้นใหญ่เย็นตาโฟ']
จบคำสั่ง
2 ป้อนชื่ออาหารสุดโปรดของคุณ ผัดกระเพรา
อาหารโปรดของคุณคือ ['เส้นใหญ่เย็นตาโฟ', 'ผัดกระเพรา']
จบคำสั่ง
3 ป้อนชื่ออาหารสุดโปรดของคุณ ไข่เจียว
อาหารโปรดของคุณคือ ['เส้นใหญ่เย็นตาโฟ', 'ผัดกระเพรา', 'ไข่เจียว']
จบคำสั่ง
4 ป้อนชื่ออาหารสุดโปรดของคุณ ข้าวต้มหมู
อาหารโปรดของคุณคือ ['เส้นใหญ่เย็นตาโฟ', 'ผัดกระเพรา', 'ไข่เจียว', 'ข้าวต้มหมู']
จบคำสั่ง
5 ป้อนชื่ออาหารสุดโปรดของคุณ
```

ตัวอย่างนี้ จะแก้ไขปัญหาการวนทำซ้ำแบบไม่มีวันสิ้นสุดที่เกิดขึ้นในตัวอย่างที่ 5 โดยจะเพิ่มคำสั่ง if เพื่อตรวจสอบค่าในตัวแปร b (บรรทัดที่ 5) มีการป้อนคำว่า exit เพื่อต้องการหยุดการวนทำซ้ำเกี่ยวกับการป้อนชื่ออาหารสุดโปรด ด้วยคำสั่ง break ในบรรทัดที่ 6 และเมื่อไรที่คำสั่ง break ทำงานก็จะกระโดดกลับไปที่บรรทัดที่ 3 ที่เป็นคำสั่ง while พร้อมกับเปลี่ยนสถานะของเงื่อนไขในการวนทำซ้ำจาก True ให้เป็น False เพียงเท่านี้คำสั่ง while ก็เลิกทำการวนทำซ้ำ ในกลุ่มคำสั่งย่อยภายในทั้งหมด (บรรทัดที่เป็นกลุ่มคำสั่งย่อยเริ่มจากบรรทัดที่ 5 ถึง 8) และเริ่มในคำสั่งถัดไปที่อยู่ในระดับเดียวกันกับคำสั่ง while คือ บรรทัดที่ 9 เป็นต้นไป เพื่อให้เข้าใจลักษณะการทำงานของคำสั่งการวนทำซ้ำร่วมกับคำสั่ง break โดยศึกษาจากเส้นทางการทำงานของคำสั่งดังรูปที่ 5.1



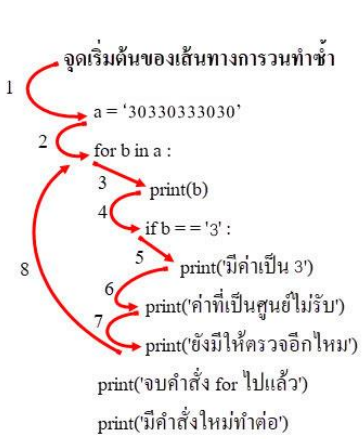
แสดงเส้นทางการทำงานของคำสั่งการวนทำซ้ำแบบไม่มีวันสิ้นสุด



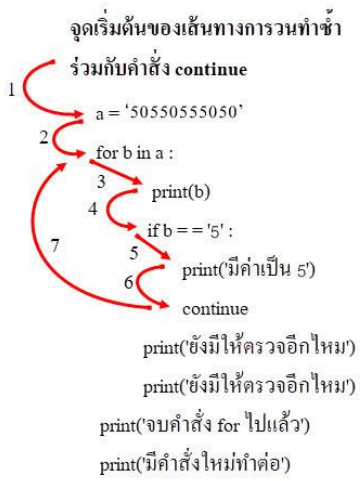
แสดงเส้นทางการทำงานของคำสั่งการวนทำซ้ำ เมื่อค่าในตัวแปร b มีการป้อนค่า exit และตรงตามเงื่อนไขของคำสั่ง if จึงทำให้คำสั่ง break ทำงาน

รูปที่ 5.1 แสดงเส้นทางการทำงานของคำสั่งการวนทำซ้ำแบบไม่มีวันสิ้นสุดและวิธีการหยุดการวนทำซ้ำด้วยคำสั่ง break

อีกคำสั่งหนึ่งที่นิยมนำมาใช้ร่วมกับคำสั่งการวนทำซ้ำ (for และ while) คือ คำสั่ง continue ที่มีรูปแบบการใช้งานคล้ายๆ คำสั่ง break แต่การทำงานจะแตกต่างกันตรงที่คำสั่ง continue จะตัดขึ้นรอบใหม่ของการวนทำซ้ำ เพื่อให้ง่ายต่อการทำความเข้าใจการทำงานของคำสั่ง continue ขอให้ดูจากรูปที่ 5.2 ที่เปรียบเทียบการทำงานของคำสั่งการวนทำซ้ำที่ไม่มีคำสั่ง continue และที่มีคำสั่ง continue



แสดงเส้นทางการทำงานปกติของกลุ่มคำสั่งย่อยภายใต้คำสั่งการวนทำซ้ำหลักคือคำสั่ง for



แสดงเส้นทางการทำงานของกลุ่มคำสั่งย่อยที่มีคำสั่ง continue ใช้งานร่วมในกลุ่ม และผลจากการทำงานของคำสั่ง for เพื่อขึ้นรอบใหม่ทันที โดยไม่สนใจคำสั่งย่อยในกลุ่มที่เหลือหลังคำสั่ง continue จะว่าไปก็เหมือนการตัดตอนคำสั่งย่อยเพื่อทำงานรอบใหม่ไปเลย

รูปที่ 5.2 แสดงเส้นทางการเปรียบเทียบการทำงานของการวนทำซ้ำปกติ และเพิ่มคำสั่ง continue ในกลุ่มคำสั่งย่อยของคำสั่งวนทำซ้ำ for

++++++++++++++++++++++++++รูปที่ 007,008,009 และ 010 ++++++++++++++++++++

สรุปได้ว่าคำสั่งการวนทำซ้ำ ซึ่งประกอบด้วยคำสั่ง for และ while มีความเหมาะสมต่อการนำมาใช้กับกลุ่มคำสั่งย่อยที่ต้องการวนทำซ้ำ โดยที่คำสั่งทั้งคู่จะมีความแตกต่างกันที่วิธีการหยุดวนทำซ้ำ ที่มีข้อสังเกตจากลักษณะของเงื่อนไขคู่กับคำสั่งวนทำซ้ำ เช่น เงื่อนไขกำหนดการหยุดวนทำซ้ำที่ชัดเจน สมมุติเริ่มนับตัวเลขจาก 4 ไปจนถึง 20 เพื่อทำคำสั่งย่อยเป็นจำนวน 17 รอบ เงื่อนไขแบบนี้เหมาะกับคำสั่ง for เป็นอย่างยิ่ง แต่ถ้าเงื่อนไขของการหยุดวนทำซ้ำมีรูปแบบคือ a == 'จบ' หรือ a > 500 หรืออะไรที่มีเครื่องหมาย =,>,< ทำนองนี้ ต้องใช้คำสั่ง while เท่านั้น นอกจากนี้ยังมีคำสั่งพิเศษที่ใช้ภายในคำสั่ง for และ while คือ คำสั่ง break และ continue ที่มีรูปแบบการเขียนใช้งานคล้ายกัน คือต้องใช้ภายในร่วมกับกลุ่มคำสั่งย่อยของคำสั่ง for และ while เท่านั้น ทั้งสองคำสั่งพิเศษ (break และ continue) เมื่อถึงเวลาทำงานจะกระโดดไปที่คำสั่ง for หรือ while ทันที ที่จุดนี้จะแตกต่างกับที่คำสั่ง break จะหยุดการวนทำซ้ำ และออกจากคำสั่ง for หรือ while ไปทำคำสั่งอื่นๆ ภายนอกต่อไป แต่คำสั่ง continue จะเริ่มรอบใหม่สำหรับทำงานของกลุ่มคำสั่งย่อยภายในของคำสั่ง for หรือ while ในขณะนั้นต่อไป

ทดลองพิมพ์คำสั่ง for ร่วมกับคำสั่งพิเศษ break และ continue ตามตัวอย่างต่อไปนี้

ตัวอย่างที่ 6

```
บรรทัดที่ 1    a = ['fc shirts','fcskirt','fc pants','fc hat']
บรรทัดที่ 2    brand = []
บรรทัดที่ 3    for c in a :
บรรทัดที่ 4        if 'fc' in c :
บรรทัดที่ 5            brand.append(c[2:])
บรรทัดที่ 6            print('Fashion clothing %s' %c[2:])
บรรทัดที่ 7            if 'skirt' in c :
บรรทัดที่ 8                print('No discounts.!!')
บรรทัดที่ 9                continue
บรรทัดที่ 10           elif 'hat' in c :
บรรทัดที่ 11               print('Waiting to order')
บรรทัดที่ 12               break
บรรทัดที่ 13       print('Waiting to order')
บรรทัดที่ 14   print('ร้านนี้ขายเสื้อผ้าแฟชัน %s' %brand)
บรรทัดที่ 15   print('ทำคำสั่งต่อไปเลยคะ !!!')
```

++++++++++++++++++++++รูปโคด 011 และ 012++++++++++++++++++++++++++++++++

จากตัวอย่างที่ 6 จะเป็นการค้นหาและจัดเก็บยี่ห้อเสื้อผ้าลงในตัวแปรที่ชื่อ Brand (ซึ่งเป็นตัวแปรแบบ Sequence ชนิด List) จากกลุ่มของสินค้าที่เก็บไว้ในตัวแปร a โดยใช้การวนทำซ้ำจากคำสั่ง for ดึงสิ้นค้าจากตัวแปร a มาทีละ 1 รายการ เพื่อจะแจ้งลูกค้าถึงส่วนลดของราคาสินค้า 20% ของแต่ละรายการยกเว้นกระโปรงที่ไม่ส่วนลด (สังเกตเมื่อมีการพิมพ์ skirt' จะไม่มีการพิมพ์คำว่า "ลดราคา 20% วันนี้เท่านั้น" เหมือนเสื้อผ้าอื่นๆ เนื่องจากมีคำสั่ง continue ในบรรทัดที่ 9) และการอ่านรายการสินค้าจากคำสั่ง for จะสิ้นสุดก่อนความเป็นจริง เนื่องจากถ้าพบทีวียี่ห้อ hat (สมมุติว่าหมวกที่จะขายยังมาไม่ถึง) จากการตรวจเงื่อนไขในบรรทัดที่ 10 ก็จะทำการ break ด้วยคำสั่งจากบรรทัดที่ 12 เป็นผลให้การวนทำซ้ำของคำสั่ง for นี้หยุดลง และกระโดดไปทำคำสั่งในบรรทัดที่ 14,15 ต่อไป

ตัวอย่างที่ 7 การใช้คำสั่ง while ร่วมกับ break และ continue สมมุติข้อมูลรายชื่อลูกค้าที่มีรูปแบบการป้อนข้อมูลดังนี้

1406: มาณี:กทม

3767:ประเสริฐ:อยุธยา

7499:หน้าตาดี:ขอนแก่น

และต้องการผลลัพธ์เป็นรายงานที่มีรูปแบบต่อไปนี้

รหัส ชื่อ จังหวัด

1406 มาณี (กทม)

3767 ประเสริฐ (อยุธยา)

7499 หน้าตาดี (ขอนแก่น)

บรรทัดที่ 1 a = []

บรรทัดที่ 2 while True :

บรรทัดที่ 3 b = input('ร้านค้า \n เพิ่ม [a] \n แสดง [s] \n ออกจากระบบ [x]')

บรรทัดที่ 4 b = b.lower()

บรรทัดที่ 5 if b = = 'a' :

บรรทัดที่ 6 c = input('ป้อนรายการลูกค้า')

บรรทัดที่ 7 a.append(c)

บรรทัดที่ 8 print('***ข้อมูลได้เข้าสู่ระบบแล้ว***')

บรรทัดที่ 9 elif b = = 's' :

บรรทัดที่ 10 print('{0:-<6}{0:-<10}{0:-<10}'.format(''))

บรรทัดที่ 11 print('{0:<6}{1:<10}{2:10}'.format('รหัส','ชื่อ','จังหวัด'))

บรรทัดที่ 12 print('{0:-<6}{0:-<10}{0:-<10}'.format(""))

บรรทัดที่ 13 for d in a :

บรรทัดที่ 14 e = d.split(":")

บรรทัดที่ 15 print('{0[0]:<6} {0[1]:<10} ({0[2]:10})'.format(e))

บรรทัดที่ 16 continue

บรรทัดที่ 17 elif b = = 'x' :

บรรทัดที่ 18 break

บรรทัดที่ 19 print('ทำคำสั่งถัดไป.....')

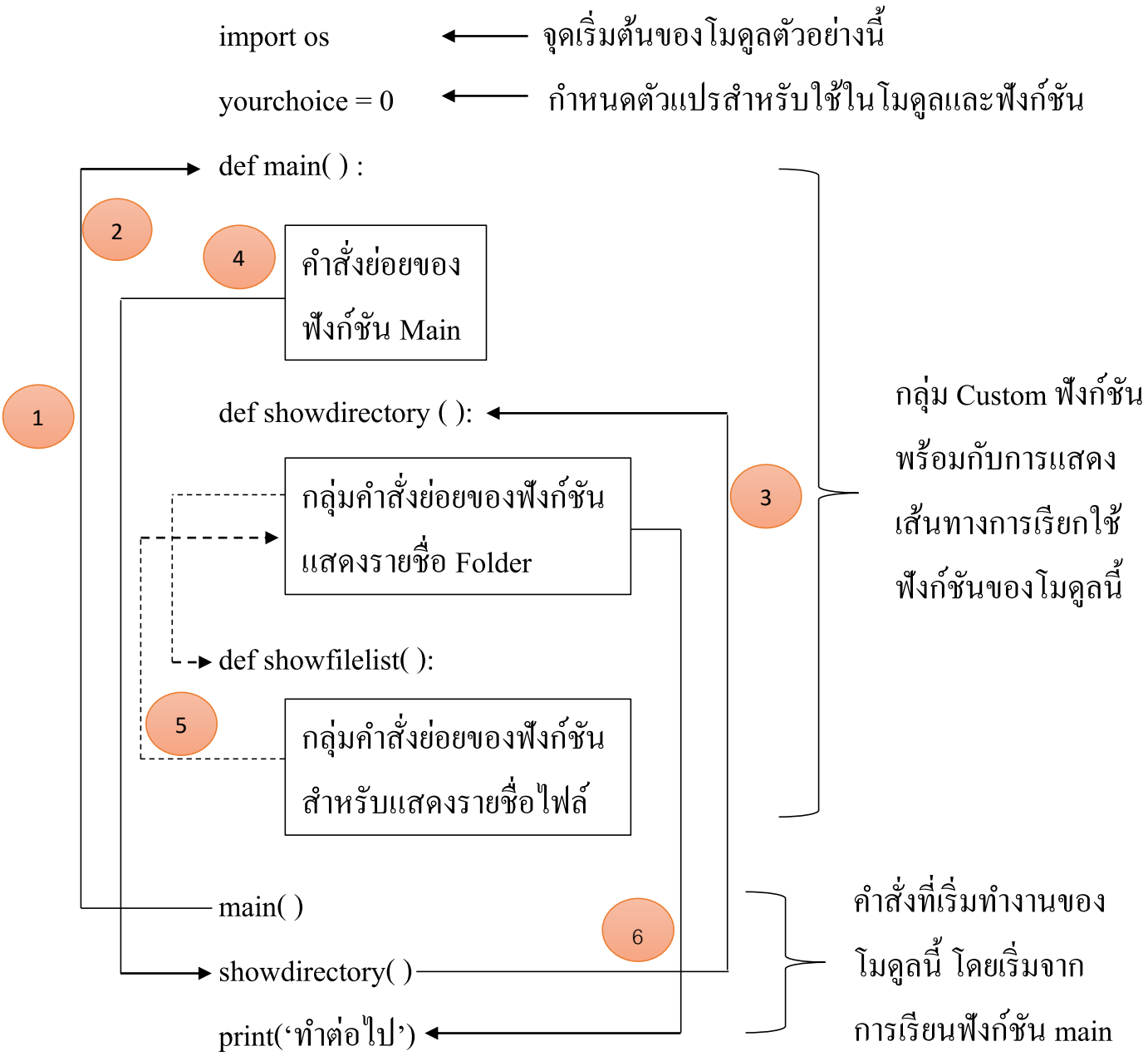
จากตัวอย่างที่ 7 เป็นโปรแกรมสมมุติของร้านค้าในส่วนของลูกค้า เมื่อโปรแกรมทำงานจะมีเมนูให้ผู้ใช้เลือกเมนูเกี่ยวกับการเพิ่มลูกค้า, การพิมพ์รายการของลูกค้า และการสิ้นสุดการใช้งานของโปรแกรม เป็นต้น โดยที่ลักษณะการทำงานของโปรแกรมจะใช้วิธีการวนทำซ้ำด้วยคำสั่ง while ที่ใช้เงื่อนไขแบบไม่มีวันสิ้นสุดคือ True แสดงว่ากลุ่มคำสั่งย่อยภายในคำสั่ง while จะต้องมีคำสั่งพิเศษที่ชื่อ break เพื่อหยุดการวนทำซ้ำเป็นอย่างแน่นอน (สังเกตบรรทัดที่ 17.18 เป็นการตรวจดูการเลือกเมนูเพื่อเลิกใช้งาน โดยการพิมพ์ตัวอักษร x เพื่อสั่งให้คำสั่ง break ทำงานนั่นเอง) สำหรับเทคนิคการสร้างเมนูให้ผู้ใช้เลือกก็อาศัยเทคนิคง่ายๆ จากบรรทัดที่ 2 คือการรอรับการป้องกันข้อมูลผ่านคีย์บอร์ดจากคำสั่ง input ที่สามารถแสดงข้อความได้ เราก็ดัดแปลงข้อความนั้นมาทำเป็นเมนูด้วยการเพิ่มเครื่องหมาย \n สำหรับขึ้นบรรทัดใหม่เท่านั้นก็เป็นเมนูได้แล้ว เมื่อผู้ใช้งานต้องการเพิ่มรายการลูกค้าให้พิมพ์ตัวอักษร a ซึ่งจะทำให้คำสั่งบรรทัดที่ 5,6 และ 7 ทำงาน หากต้องการดูรายการลูกค้าทั้งหมด ให้พิมพ์ตัวอักษร s จะมีการทำงานในบรรทัดที่ 9 ไปจนถึงบรรทัดที่ 16 สำหรับการพิมพ์รายการของลูกค้าแต่ละคน ตามรูปแบบรายงานที่กำหนดไว้ในโจทย์

บทที่ 6 การสร้างโพรซีเจอร์หรือฟังก์ชันของ Python

โพรซีเจอร์ หรือฟังก์ชัน มักจะเป็นชื่อที่นักพัฒนาโปรแกรมทั้งหลายต้องคุ้นหู และจำเป็นต้องรู้จักรูปแบบการเขียน ประโยชน์ของการใช้งาน เทคนิคการเขียนในลักษณะต่างๆ ตามความสามารถของแต่ละภาษา ทั้งโพรซีเจอร์และฟังก์ชันจะมีความคล้ายคลึงกันมาก จึงมักจะเกิดคำถามสำหรับนักพัฒนาโปรแกรมมือใหม่ ถึงการเลือกใช้โพรซีเจอร์หรือฟังก์ชันเสมอ กล่าวตามหลักการวิชาการมักจะให้คำจำกัดความเกี่ยวกับโพรซีเจอร์และฟังก์ชัน พอสรุปให้ง่ายต่อการทำความเข้าใจ ตามตารางต่อไปนี้

โพรซีเจอร์	ฟังก์ชัน
1. เป็นที่อยู่ของกลุ่มคำสั่งที่มีโอกาสเรียกใช้งานซ้ำๆได้ 2. สามารถกำหนดตัวแปรที่ใช้งานเฉพาะภาพในโพรซีเจอร์นี้ได้เท่านั้น 3. รองรับการผ่านค่าข้อมูลจากภายนอกในรูปแบบอาร์กิวเมนต์ 4. สามารถดึงตัวแปรภายนอกโพรซีเจอร์มาใช้ร่วมภายในโพรซีเจอร์ได้ 5. ไม่สามารถส่งค่าผลลัพธ์กลับในรูปแบบการ return ได้	ข้อ 1-4 มีคุณสมบัติเหมือนกับโพรซีเจอร์ ข้อ 5 สามารถ return ค่าผลลัพธ์ภายในของฟังก์ชันกลับไปยังจุดที่เรียกฟังก์ชันมาใช้งานได้

จากรายละเอียดในตาราง จะเห็นความแตกต่างกันเพียงข้อที่ 5 ความสามารถในการส่งค่าผลลัพธ์ภายในกลับออกไปด้วยการใช้คำสั่ง return ซึ่งจะมีใช้เฉพาะกับฟังก์ชันเท่านั้น ด้วยเหตุที่มีความแตกต่างจากลักษณะการใช้งานเพียงเล็กน้อย ภาษา Python จึงรวมโครงสร้างของทั้งโพรซีเจอร์และฟังก์ชันให้เป็นเรื่องเดียวกัน โดยจะอ้างอิงคุณลักษณะของการใช้งานแบบฟังก์ชันเป็นหลัก ดังนั้นในภาษา Python จะมีเพียงแค่การสร้างฟังก์ชัน (Custom Function) สำหรับใช้งานในโมดูลเท่านั้น ก่อนที่จะทำความเข้าใจรูปแบบการสร้างฟังก์ชันด้วยคำสั่งของภาษา Python ลองมาดูโครงสร้างโดยรวมและเส้นทางการทำงานของฟังก์ชันตามรูปที่ 1



รูปที่ 1 แสดงการสร้างฟังก์ชัน, วิธีการเรียกใช้งาน พร้อมเส้นทางการทำงานของแต่ละฟังก์ชันในโมดูลของ Python

ตำแหน่งของการสร้างฟังก์ชันสำหรับการใช้งายภายในโครงสร้างของโมดูล รวมทั้งลักษณะวิธีการเรียกใช้ฟังก์ชัน (สังเกตส้นทางเมื่อฟังก์ชันแต่ละชื่อถูกเรียกใช้งาน โดยเริ่มจากฟังก์ชันที่ชื่อ main เป็นจุดเริ่มต้นของโมดูลนี้) ยกตัวอย่างตามรูปที่ 1 สำหรับโมดูลนี้คำสั่งจะเริ่มต้นที่หลังสันประข้างล่างของกลุ่มฟังก์ชันทั้งหมด คำสั่งแรกหลังเส้นประจะพบการเรียนใช้งานฟังก์ชันที่ชื่อว่า main เมื่อฟังก์ชันนี้ทำงานจะ

กระโดดไปยังฟังก์ชัน main ตามเส้นทางหมายเลข 1 และเริ่มทำงานกับกลุ่มคำสั่งย่อยภายในฟังก์ชัน main จนสิ้นสุดจึงกระโดดกลับมาที่กลุ่มคำสั่งเริ่มต้นตามเส้นทางกลับหมายเลข 2 ที่คำสั่งถัดไป คือ คำสั่งที่เรียกฟังก์ชันที่ชื่อว่า showdirectory ซึ่งจะกระโดดไปที่ฟังก์ชันตามเส้นทางหมายเลข 3 เพื่อทำคำสั่งย่อยต่างๆ ภายในของฟังก์ชัน showdirectory ขณะที่ทำคำสั่งย่อยภายใน จะมีการเรียกฟังก์ชันที่ชื่อ showfilelist ซึ่งจะกระโดดไปยังที่อยู่ของฟังก์ชันดังกล่าวตามเส้นทางหมายเลข 4 เพื่อทำงานกับกลุ่มคำสั่งย่อยภายในของฟังก์ชัน showfilelist จนกระทั่งสิ้นสุดการทำงานของกลุ่มคำสั่งย่อยจึงกระโดดกลับตามเส้นทางหมายเลข 5 มายังกลุ่มคำสั่งย่อยภายในของฟังก์ชัน showdirectory และทำคำสั่งย่อยภายในของฟังก์ชันนี้ จนกระทั่งเสร็จสิ้นจึงจะกระโดดกลับไปกลุ่มคำสั่งหลักด้วยเส้นทางหมายเลข 6 เพื่อทำคำสั่งในกลุ่มคำสั่งหลักต่อไปจนจบโปรแกรม คงพอจะเห็นภาพการทำงานและวิธีเรียกใช้งานฟังก์ชันภายในโมดูลของภาษา Python และเพื่อให้ง่ายต่อการทำความเข้าใจในรายละเอียดของการเขียนโปรแกรมร่วมกับฟังก์ชันที่จัดสร้างขึ้น จะขอแบ่งการอธิบายออกเป็นหัวข้อดังต่อไปนี้

1. รูปแบบการสร้างฟังก์ชันของภาษา Python และวิธีเขียนโปรแกรมเพื่อเรียกใช้งาน
2. การกำหนดขอบเขตของตัวแปรภายในของฟังก์ชัน
3. ลักษณะการใช้คีย์เวิร์ดอาร์กิวเมนต์และพารามิเตอร์ร่วมกับฟังก์ชัน (Argument And Parameter Function Keyword)
4. ความแตกต่างระหว่างฟังก์ชันที่ใช้และไม่ใช้คำสั่ง return

<u>รูปแบบการสร้างฟังก์ชันของภาษา Python และวิธีเขียนโปรแกรมเพื่อเรียกใช้งาน</u>

ลักษณะการสร้างฟังก์ชันมีรูปแบบ (Syntax) ของการเขียนที่ค่อนข้างง่าย ดังรูปที่ 2 แสดงวิธีการเขียนฟังก์ชันที่แบ่งตามการนำไปใช้งาน 4 รูปแบบ ได้แก่

1. การสร้างฟังก์ชันที่ทำงานเหมือนโพรซีเจอร์ (ไม่มีการส่งค่าใดๆ กลับ)
2. การสร้างฟังก์ชันที่ทำงานเหมือนโพรซีเจอร์ แต่มีการส่งผ่านค่าต่างๆ ร่วมกับตัวแปรอาร์กิวเมนต์
3. การสร้างฟังก์ชันร่วมกับการส่งค่าผลลัพธ์กลับด้วยคำสั่ง return
4. การสร้างฟังก์ชันที่มีการใช้ตัวแปรอาร์กิวเมนต์และการส่งค่าผลลัพธ์กลับ

1.	def ชื่อฟังก์ชัน( ) : a = 100 print('a=%d' % a) print(a+10)	เหมาะต่อการใช้งานกับกลุ่มคำสั่งที่ทีการเรียกใช้ซ้ำๆหรือถูกเรียกใช้งานจากหลายๆ ที่โปรแกรมของโมดูล
2.	def ชื่อฟังก์ชัน(ตัวแปรอาร์กิวเมนต์): a = ค่าอาร์กิวเมนต์ print(a+10)	รูปแบบในหัวข้อนี้จะถูกเรียดใช้งานเหมือนหัวข้อที่ 1 แต่สามารถส่งผ่านค่าจากการเรียกใช้งานฟังก์ชันด้วยตัวแปรอาร์กิวเมนต์ที่กำหนดร่วมกับฟังก์ชัน
3.	def ชื่อฟังก์ชัน( ) : a = 1 b = 2 print(a + b) return a + b	การสร้างฟังก์ชันลักษณะนี้เหมาะต่อการประมวลผลที่เกิดจากกลุ่มคำสั่งย่อยภายใน ซึ่งผลลัพธ์ที่ได้สามารถส่งย้อนกลับ โดยการใช้คำสั่ง return ของฟังก์ชัน
4.	def ชื่อฟังก์ชัน(ตัวแปรอาร์กิวเมนต์): a = ค่าอาร์กิวเมนต์ b = 5 print(a * b) return a * b	การทำงานของฟังก์ชันในหัวข้อนี้มีหลักการทำงานเหมือนหัวข้อที่ 3 โดยแตกต่างกันตรงที่ฟังก์ชันสามารถผ่านค่าข้อมูลจากภายนอกด้วยตัวแปรอาร์กิวเมนต์มาใช้งานภายในฟังก์ชันได้

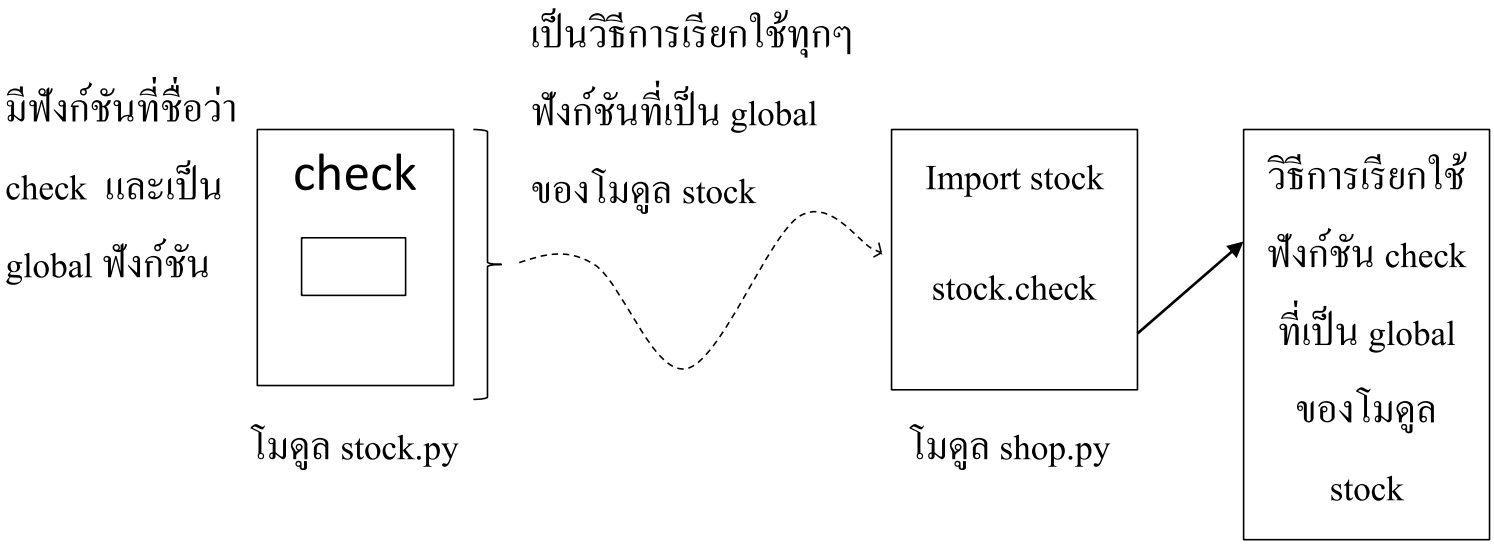
รูปที่ 2 แสดงรูปแบบวิธีการสร้างฟังก์ชันของภาษา Python

ซึ่งทั้ง 4 รูปแบบจะถูกนำมาใช้งานภายในโมดูลของภาษา Python ตามคุณสมบัติของแต่ละรูปแบบ โดยนักพัฒนาโปรแกรมจะเป็นผู้ที่ตัดสินใจเลือกใช้งานให้ตรงกับรูปแบบโปรแกรมคำสั่งที่กำลังพัฒนา และตรงตามการออกแบบโครงสร้าง เพื่อพัฒนาโปรแกรม เช่น โปรแกรมที่เป็นลักษณะ Procedural ซึ่งก็เป็นรูปแบบการเขียนโปรแกรมที่เป็นพื้นฐานของภาษา Python ที่มีการใช้ฟังก์ชันทั้ง 4 รูปแบบ เป็นส่วนร่วมภายในโมดูล และแบ่งลักษณะการใช้งานฟังก์ชันทั้ง 4 รูปแบบ ออกเป็น 2 ลักษณะ คือ

1. ฟังก์ชันที่ใช้งานแบบ Global
2. ฟังก์ชันที่ใช้งานแบบ Local

ฟังก์ชันที่ใช้แบบ

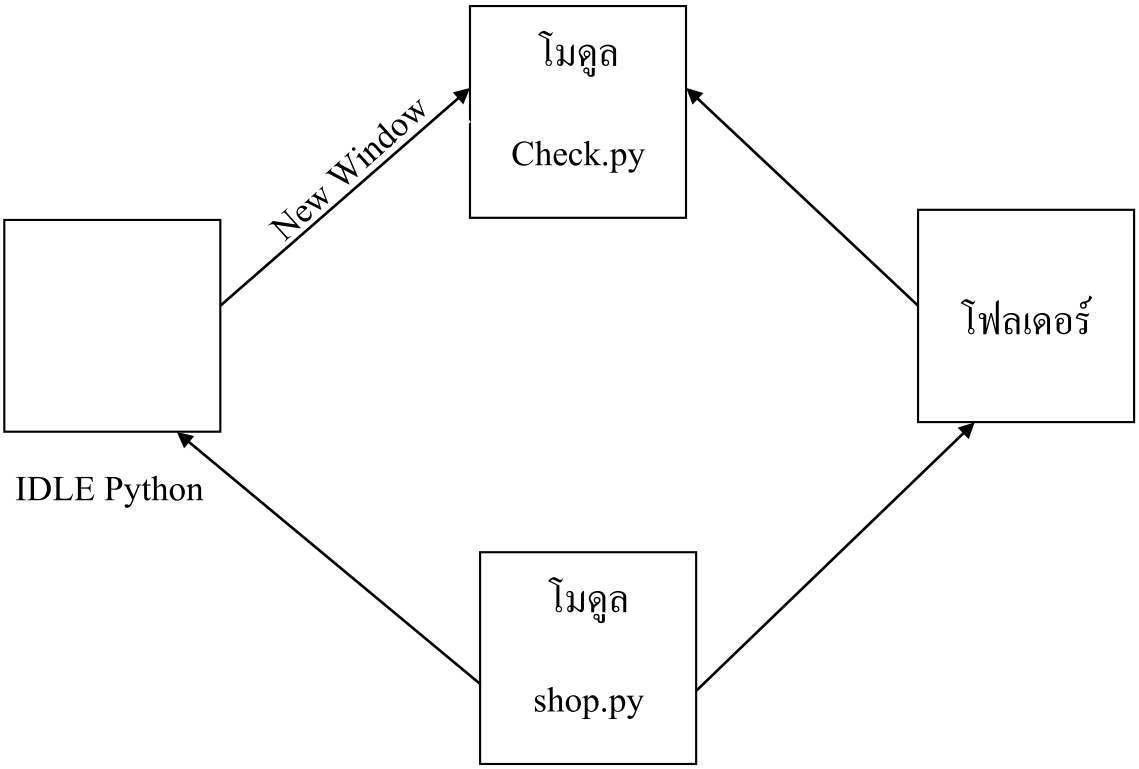
ฟังก์ชันประเภทนี้ถูกสร้างขึ้นภายในโมดูลมาตรฐานปกติของ Python ซึ่งสามารถนำฟังก์ชันนี้ไปใช้งานร่วมกับโมดูลอื่นๆ โดยเชื่อมโยงการใช้งานด้วยคำสั่ง import จากรูปที่ 3 แสดงวิธีการ import การใช้ Global ฟังก์ชันระหว่างโมดูลกับโมดูล



รูปที่ 3 แสดงลักษณะการสร้าง Global ฟังก์ชันและการเรียกใช้งานฟังก์ชันระหว่างโมดูล

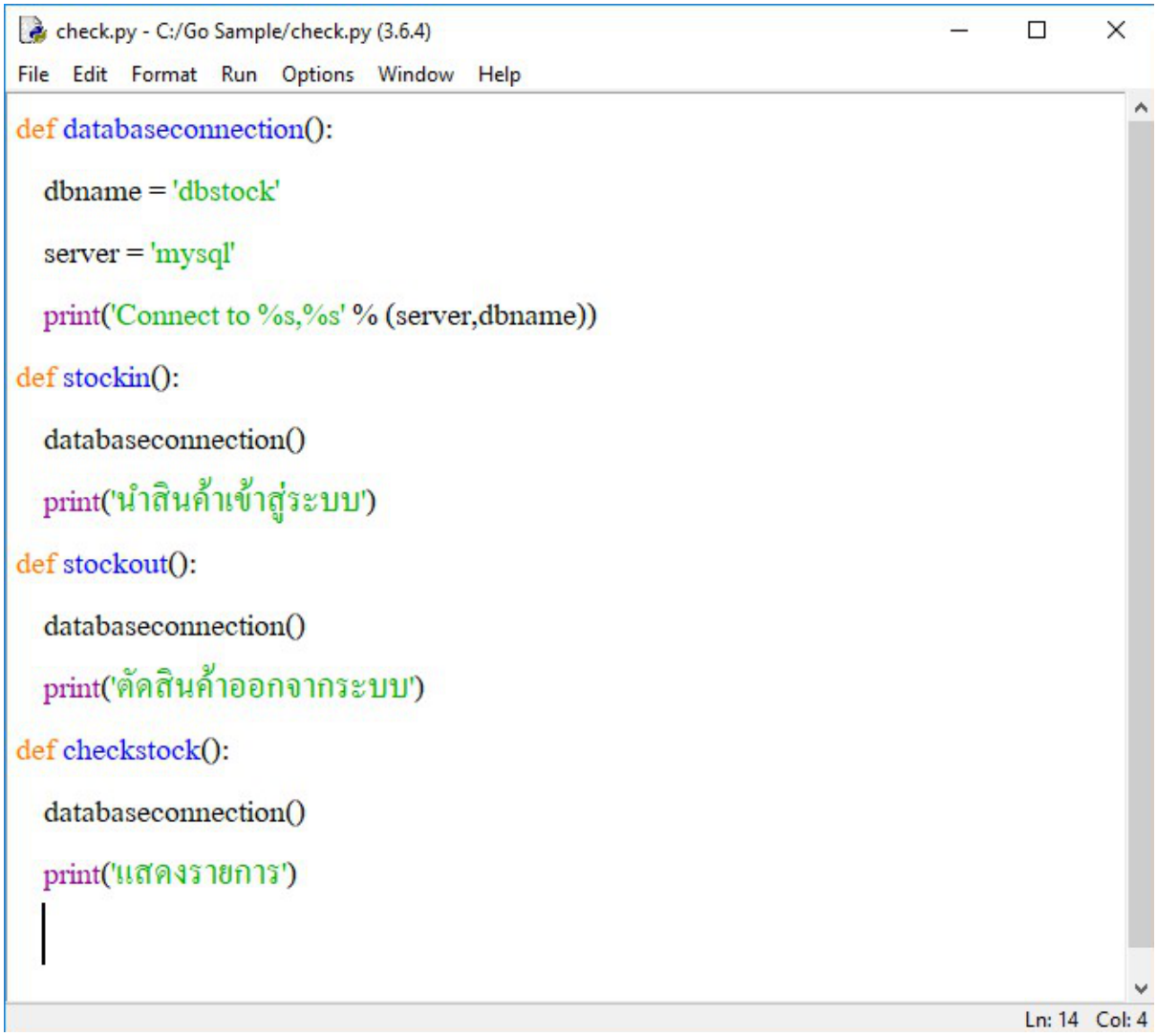
เงื่อนไขเล็กๆ น้อยๆ ที่ผู้ใช้งานไม่ควรพลาดในการสร้างโมดูลและฟังก์ชันลักษณะ Global สำหรับใช้งานร่วมกับโมดูลอื่นๆ คือการจดบันทึกไฟล์โมดูลทั้งหมดไว้ในโฟลเดอร์เดียวกัน มาลองทำตัวอย่างเกี่ยวกับฟังก์ชันประเภท Global

ตัวอย่างที่ 1 สร้างโฟลเดอร์ใน Local Drive ที่ชื่อว่า Go Sample (ถ้าใช้ระบบปฏิบัติการวินโดวส์ให้ทดลองสร้างไว้ใน Drive C:\Go Sample) จากนั้นเปิดโปรแกรม IDLE Python เลือกเมนู File, เมนูย่อยNew Window เมื่อจอภาพโมดูลของ Python ปรากฏให้บันทึกไฟล์ลงในโฟลเดอร์ Go Sample โดยใช้ชื่อว่า check.py กลับไปที่ IDLE Python เปิด New Window อีกครั้งหนึ่ง ทำการบันทึกจอภาพโมดูลที่ 2 ลงในไฟล์ที่ชื่อว่า shop.py ลงในโฟลเดอร์เดียวกับไฟล์ check.py



รูปที่ 4 ลำดับการสร้างโมดูลไปเก็บไว้ใน C:\Go Sample

ทดลองสร้างฟังก์ชัน Global ลงในโมดูล check.py ดังนี้



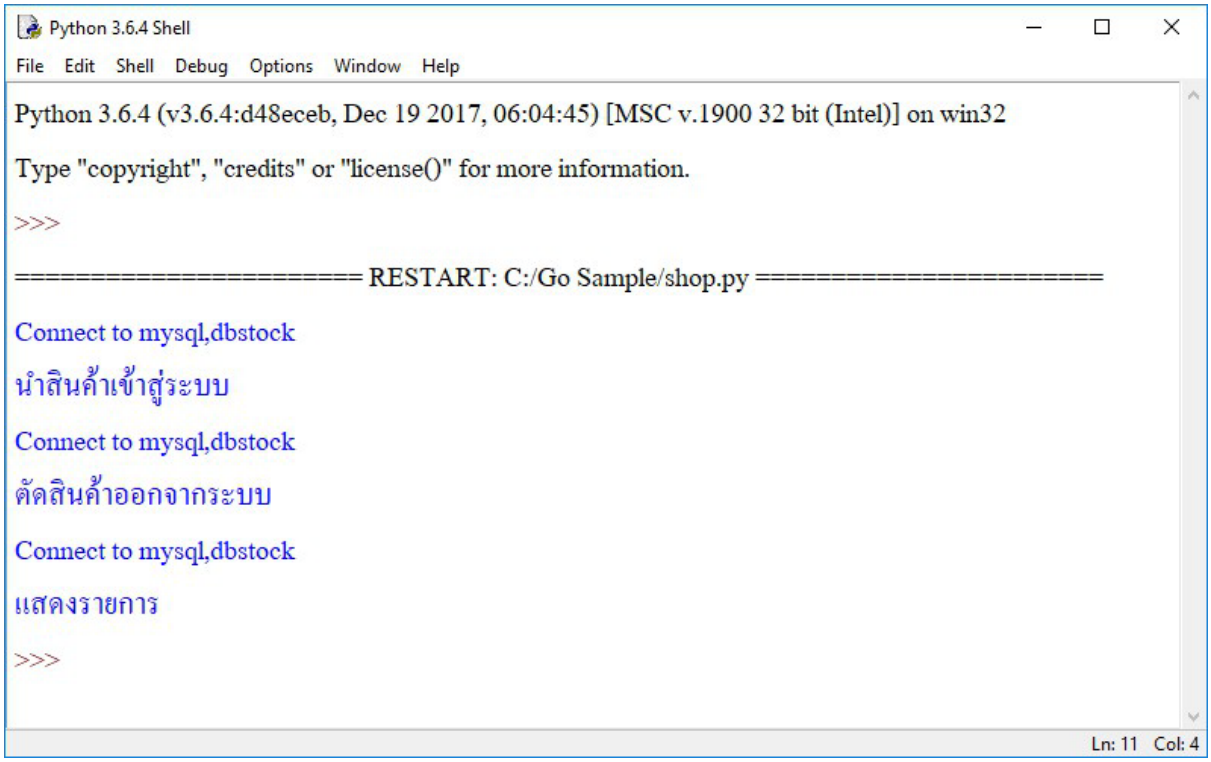
จากนั้นทำการบันทึกไฟล์โมดูล check.py เมื่อเสร็จสิ้นแล้วให้ทำการทดลองเปิดโมดูล shop.py เพื่อเรียกใช้ฟังก์ชัน Global จากโมดูล check.py ตามคำสั่งต่อไปนี้

Import check ⟵ สำหรับอ้างอิงโมดูล check.py

check.stockin( ) ⟵ เรียกใช้ฟังก์ชัน global ที่ชื่อ stockin จากโมดูล check.py

check.stockout( )

check.checkstock( )



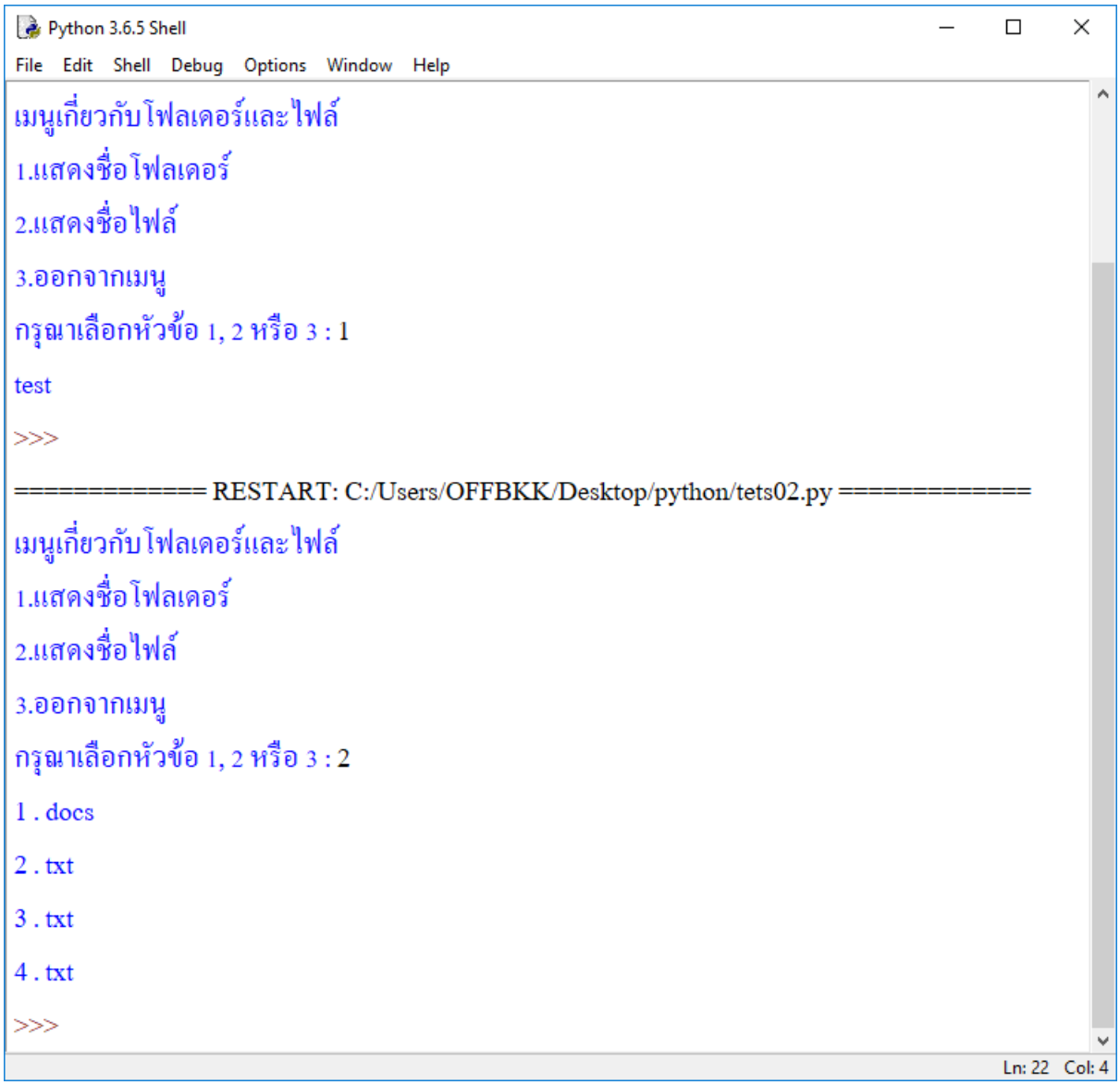
ทำการทดสอบโปรแกรมโดยการรันคำสั่งในโมดูล shop เพื่อดูผลลัพธ์การเรียกใช้งานฟังก์ชัน Global จากโมดูล check.py ข้อควรสังเกตการณ์เรียกใช้ฟังก์ชันแบบ Global เนื่องจากชื่อฟังก์ชันแบบ Global มีโอกาสซ้ำกับชื่อฟังก์ชันภายในโมดูล shop ซึ่งเป็นฟังก์ชันแบบ Local ได้ รูปแบบการเขียนโปรแกรมเรียกใช้ฟังก์ชัน Global จำเป็นต้องมีชื่อโมดูล (บางครั้งเรียก Name Space) ของฟังก์ชัน Global พิมพ์ไว้ข้างหน้าการเรียกใช้ฟังก์ชันเสมอ

ฟังก์ชันที่ใช้งานแบบ Local

การสร้างฟังก์ชันใดๆ ภายในโมดูลมาตรฐานของ Python และมีการเรียกใช้งานฟังก์ชันเฉพาะภายในโมดูลนั้น กลุ่มฟังก์ชันภายในโมดูลนั้นจะถูกจัดให้เป็นฟังก์ชันภายในหรือแบบ Local นั่นเอง ทดลองสร้างและใช้งานฟังก์ชันแบบ Local ตามตัวอย่างดังนี้

ตัวอย่างที่ 1 ทำการเปิดโมดูลวินโดว์จาก IDLE Python

```
import os # อ้างอิงชุดคำสั่งเกี่ยวกับไฟล์และโฟลเดอร์ของระบบปฏิบัติการ
yourchoice = ''

def main():
    global yourchoice #ประกาศว่าจะใช้ตัวแปรของโมดูล
    while True:
        print('เมนูเกี่ยวกับโฟลเดอร์และไฟล์')
        print('1.แสดงชื่อโฟลเดอร์')
        print('2.แสดงชื่อไฟล์')
        print('3.ออกจากเมนู')
        yourchoice=input('กรุณาเลือกหัวข้อ 1, 2 หรือ 3 : ')
        if yourchoice!='':
            break

def showdirectory():
    for sNameDir in os.listdir('c:\\python31'):
        if not '.'in sNameDir:
            print(sNameDir)

def showfilelist():
    for sNameFile in os.listdir('c:\\python31'):
        if '.' in sNameFile:
            filename,extname=sNameFile.split('.')
            print(filename,'.',extname)

main() #โปรแกรมเริ่มต้นจากการเรียกฟังก์ชัน Main
if yourchoice=='1':    #ตรวจหัวข้อที่ต้องการ
    showdirectory()    #แสดงรายชื่อโฟลเดอร์ใน Drive C:
elif yourchoice=='2':
    showfilelist()     #แสดงรายชื่อไฟล์ใน Drive C:
```



จากตัวอย่างนี้ จะต้องทดลองสร้างโฟลเดอร์ python31 ไว้ใน Drive C:\ โดยสร้างโฟลเดอร์และไฟล์นามสกุลต่างๆ ไว้ในโฟลเดอร์ python31 ก่อน เพื่อให้โปรแกรมที่เขียนขึ้นสามารถเรียกดูไฟล์และโฟลเดอร์ได้ โดยคำสั่งที่เขียนขึ้นจะประกอบด้วยฟังก์ชันแบบ Local ในโมดูลนี้มีทั้งหมด 3 ฟังก์ชัน main, showdirectory, showfilelist โดยฟังก์ชันแรกที่ถูกเรียกใช้งานคือฟังก์ชัน main ทำหน้าที่เป็นเมนูสำหรับเรียกใช้ฟังก์ชัน showdirectory และ showfilelist ผ่านตัวแปรที่ชื่อ yourchoice ที่เก็บค่าหมายเลขหัวเรื่องจากภายในฟังก์ชัน main สำหรับฟังก์ชันแสดงรายชื่อโฟลเดอร์ใน Drive C คือฟังก์ชันที่ชื่อ showdirectory ประกอบด้วยคำสั่ง os.listdir( ) ร่วมกับคำสั่ง for เพื่ออ่านรายชื่อโฟลเดอร์และไฟล์ตามที่ระบุชื่อ Path หรือ Drive ในตัวอย่างนี้ระบุให้อ่านใน Drive C:\ ทั้งหมด เนื่องจากคำสั่ง os.listdir( ) จะอ่านรายชื่อโฟลเดอร์และไฟล์ จึงจำเป็นต้องเลือกเฉพาะชื่อโฟลเดอร์เท่านั้น วิธีการก็ใช้คำสั่ง if เพื่อสร้างเงื่อนไขในการแยกชื่อที่เป็น

โฟลเดอร์ (สังเกตไม่มีการใช้เครื่องหมาย. และนามสกุล) ออกจากชื่อไฟล์โดยการค้นหาเครื่องหมาย. ภายในตัวแปร sNamedir ที่มีรูปแบบคำสั่งในการค้นหาเครื่องหมายดังนี้ '.'in sNamedir ซึ่งผลลัพธ์ที่ได้จากการค้นหาเครื่องหมายจุด เมื่อตรวจพบจะมีค่าเป็น True เมื่อนำคำสั่งการค้นหามารวมกับคำสั่ง if (if '.' In sNamedir) จะเป็นการสร้างเงื่อนไขเพื่อตรวจชื่อในค่าตัวแปร sNamedir ที่เป็นเกี่ยวกับไฟล์เท่านั้น ถ้าหากต้องการตรวจชื่อที่อยู่ในตัวแปร sNamedir ที่เป็นโฟลเดอร์ ต้องดัดแปลงคำสั่งการตรวจเงื่อนไขตามนี้ if not '.' in sNamedir จะเห็นว่ามีการเพิ่มคำสั่ง not แทรกลงในการตรวจเงื่อนไข แค่นี้ก็จะค้นหาเฉพาะชื่อที่เป็นโฟลเดอร์เท่านั้น

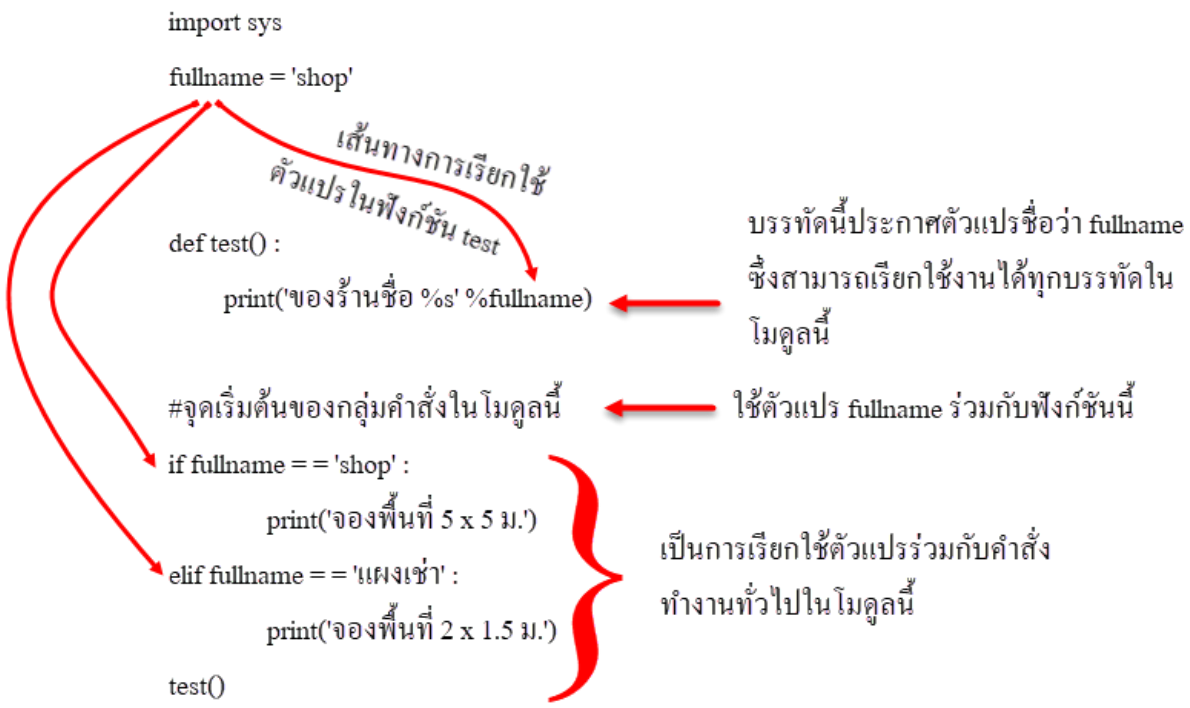
นอกจากการใช้งานฟังก์ชันที่เป็น Global และ Local แล้วก็มีการสร้างหรือเขียนฟังก์ชันเพื่อใช้งานภายใต้รูปแบบการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) ตอนนี้หากเปิดโมดูลมาตรฐานปกติทั่วไปมาเขียนโปรแกรม OOP แล้วเราจะเรียกโมดูลเป็น Class Module แทนคำว่าโมดูลเฉยๆ และแน่นอน คำว่าฟังก์ชันที่เราสร้างในโมดูลเฉยๆ ก็ต้องเปลี่ยนชื่อเรียกฟังก์ชันใหม่เป็นคำว่า Method ของ Class Module โดยแบ่งลักษณะการใช้งาน Methon ออกเป็น 2 ลักษณะการใช้งาน คือ

1. Internal Method หรือ Private Method
2. Interface Method

<u>การกำหนดขอบเขตของตัวแปรภายในของฟังก์ชัน</u>

การเขียนโปรแกรมในโมดูล สิ่งหนึ่งที่ขาดไม่ได้เลยก็คือ ตัวแปร เมื่อมีการใช้งานตัวแปรในโมดูลก็ย่อมมีโอกาสที่ต้องนำตัวแปรเหล่านั้นมาใช้งานร่วมกับกลุ่มฟังก์ชันที่สร้างไว้ภายในโมดูลเช่นกัน ลักษณะของการประกาศตัวแปรในโมดูล เพื่อนำมาใช้ในฟังก์ชันมีรูปแบบตามตัวอย่างต่อไปนี้

ตัวอย่างที่ 1 การใช้ตัวแปรระดับโมดูลร่วมกับคำสั่งย่อยภายในฟังก์ชัน



จากตัวอย่างที่ 1 เป็นการแสดงตัวแปรที่ประกาศและใช้งานในโมดูล สังเกตในฟังก์ชัน test จะมีการเรียกใช้งานตัวแปร fullname ร่วมกับกลุ่มคำสั่งย่อยภายในฟังก์ชัน เมื่อทำการทดสอบรันโปรแกรมตัวแปร fullmane ที่ประการไว้และต้องนำมาใช้ร่วมภายในฟังก์ชัน test ก็สามารถแสดงค่าข้อมูลที่เก็บไว้ในตัวแปรได้อย่างถูกต้อง ซึ่งผลลัพธ์ที่ได้จะแสดงดังต่อไปนี้

ผลลัพธ์ของการทดสอบโปรแกรมในตัวอย่างที่ 1

>>> จองพื้นที่ 3 X 3 ม.

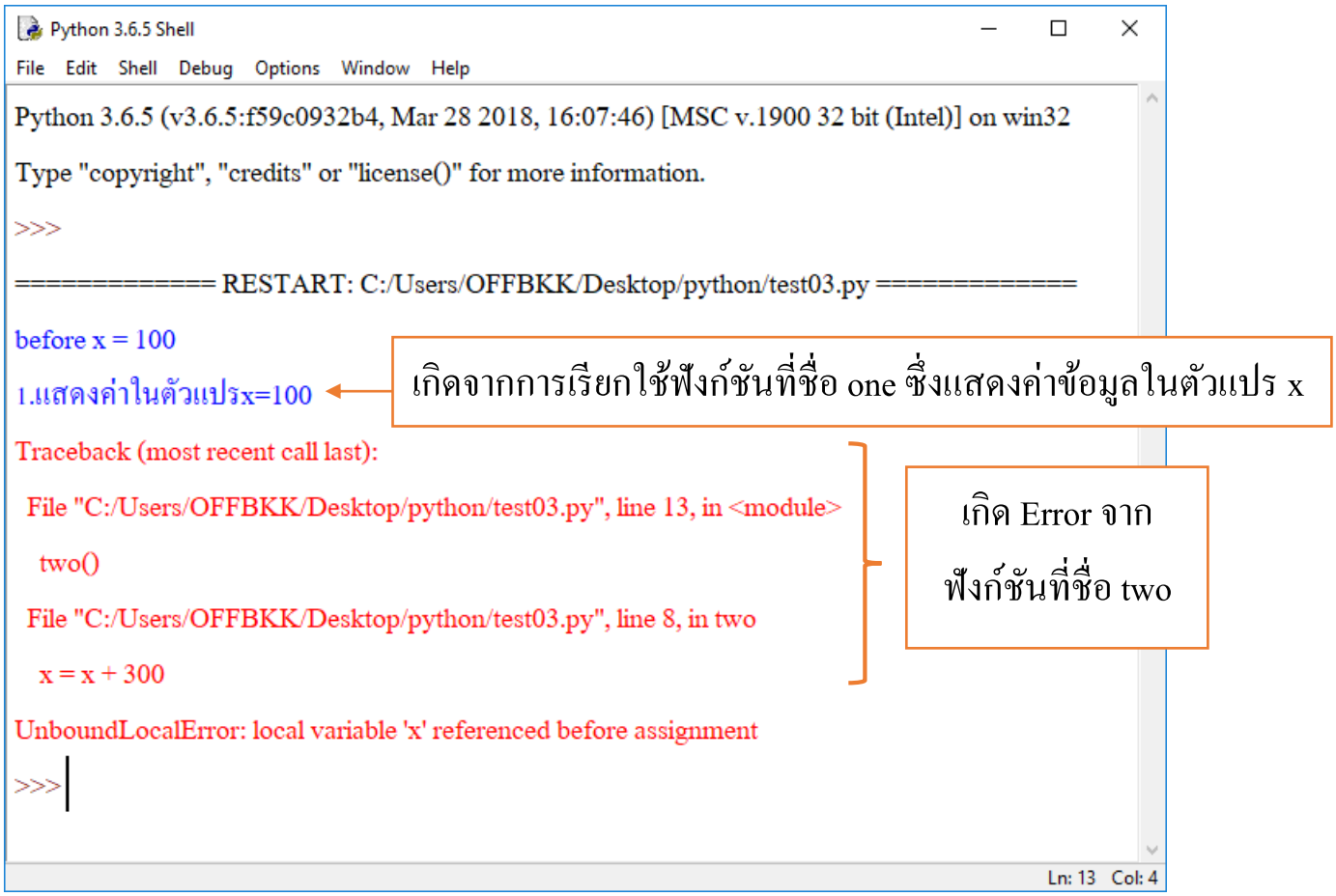
>>> ของร้านชื่อ shop

ตัวอย่างที่ 2 ทำไมถึงต้องใช้คำสั่ง Global กับตัวแปรของโมดูลภายในฟังก์ชัน

```
import sys
x=100

def one():
   print('1.แสดงค่าในตัวแปรx=%d'%x)

def two():
   x = x + 300
   print('2. แสดงค่าตัวแปร x = %d โดยการบวกเพิ่มอีก 300' %x)

print('before x = %d' %x)
one()
two()
print('after x = %d'%x)
```

โปรแกรมเริ่มต้นทำงานที่จุดนี้โดยเริ่มจากฟังก์ชัน one และ two

เมื่อรันโปรแกรมเพื่อทดสอบจะได้ผลลัพธ์ดังนี้



น่าแปลกมากที่เกิด Error เกิดขึ้นเมื่อเรียกใช้ฟังก์ชันที่ชื่อว่า two ในบรรทัด (x=x+300) ที่ตัวแปรของโมดูลมีการเปลี่ยนแปลงค่าใหม่ภายในฟังก์ชันนี้ ข้อความ Error ที่ปรากฏจะถามถึงตัวแปร x ที่กำลังจะเปลี่ยนแปลงค่าใหม่เป็นตัวแปร x แบบ Local ภายในฟังก์ชันหรือตัวแปร x นี้ได้มาจากข้างนอกฟังก์ชัน (ตัวแปรของโมดูล) ตรงนี้ผู้พัฒนาโปรแกรมต้องเป็นผู้เลือกการใช้ตัวแปร x ตัวนี้เอง ถ้าใช้ตัวแปร x แบบ Local ก็แก้ไขคำสั่งเพิ่มตามตัวอย่างต่อไปนี้ลงในฟังก์ชัน two และทำการทดลองรันโปรแกรมใหม่

```
def two():
    x = 0 เพิ่มการประกาศตัวแปร x
    x = x + 300
    print('2. แสดงค่าตัวแปร x = %d โดยการบวกเพิ่มอีก 300' %x)
```

ในบรรทัด x=0 ที่ทำการเพิ่มในฟังก์ชัน two จะถือว่าเป็นตัวแปรใหม่ที่ชื่อ x และจะใช้งานเฉพาะในฟังก์ชัน two เท่านั้น สำหรับตัวแปร x ของโมดูลจะไม่สามารถนำมาใช้งานในฟังก์ชันได้อีกเลย

ต่อมาในกรณีที่ต้องการใช้ตัวแปร x ของโมดูล ซึ่งประกาศอยู่ภายนอกฟังก์ชัน มีเงื่อนไขคือ ต้อง

เปลี่ยนแปลงค่าภายในของตัวแปร x ของโมดูลนี้ได้ตลอดเวลา ตัวอย่างการเพิ่มคำสั่ง global เสริมลงในฟังก์ชัน two แสดงไว้ดังนี้

```
def two():
    global x
    x = x+300
    print('2. แสดงค่าตัวแปร x = %d โดยการบวกเพิ่มอีก 300' %x)
```

เพิ่มคำสั่ง global ตัวแปรก่อนเปลี่ยนแปลงค่าในตัวแปร x

เมื่อกำหนด global x ในฟังก์ชัน two ต่อไปนี้ การใช้งานตัวแปร x ตัวนี้จะสามารถเปลี่ยนแปลงค่าภายในตัวแปรได้ (ตัวแปร x ต้องประกาศในโมดูลก่อนเรียกใช้ในฟังก์ชันเสมอ)

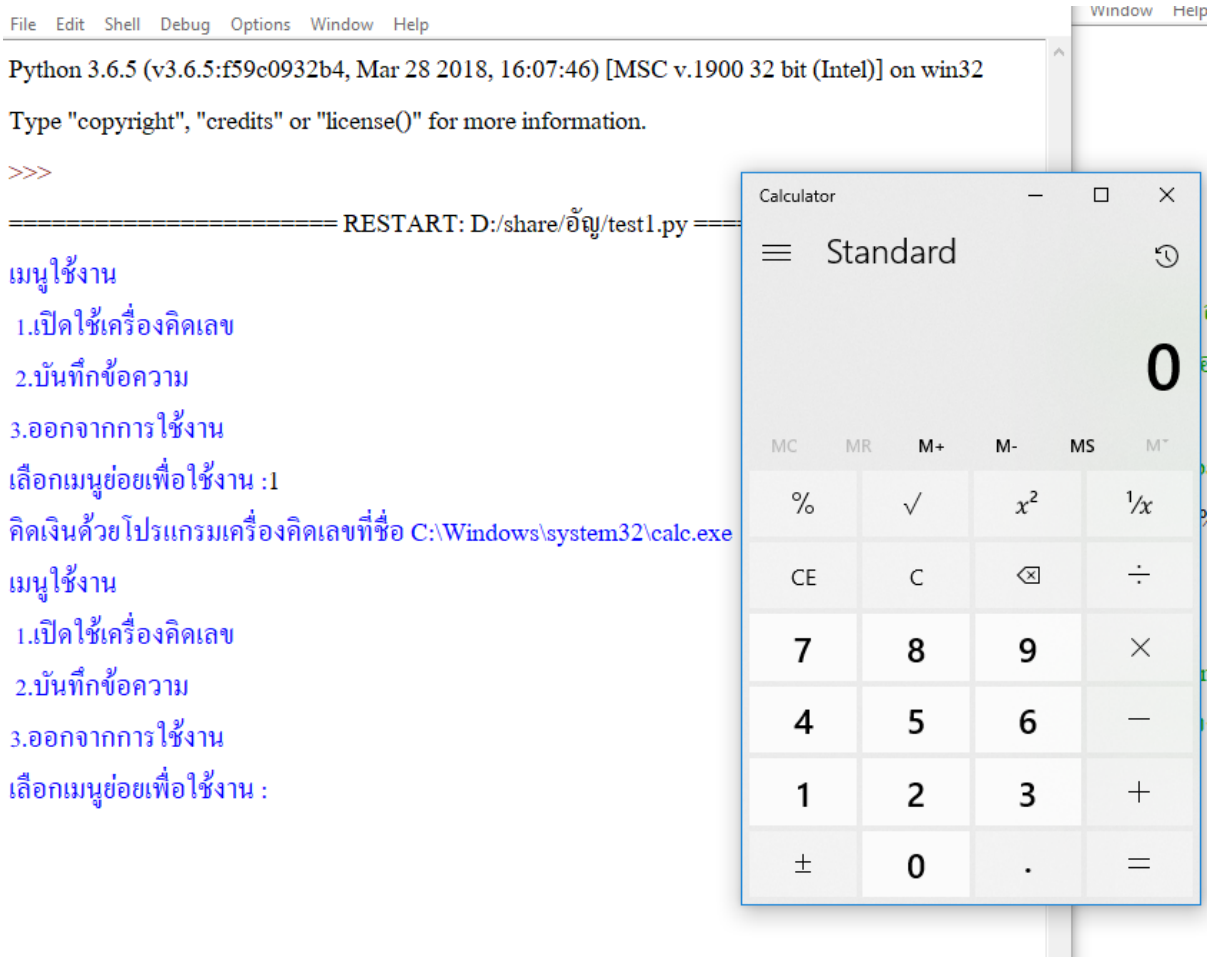
ตัวอย่างที่ 3 แสดงการกำหนดขอบเขตของตัวแปรที่ใช้งานกับฟังก์ชัน ให้เปิดวินโดว์โมดูลใหม่จาก IDLE ของ Python Shell เพื่อทดลองโปรแกรมต่อไปนี้

```
import os          #เชื่อมต่อโมดูลคำสั่งเกี่ยวกับการเรียกใช้ไฟล์ต่างๆ ใน Drive C:\
choice = 0         #ประกาศตัวแปรโมดูลเพื่อใช้งานร่วมกันในโมดูลนี้
filename = ‘’

def menu() :
        global choice        #อ้างถึงตัวแปรโมดูลมาใช้ภายในฟังก์ชันและสามารถเปลี่ยนแปลงค่าข้อมูลของตัวแปรนี้ได้ด้วย
        print(‘ เมนูใช้งาน \n 1.เปิดใช้เครื่องคิดเลข \n 2.บันทึกข้อความ \n3.ออกจากการใช้งาน’)
    choice = input('เลือกเมนูย่อยเพื่อใช้งาน :')

def opennotpad():
        filename = 'C:\\Windows\\notepad.exe'          #เป็นตัวแปร Local ของฟังก์ชันเท่านั้น
        print('บันทึกหมายเหตุด้วย %s' %filename)
        os.system(filename)

def opencalculator():
        filename = 'C:\\Windows\\system32\\calc.exe'
```

```
        print('คิดเงินด้วยโปรแกรมเครื่องคิดเลขที่ชื่อ %s'  %filename)

        os.system(filename)

#จุดเริ่มต้นของโปรแกรม
while True :
        menu()

        if choice == '1':
                opencalculator()
        elif choice == '2' :
                opennotepad()
        else :
                break
```

File Edit Format Run Options Window Help

```
import os
choice = 0
filename = ""
def menu() :
        global choice
        print('เมนูใช้งาน \n 1.เปิดใช้เครื่องคิดเลข \n 2.บันทึกข้อความ \n3.ออกจากการใช้งาน')
        choice = input('เลือกเมนูย่อยเพื่อใช้งาน :')
def opennotpad():
        filename = 'C:\\Windows\\notepad.exe'
        print('บันทึกหมายเหตุด้วย %s' %filename)
        os.system(filename)
def opencalculator():
        filename = 'C:\\Windows\\system32\\calc.exe'
        print('คิดเงินด้วยโปรแกรมเครื่องคิดเลขที่ชื่อ %s'  %filename)
        os.system(filename)
#จุดเริ่มต้นของโปรแกรม
while True :
        menu()
        if choice == '1':
                opencalculator()
        elif choice == '2' :
                opennotepad()
        else :
                break
```



จากตัวอย่างที่ 3 จะใช้ตัวแปรที่ชื่อ choice เป็นตัวแปรเก็บหัวข้อที่ต้องการใช้งาน (ตัวแปร choice จะถูกใช้งานในหลายแห่ง เช่น ในฟังก์ชันที่ชื่อ menu และที่กลุ่มคำสั่งเริ่มต้นที่คำสั่ง if กรณีนี้ตัวแปร choice ควรประกาศในระดับโมดูลจึงจะเหมาะสมต่อการใช้งาน) จากฟังก์ชันที่ชื่อ menu ซึ่งจะทำหน้าที่แสดงรายการของเมนูการใช้งานและรอรับหัวเรื่องที่ต้องการใช้งานจากคำสั่ง input ภายในฟังก์ชัน เนื่องจากจะมีการเปลี่ยนแปลงค่าข้อมูลในตัวแปร choice ที่อยู่ภายนอกฟังก์ชัน จึงจำเป็นต้องระบุคำว่า Clobal choice ลงในฟังก์ชัน menu เมื่อได้หมายเลขหัวเรื่องก็จะนำไปตรวจด้วยกลุ่มคำสั่ง if เพื่อเรียกใช้ฟังก์ชันย่อยอื่นๆ เช่น กดหมายเลข 1 จะเรียกใช้ฟังก์ชัน opencalculator ซึ่งภายในฟังก์ชันนี้จะมีตัวแปรภายในที่เป็นแบบ Local (ตัวแปรชื่อ filename) เพื่อกำหนดชื่อไฟล์เครื่องคิดเลขหรือ call.exe (ไฟล์เครื่องคิดเลขที่เรียกใช้งานจากตัวอย่างนี้จะได้รับการติดตั้ง และมีใช้งานในระบบปฏิบัติการ Windows อยู่แล้ว) โดยจะนำค่าในตัวแปร filename ไปใช้ร่วมกับคำสั่ง os.system(filename) ซึ่งทำหน้าที่รันไฟล์ที่มีนามสกุลเป็น .exe ของระบบปฏิบัติการ Windows กดหมายเลข 2 สำหรับรียกฟังก์ชัน opennotepad (ลักษณะการใช้คำสั่งและตัวแปรจะคล้ายกับฟังก์ชัน opencalculator) สำหรับเปิดโปรแกรม Notepad ของระบบปฏิบัติการสรุปได้ว่าการทำงานของโปรแกรมในตัวอย่างที่ 3 ตัวแปร choice ค่อยข้างมีบทบาทในการใช้ค่าที่ได้รับมาจากภายใน

ฟังก์ชัน menu เชื่อมโยงกับกลุ่มคำสั่งในจุดอื่นๆ ภายในโมดูล ดังนั้นฟังก์ชันใดต้องการใช้ตัวแปร choice ที่ต้องการเปลี่ยนแปลงค่าข้อมูลก็อย่าลืมใช้ Global choice ไว้ในฟังก์ชันด้วย

ลักษณะการใช้คีย์เวิร์ดอาร์กิวเมนต์และพารามิเตอร์ร่วมกับฟังก์ชัน

(Argument And Parameter Function Keyword)

" เพิ่มอาร์กิวเมนต์สัก 2 ตัวในฟังชัน Payment ด้วย " ถ้าใครเป็นนักพัฒนาโปรแกรมมือใหม่หรือมือเก่าก็คงไม่มีปัญหาในการสร้างอาร์กิวเมนต์เพิ่มให้กับฟังก์ชันมากนัก เพียงแต่ขอตัวอย่างรูปแบบ (Syntax) ของการเขียนใน python ว่าเค้ามีรูปแบบอย่างไร เท่านี้ก็ลุยทำงานที่เหลือต่อไปได้ ก่อนที่จะมาดูวิธีการใช้อาร์กิวเมนต์หรือบางครั้งก็เรียกว่าพารามิเตอร์เพิ่มให้กับฟังก์ชัน ต้องมาดูเหลุผลว่าทำไมต้องใช้อาร์กิวเมนต์หรือพารามิเตอร์ร่วมกับฟังก์ชัน คำตอบง่ายๆ ก็คือ เราต้องการส่งค่าข้อมูลผ่านเข้าไปภายในฟังก์ชัน เพื่อที่จะได้นำค่าข้อมูลเหล่านั้นไปใช้งานกับตัวแปร Local, การคำนวณ หรือใช้กับกลุ่มคำสั่งย่อยภายในฟังก์ชัน

รูปที่ 1 แสดงรูปแบบการกำหนดอาร์กิวเมนต์ให้กับฟังก์ชัน

```
def rectanglearea(อาร์กิวเมนต์ที่1, อาร์กิวเมนต์ที่2)
    a = x * y
    return a
```

รูปที่ 1 แสดงรูปแบบการกำหนดอาร์กิวเมนต์สำหรับให้งานกับฟังก์ชัน

อาร์กิวเมนต์ที่ใช้ร่วมกับฟังก์ชันของ Python จะประกอบด้วย 2 ลักษณะคือ

1. อาร์กิวเมนต์แบบตำแหน่งและจำนวนใช้งานที่แน่นอน (Positional Argument)
2. อาร์กิวเมนต์แบบขยายจำนวนตามค่าข้อมูล (Sequence Unpacking Operator)

อาร์กิวเมนต์แบบตำแหน่งและจำนวนใช้งานที่แน่นอน (Positional Argument)

การใช้งานอาร์กิวเมนต์ลักษณะนี้มักนิยมใช้งานทั่วไปร่วมกับฟังก์ชันปกติ รูปแบบการประกาศอาร์กิวเมนต์ค่อนข้างเขียนง่าย ตำแหน่งของอาร์กิวเมนต์ในการประกาศและใช้งานก็แน่นอนไม่เปลี่ยนแปลง วิธีการผ่านค่าข้อมูลเข้าไปจัดเก็บและเรียกใช้ค่าข้อมูลภายในอาร์กิวเมนต์คล้ายกับการใช้งานตัวแปรทั่วไป ดังแสดงไว้ในตัวอย่างต่อไปนี้

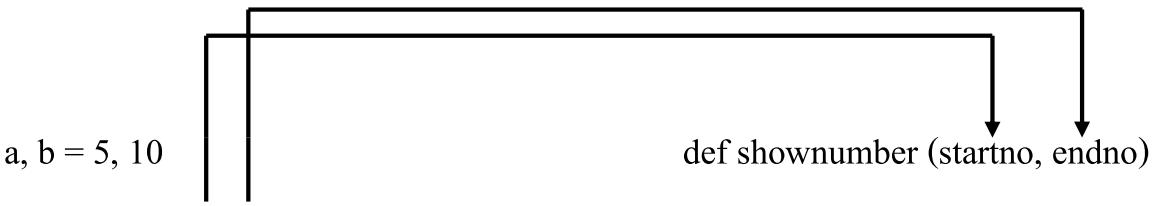
ตัวอย่าง แสดงวิธีประกาศอาร์กิวเมนต์ที่ชื่อว่า startno และ endno

```
def shownumber (startno, endno) :
    for x in range (startno, endno) :
     print(x)
```

การระบุ startno และ endno ให้กับฟังก์ชัน shownumber เป็นทางผ่านค่าข้อมูลจากภายนอกฟังก์ชันเข้าไปใช้งานภายในฟังก์ชัน ด้วยการเรียกใช้งานฟังก์ชันตามตัวอย่างต่อไปนี้

```
a, b = 5, 10
shownumber(a, b)
```

โดยกำหนดค่า 5 ให้กับตัวแปร a และค่า 10 ให้กับตัวแปร b เมื่อเรียกใช้ฟังก์ชัน shownumber ก็กำหนดตัวแปร a ณ ตำแหน่งอาร์กิวเมนต์แรกฟังก์ชัน ซึ่งจะตรงกับอาร์กิวเมนต์ที่ชื่อ startno เช่นเดียวกับตัวแปร b ที่จะกำหนดตำแหน่งอาร์กิวเมนต์ที่ 2 ซึ่งตรงกับอาร์กิวเมนต์ endno



<u>คำสั่งด้านเรียกใช้ฟังก์ชัน</u> <u>ฟังก์ชันและคำสั่งภายใน</u>

รูปที่ 2 แสดงการผ่านค่าข้อมูลจากตัวแปร a และ b ตรงตามตำแหน่งของอาร์กิวเมนต์แต่ละตัว

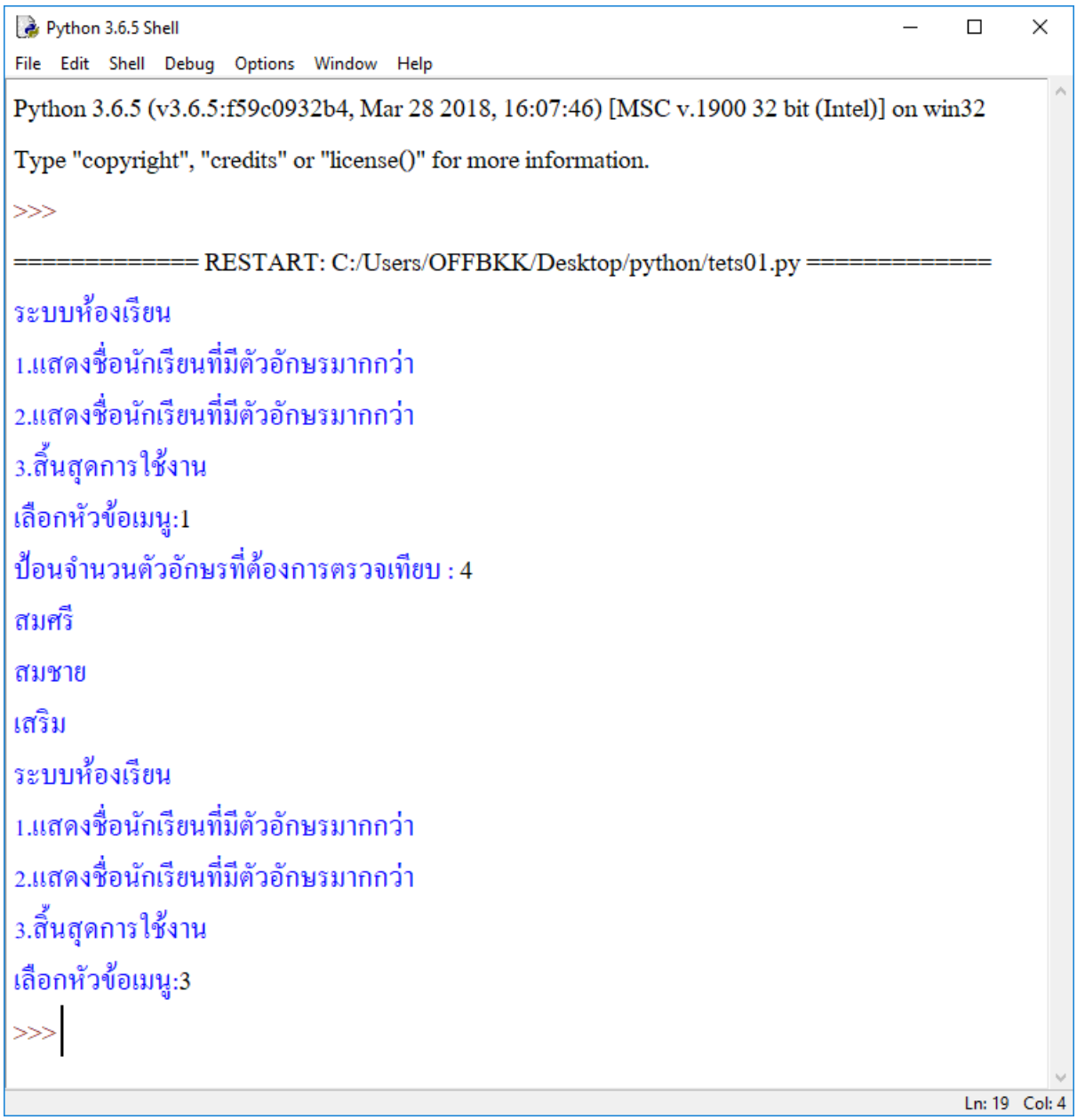
การโอนถ่ายค่าข้อมูลในตัวแปร a ไปยังอาร์กิวเมนต์ startno จะเกิดขึ้นโดยอัตโนมัติ (ชนิดการโอนย้ายข้อมูลระหว่างตัวแปรปกติกับอาร์กิวเมนต์จะเป็นแบบ Tuple ซึ่งมีความหมายว่า เมื่อค่าข้อมูลอาร์กิวเมนต์มีการเปลี่ยนแปลงจากเดิมไป จะไม่ส่งผลให้ค่าข้อมูลในตัวแปรต้นทางเปลี่ยนตามไป ตัวอย่าง ตัวแปร a มีค่าคือ 5

ซึ่งเป็นตัวแปรต้นทาง เมื่อตัวแปร a ผ่านค่าข้อมูลไปยังอาร์กิวเมนต์ startno จาก 5 ไปเป็น 7 ค่าข้อมูลในตัวแปรต้นทางคือ ตัวแปร a ก็คงค่าเท่ากับ 5 ไว้ดังเดิม) เมื่อฟังก์ชันถูกเรียกใช้งานเสนอ ลองพิมพ์กลุ่มสั่งเกี่ยวกับฟังก์ชันร่วมกับอาร์กิวเมนต์แบบที่ 1 ตามตัวอย่างต่อไปนี้

ตัวอย่างที่ 1 แสดงการเรียกใช้ฟังก์ชันพร้อมการผ่านค่าข้อมูลให้กับอาร์กิวเมนต์ของฟังก์ชัน เปิดวินโดว์โมดูลใหม่จาก IDLE ของ Python Shell เพื่อทดลองโปรแกรม

```
englishroom = ['แดง', 'เอ', 'โจ้', 'สมศรี', 'ปาน', 'สมชาย','เสริม']


def namemorechar(room, numchar) :
    for x in room :
        if len(x) > numchar :
            print(x)


def namelesschar(room, numchar) :
    for x in room :
        if len(x) < numchar :
            print(x)


while True :
    print('ระบบห้องเรียน')
    print('1.แสดงชื่อนักเรียนที่มีตัวอักษรมากกว่า')
    print('2.แสดงชื่อนักเรียนที่มีตัวอักษรมากกว่า')
    print('3.สิ้นสุดการใช้งาน')
    a = input('เลือกหัวข้อเมนู:')
```

```
if int(a)  == 1 :
   b= int(input('ป้อนจำนวนตัวอักษรที่ต้องการตรวจเทียบ : '))
   namemorechar (englishroom, b)
elif int(a) == 2 :
   b= int(input('ป้อนจำนวนตัวอักษรที่ต้องการตรวจเทียบ : '))
   namelesschar (englishroom, b)
elif int(a) == 3 :
   break
```



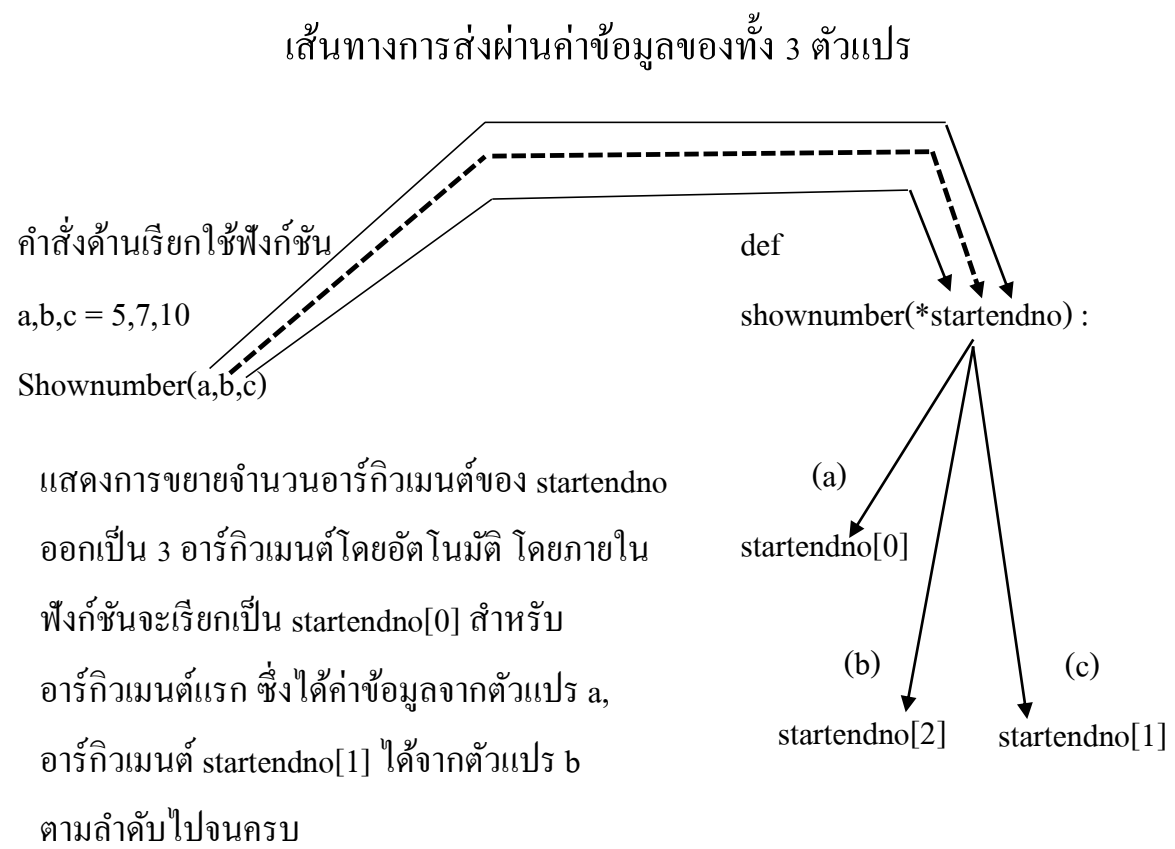
จากตัวอย่างที่ 1 สมมุติให้เป็นโปรแกรมเกี่ยวกับรายชื่อนักเรียนในห้องเรียนภาษาอังกฤษ โดยจัดเก็บรายชื่อนักเรียนไว้ในตัวแปร englishroom ในโปรแกรมตัวอย่างนี้ประกอบด้วย 2 ฟังก์ชันทำหน้าที่ค้นหาชื่อที่มีตัวอักษรของชื่อมากน้อยตามที่ต้องการ ยกตัวอย่าง ถ้าต้องการแสดงรายชื่อนักเรียนที่มีมากกว่า 3 ตัวอักษร ให้เลือกเมนูหมายเลข 1 จะเป็นการเลือกฟังก์ชันชื่อ namemorechar ที่ประกอบด้วยอาร์กิวเมนต์ชื่อ room สำหรับรับค่าข้อมูลที่เป็นรายชื่อนักเรียน ซึ่งจะได้จากตัวแปร englishroom และอาร์กิวเมนต์ชื่อ numchar สำหรับรับจำนวนตัวอักษรที่ต้องการเปรียบเทียบในฟังก์ชัน โดยจะได้ค่าจากตัวแปร b หรือหากต้องการแสดงรายชื่อนักเรียนที่มีน้อยกว่า 3 ตัวอักษร ให้เลือกเมนูหมายเลข 2 ซึ่งจะเรียกใช้ฟังก์ชันชื่อ namemorechar ที่มี 2 อาร์กิวเมนต์และมีการทำงานคล้ายๆ กับฟังก์ชันแรก

อาร์กิวเมนต์แบบขยายจำนวนตามค่าข้อมูล (Sequence Unpacking Operator)

การสร้างฟังก์ชันที่มีอาร์กิวเมนต์สำหรับใช้งาน โดยส่วนใหญ่จะกำหนดอาร์กิวเมนต์ใช้งานเป็นจำนวนที่แน่นอน แต่ในกรณีที่ต้องการสร้างฟังก์ชันที่จำเป็นต้องมีอาร์กิวเมนต์ แต่ไม่ต้องการกำหนดจำนวนที่แน่นอน แบบนี้ควรจะใช้อาร์กิวเมนต์แบบ Sequence Unpacking Operator (ในภาษา Python จะใช้เครื่องหมาย * นำหน้าอาร์กิวเมนต์แบบนี้) ลองมาดูรูปแบบการกำหนดอาร์กิวเมนต์แบบนี้ร่วมกับฟังก์ชันตามตัวอย่างต่อไปนี้

```
def shownumber(*startendno) :
        for x in range(startendno[0], startendno[1])
        print(x)
```

จากตัวอย่างนี้ฟังก์ชัน shownumber จะมีเพียงแค่หนึ่งอาร์กิวเมนต์ที่ชื่อ *startendno (ซึ่งในตัวอย่างจากหัวข้อที่ 1 ซึ่งเป็นแบบ Positional Argument จะมี 2 อาร์กิวเมนต์คือ startno endno) แต่สามารถรองรับการขยายจำนวนอาร์กิวเมนต์ที่ต้องใช้งานโดยอัตโนมัติ ตามรูป 9 แสดงการเรียกใช้ฟังก์ชันที่มีอาร์กิวเมนต์แบบขยายจำนวนตามค่าข้อมูล

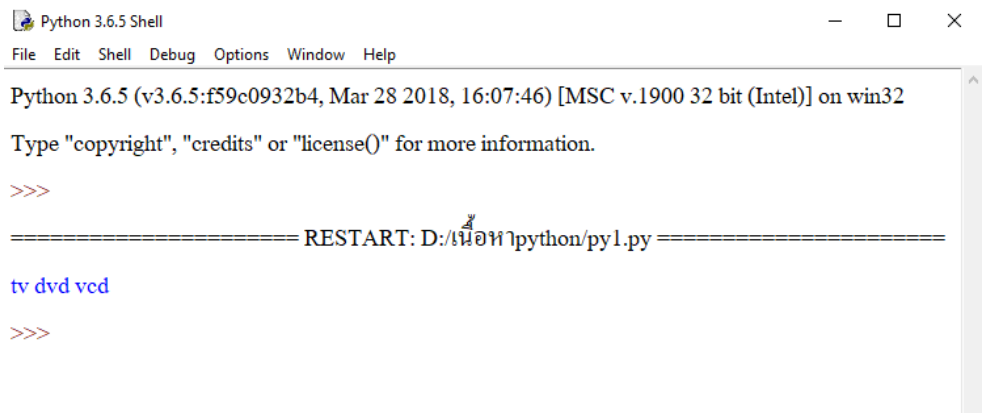


ภาพแสดงการเรียกใช้ฟังก์ชันและผ่านค่าข้อมูลไปให้กับอาร์กิวเมนต์

ทดลองการใช้งานอาร์กิวเมนต์ตามตัวอย่างโปรแกรมต่อไปนี้

ตัวอย่างที่ 1 เปิดวินโดว์โมดูลใหม่จาก IDLE ของ Python Shell ทดลองป้อนคำสั่งต่อไปนี้

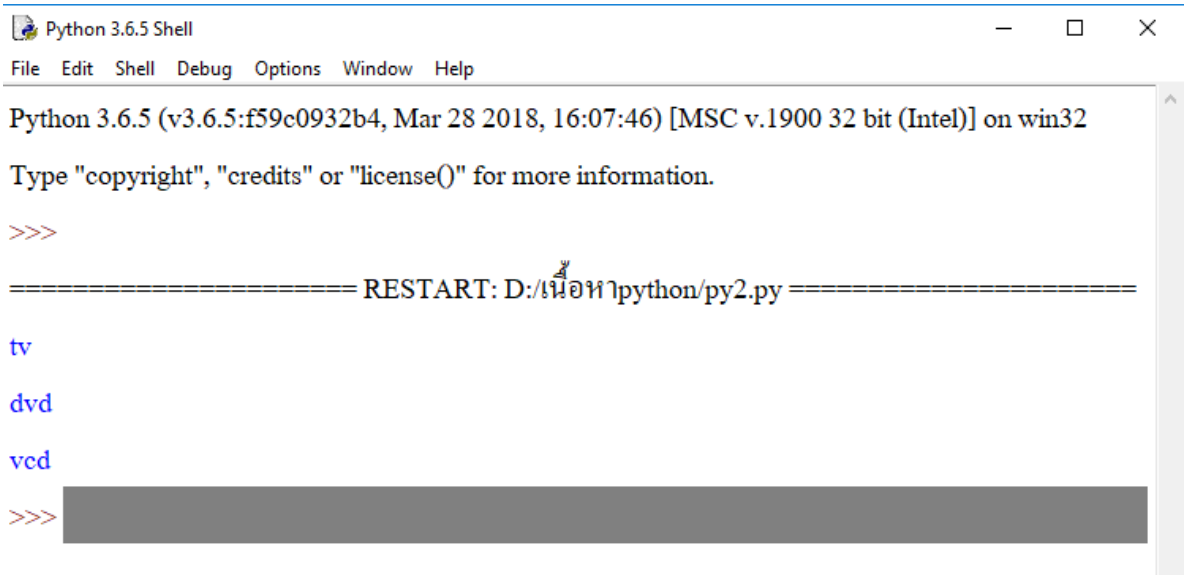
```
def showitem(*item):
   print(item[0], item[1], item[2])


showitem('tv','dvd','vcd')
```



จากตัวอย่างที่ 1 แสดงการผ่านค่าตัวแปรจำนวน 3 ค่าไปยังฟังก์ชัน showitem ซึ่งภายในฟังก์ชันจะแสดงวิธีการเขียนโปรแกรมดึงค่าจากอาร์กิวเมนต์แบบขยายจำนวนตามค่าข้อมูลด้วย (Item[0], Item[1], Item[2])

ตัวอย่างที่ 2 แสดงวิธีดึงค่าจากอาร์กิวเมนต์ในฟังก์ชันด้วยคำสั่ง for

```
def showitem(*item) :
   for x in item :
        print(x)


showitem('tv','dvd','vcd')
```



ตัวอย่างนี้แสดงการผ่านค่าจากตัวแปรต้นทางจำนวน 3 ค่า ผ่านอาร์กิวเมนต์ที่ชื่อ * item เมื่อต้องการใช้ค่าในอาร์กิวเมนต์ภายในฟังก์ชันจะใช้วิธีอ่านค่าด้วยคำสั่งวนทำซ้ำหรือคำสั่ง for เพื่ออ่านค่าที่เก็บไว้ในอาร์กิวเมนต์ * item ตั้งแต่ค่าแรกไปจนค่าสุดท้าย (ลักษณะค่าที่จัดเก็บใน * item จะมีวิธีเก็บเหมือนกับตัวแปร Collection Data ประเภท Tuple ภาพจำลองการเก็บค่าข้อมูลในอาร์กิวเมนต์เป็นดังนี้

* item = (TV', 'DVD', 'VCD')

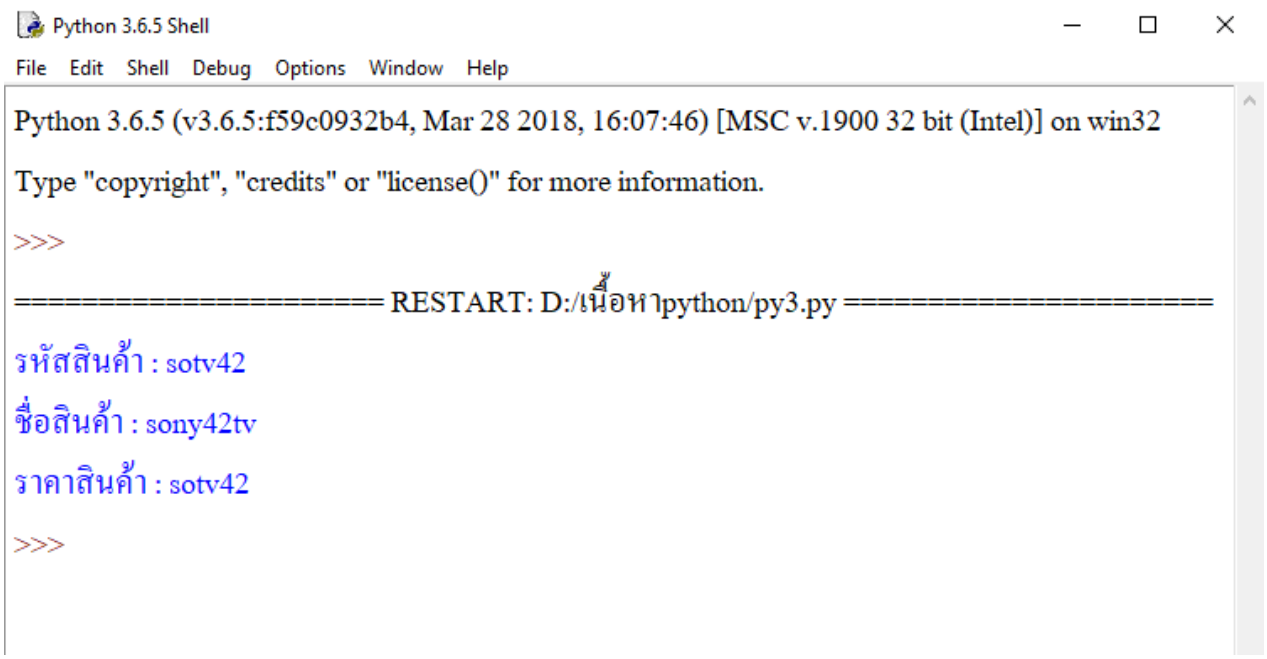
↓ ↓ ↓

การอ้างตำแหน่งโดยตรง (Item[0], Item[1], Item[2])

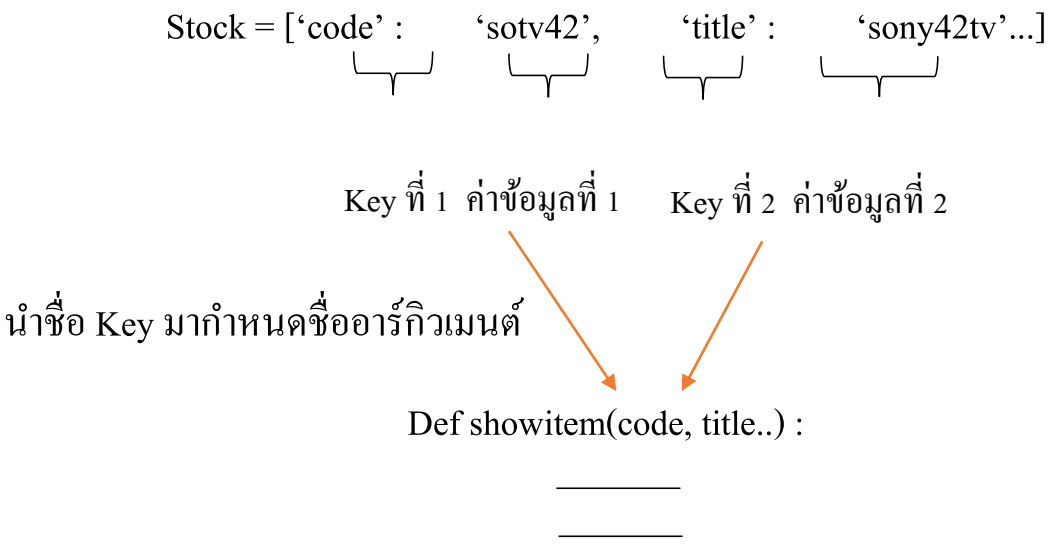
ดังนั้นการเรียกใช้ค่าภายในอาร์กิวเมนต์จึงเรียกได้จากตำแหน่งโดยอ้างอิงหมายเลข Index ตามตัวอย่างที่ 1 หรือใช้วิธีการอ่านจากคำสั่ง for ในตัวอย่างที่ 2

ตัวอย่างที่ 3 แสดงโปรแกรมการใช้อาร์กิวเมนต์แบบ Sequence Unpacking Operator โดยผ่านตัวแปร Collection Data ประเภท Dictionary ด้วยเทคนิคการผ่านค่าข้อมูล **- Operator

```
stock = {'code' : 'sotv42','title' : 'sony42tv','cost' : '50000'}


def showitem(code, title, cost) :
        print('รหัสสินค้า : %s' %code)
        print('ชื่อสินค้า : %s' %title)
        print('ราคาสินค้า : %s' %code)
showitem(**stock)
```



ตัวอย่างที่ 3 เป็นเทคนิคการใช้ Unpacking Operator ในฝั่งค่าข้อมูลต้นทาง ให้สังเกตที่ตัวแปรต้นทาง ซึ่งปกติจะมีจำนวนตัวแปรเท่ากับจำนวนค่าข้อมูลที่ต้องการผ่านไปยังอาร์กิวเมนต์ของฟังก์ชัน แต่ในตัวอย่างนี้ตัวแปรต้นทางจะมีเพียงตัวเดียว และเป็นตัวแปรชนิด Dictionary คือมีค่าข้อมูลในตัวแปรได้มากกว่าหนึ่งค่า ในที่นี้ประกอบด้วย code : sotv42, title : sony42tv, cost : 50000 ซึ่งนับได้จำนวน 3 ค่าข้อมูล เมื่อส่งผ่านไปยังอาร์กิวเมนต์ของฟังก์ชัน ซึ่งประกอบด้วย code, title, cost (การตั้งชื่ออาร์กิวเมนต์ของฟังก์ชันเมื่อต้องใช้เทคนิคการส่งผ่านค่าจากตัวแปรต้นทางหนึ่งตัวแปรหลายค่าข้อมูล ชื่ออาร์กิวเมนต์จำเป็นต้องตั้งชื่อให้ตรงกับชื่อของคีย์ (Key) ในค่าข้อมูลต้นทางเสมอ



จึงจะสามารถส่งผ่านข้อมูลด้วยเทคนิค Unpacking Operator ต้นทางได้ จากตัวแปร stock ซึ่งอาศัยเทคนิคการกระจายค่าข้อมูลภายในแบบ Dictionary ให้ลงพอดีกับอาร์กิวเมนต์จึงต้องระบุเครื่องหมาย ** ไว้หน้าตัวแปรต้นทาง ซึ่งจะเขียนได้เป็น **stock ร่วมกับการเรียกฟังก์ชัน showitem(**stock) เพื่อใช้งาน

ความแตกต่างระหว่างฟังก์ชันที่ใช้และไม่ใช้คำสั่ง return

การสร้างฟังก์ชันเพื่อใช้งานในโมดูลบางกรณี เมื่อมีการประมวลผลภายในฟังก์ชันเสร็จสิ้น ก็มีความจำเป็นที่ต้องการผลลัพธ์ที่ได้จากการประมวลผลภายใน ส่งกลับออกไปใช้งานภายนอกฟังก์ชัน ด้วยเงื่อนไขดังที่ได้กล่าวมา จึงมีความจำเป็นที่ต้องเพิ่มเติมคำสั่งบางคำสั่งลงภายในฟังก์ชัน เพื่อส่งผลลัพธ์นั้นกลับไปจุดที่เรียกใช้งานฟังก์ชัน ซึ่งคำสั่งนั้นก็คือ คำสั่ง return นั่นเอง ต่อไปนี้เป็นการแสดงรูปแบบการใช้คำสั่ง return ภายในฟังก์ชัน Factorial ตามตัวอย่าง

```
def factorial(n):
    a = 1
    for x in range(1, n) :
        a = a * x
    return a
```

เมื่อมีการคำนวณค่าของ Factorial ภายใต้คำสั่ง for เสร็จสิ้นแล้วจะทำการส่งค่าที่อยู่ในตัวแปร a กลับไปจุดที่เรียกใช้ฟังก์ชัน ตามตัวอย่างการเรียกใช้ฟังก์ชัน Factorial ดังนี้

Resultfactorial = factorial(5)

การเรียกใช้งานฟังก์ชันที่มีการส่งค่าผลลัพธ์กลับมาจำเป็นต้องมีตัวแปรต้นทางรองรับค่าผลลัพธ์ที่ได้จากฟังก์ชัน ซึ่งในที่นี้คือตัวแปรต้นทาง resultfactorial ในทางตรงกันข้ามกับฟังก์ชันที่ไม่มีคำสั่ง return ภายในฟังก์ชัน ก็ไม่มีความจำเป็นต้องมีตัวแปรต้นทางรอรับค่า เนื่องจากฟังก์ชันนั้นๆ จะไม่ส่งค่าผลลัพธ์ใดๆ กลับออกมายังจุดที่ทำการเรียกใช้ฟังก์ชันนั่นเอง

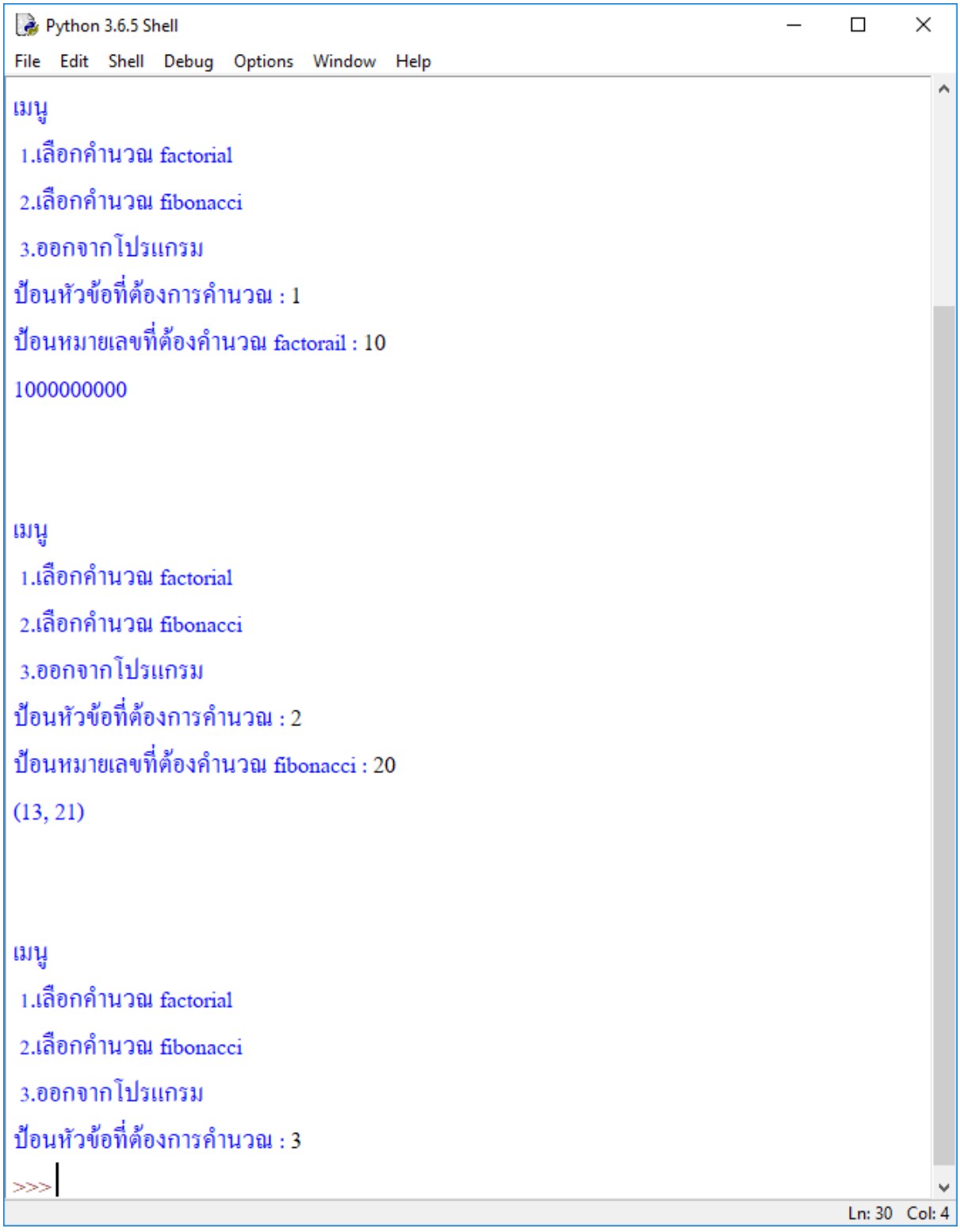
ทดลองโปรแกรมตามตัวอย่างต่อไปนี้ สำหรับการใช้คำสั่ง return ร่วมกับฟังก์ชัน

ตัวอย่างที่ 1 เปิดวินโดว์โมดูลใหม่จาก IDLE ของ Python Shell และทดลองป้อนคำสั่งต่อไปนี้

py4.py - C:\Users\OFFBKK\Desktop\python\py4.py (3.6.5)

File Edit Format Run Options Window Help

```
def factorial(n) :
    a = 1
    for x in range(1, n) :
        a = a*n
    return a
def fibonacci(n) :
    a, b = 0, 1
    while b < n :
        a, b = b, a+ b
    return a, b
while True :
    print('\n\nเมนู\n 1.เลือกคำนวณ factorial \n 2.เลือกคำนวณ fibonacci \n 3.ออกจากโปรแกรม')
    choice = input('ป้อนหัวข้อที่ต้องการคำนวณ : ')
    if int(choice) == 1 :
        x = int(input('ป้อนหมายเลขที่ต้องคำนวณ factorail : '))
        y = factorial(x)
        print(y)
    elif int(choice) == 2 :
        x = int(input('ป้อนหมายเลขที่ต้องคำนวณ fibonacci : '))
        y = fibonacci(x)
        print(y)
    elif int(choice) == 3 :
        break
```



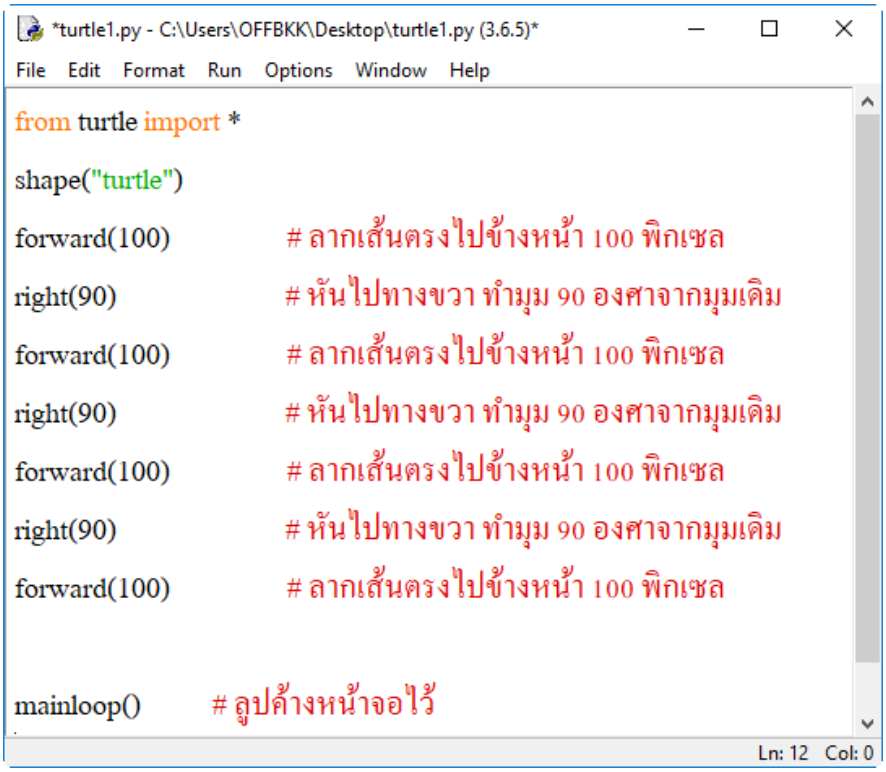
ตัวอย่างข้างต้นเป็นโปรแกรมสำหรับคำนวณค่า Factorial และ Fibonacci โดยเลือกจากเมนูที่ต้องการคำนวณ เช่น เลือกเมนูหมายเลข 1 สำหรับคำนวณค่า Factorial โดยเรียกใช้ฟังก์ชันที่ชื่อ Factorial พร้อมส่งค่าข้อมูลผ่านอาร์กิวเมนต์ของฟังก์ชัน เข้าสู่กระบวนการคำนวณกลุ่มคำสั่งในลูปหรือคำสั่ง for จน

ได้คำตอบสุดท้ายที่เก็บค่าผลลัพธ์นั้น ตัวแปร a ในขั้นตอนสุดท้ายของฟังก์ชันคือการใช้คำสั่ง return ค่าในตัวแปร a กลับออกไปข้างนอกฟังก์ชัน ซึ่งจะส่งผ่านค่าข้อมูลไปยังตัวแปรต้นทางที่ชื่อ y สำหรับฟังก์ชัน Fibonacci ก็มีลักษณะการ return ผลลัพธ์เช่นเดียวกัน

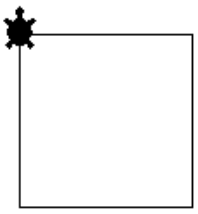
บทที่ 7 ฝึกเขียนโปรแกรมกับเต่าไพทอน

ไพทอนมีโมดูล turtle ซึ่งมีต้นฉบับมาจากภาษา Logo โดยจะมีในโปรแกรม Python เวอร์ชั่น Python 2.6 และ Python 3 เป็นต้นมา สำหรับใช้ฝึกเขียนโปรแกรมขั้นเริ่มต้น โดยใช้เต่าในการลากเส้นหรือวาดรูป ในที่นี้จะแนะนำให้ใช้โหมดสคริปต์

ตัวอย่าง ให้ทดลองเขียนคำสั่งดังต่อไปนี้



เมื่อรันโปรแกรม จะได้ผลลัพธ์คือ



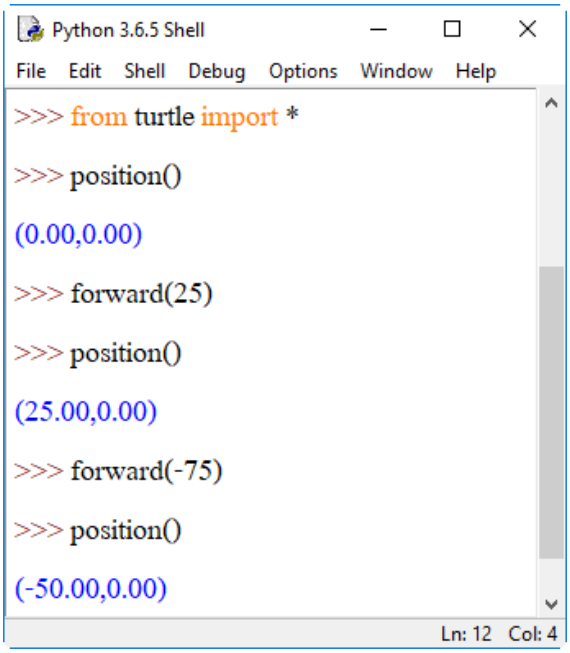
บรรทัดบนสุดของโค้ดคำสั่ง จะต้องใช้คำสั่ง from turtle import * เพื่อเรียกใช้ตัวแปร ฟังก์ชัน หรือเมธอดทั้งหมดจากโมดูล turtle เพื่อใช้ในการวาดภาพได้

คำสั่งที่ใช้ในการสั่งให้เต่าเคลื่อนที่

1. forward() / fd() ลากเส้นตรงไปข้างหน้ามีหน่วยเป็นพิกเซล โดยภายในวงเล็บใส่ตัวเลขที่เป็นชนิด integer หรือ float เช่น

forward(100) เต่าจะเดินหน้า 100 พิกเซล

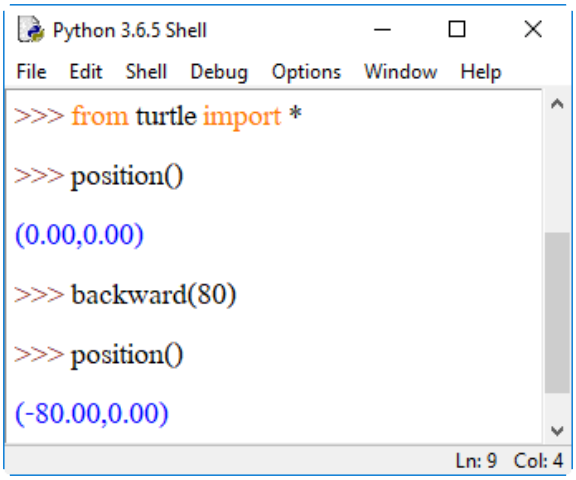
forward(-100) เต่าจะเดินหน้า -100 พิกเซล หรือถอยหลัง 100 พิกเซลนั่นเอง (ทิศตรงข้าม)

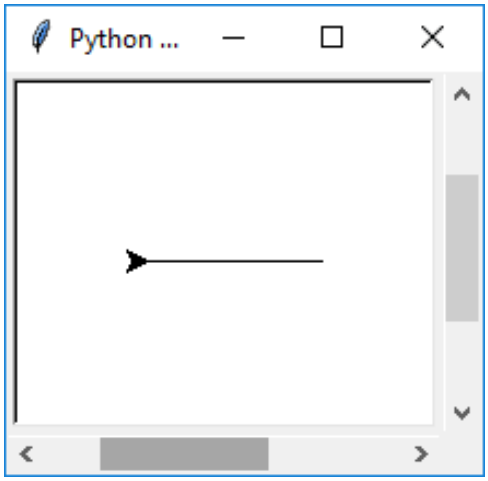


2. back() / bk() / backward() ลากเส้นตรงไปข้างหลัง มีหน่วยเป็นพิกเซล เช่น

backward(100) เต่าจะเดินถอยหลัง 100 พิกเซล

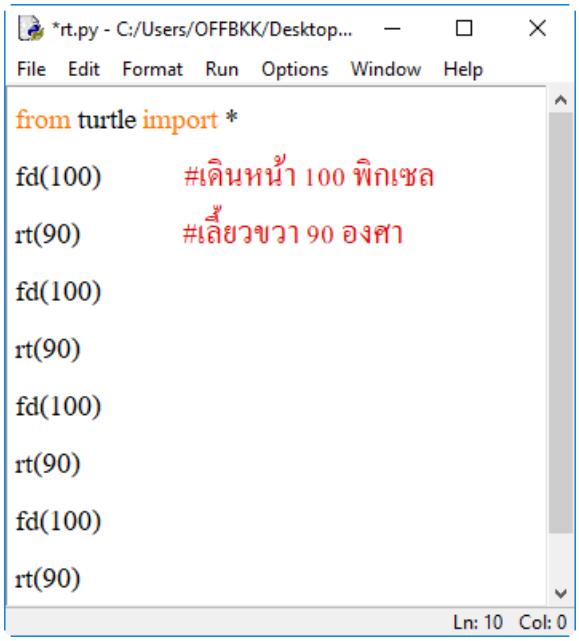
backward(-100) เต่าจะเดินถอยหลัง -100 พิกเซล หรือเดินหน้า 100 พิกเซลนั่นเอง

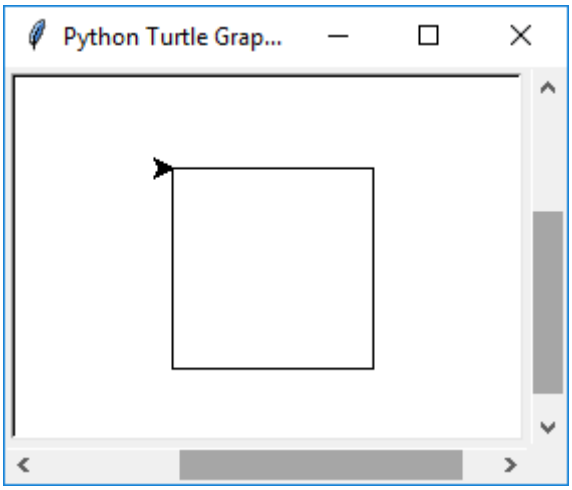




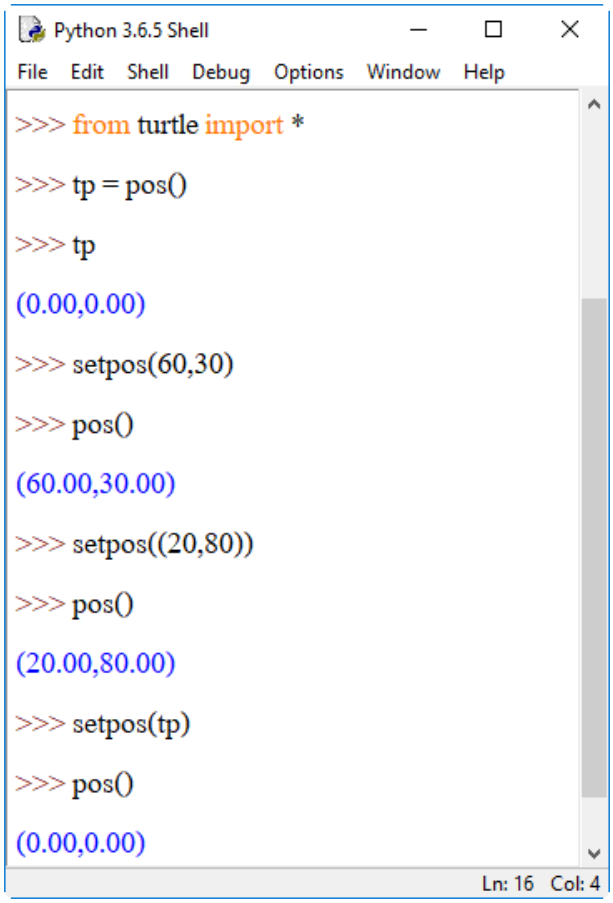
3. right() / rt() หันไปทางขวา ทำมุมตามองศาที่กำหนดจากมุมเดิม เช่น right(90) หันหัวเต่าไปทางขวา 90 องศาจากมุมเดิม

4. left() / lt() หันไปทางซ้าย ทำมุมตามองศาที่กำหนดจากมุมเดิม เช่น left(90) หันหัวเต่าไปทางซ้าย 90 องศาจากมุมเดิม



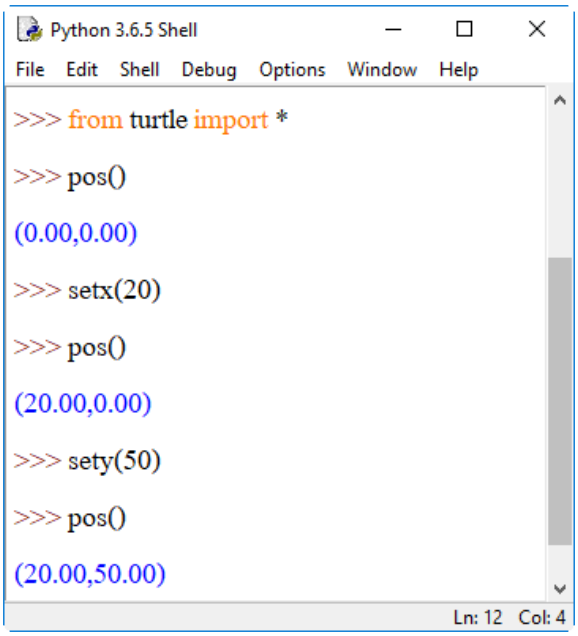


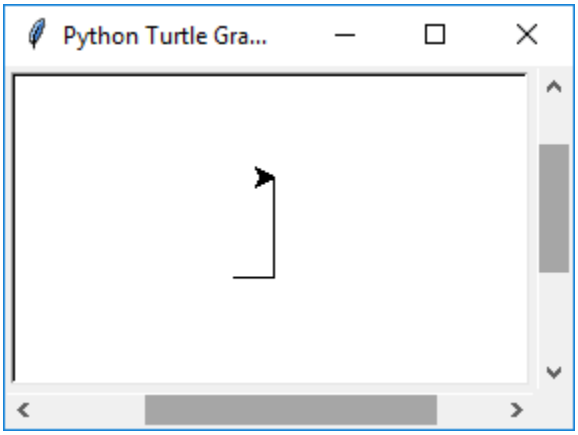
5. goto(x, y) / setpos(x, y) / setposition(x, y) ย้ายเต่าไปยังตำแหน่งที่แน่นอน โดยไม่เปลี่ยนทิศทางของเต่า ตัวอย่างเช่น



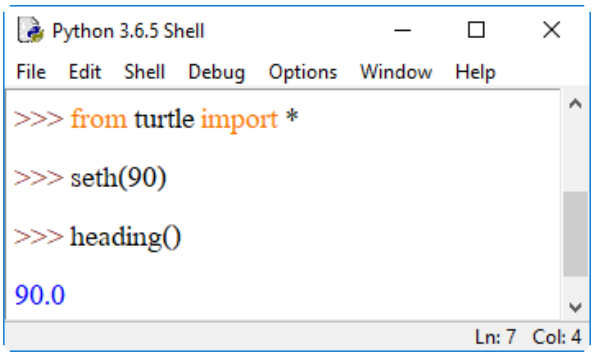
6. setx() ตั้งค่าพิกัดในแกน x ของเต่า โดยที่พิกัดในแกน y ไม่มีการเปลี่ยนแปลง

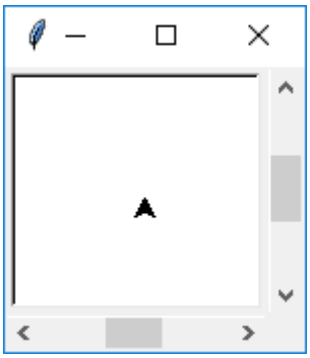
7. sety() ตั้งค่าพิกัดในแกน y ของเต่า โดยที่พิกัดในแกน x ไม่มีการเปลี่ยนแปลง



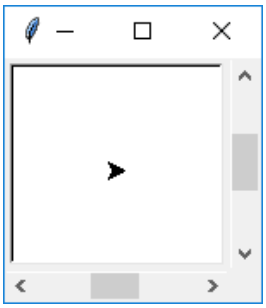


8. setheading() / seth() กำหนดหัวเต่าให้หันไปตามทิศที่ต้องการ ในวงเล็บให้ใส่ตัวเลขของศาเข้าไป โดย 0 คือทิศตะวันออก, 90 คือ ทิศเหนือ, 180 คือ ทิศตะวันตก และ 270 คือ ทิศใต้





9. home() ย้ายเต่าไปยังจุดเริ่มต้น นั่นก็คือ พิกัด (0,0) และตั้งค่าหัวเต่าไว้ที่ทิศทางเริ่มต้น นั่นคือ 0 องศา หรือชี้ไปทางขวา

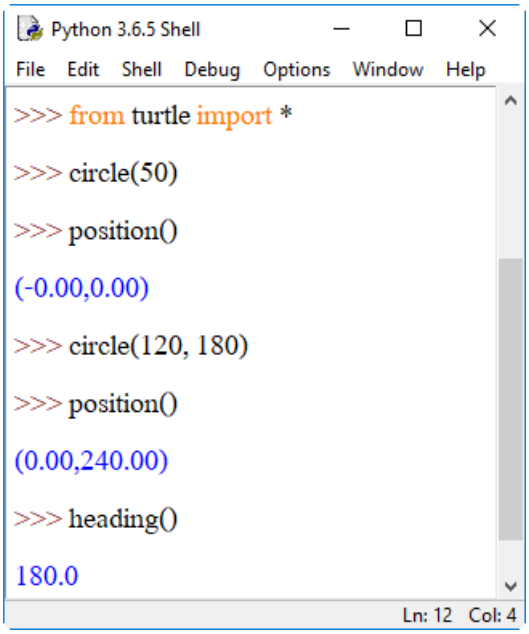


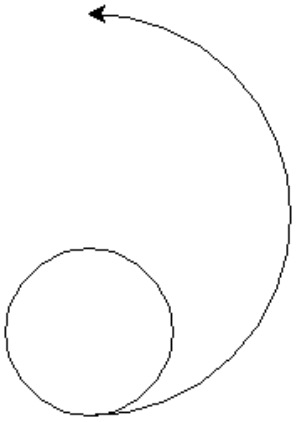
10. circle(รัศมี, องศา, จำนวนเหลี่ยม) ใช้วาดรูปวงกลม

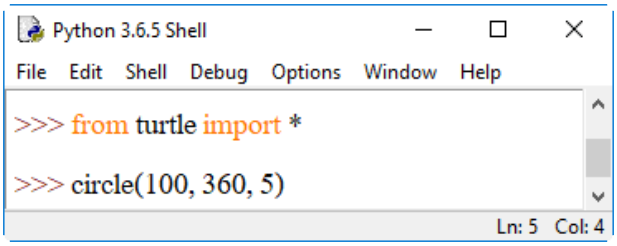
- circle(รัศมี) ใช้วาดรูปวงกลม เช่น circle(50) จะได้วงกลมที่มีรัศมีเท่ากับ 50

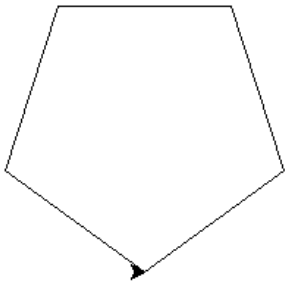
- circle(รัศมี, องศา) ใช้วาดรูปวงกลมไม่ครบวง ตามองศาที่กำหนด เช่น circle(50,180) จะได้ครึ่งวงกลมที่มีรัศมีเท่ากับ 50

- circle(รัศมี, องศา, จำนวนเหลี่ยม) ใช้วาดรูปหลายเหลี่ยม เช่น circle(100,360,5) จะได้วงกลมที่มีห้าเหลี่ยม

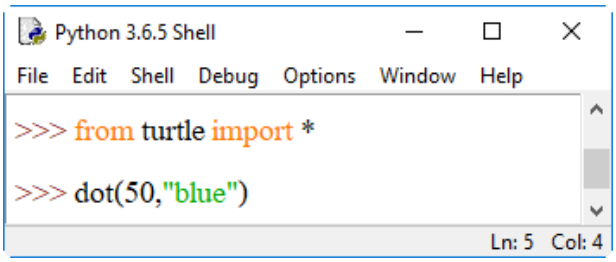


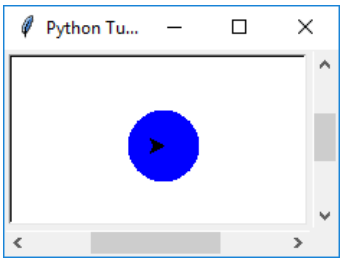






11. dot(ขนาดเส้นผ่านศูนย์กลาง, "สี") ใช้วาดจุดวงกลม โดยระบุเส้นผ่านศูนย์กลางและสี





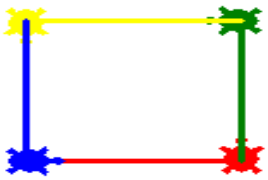
12. stamp() ปั๊มตัวเต่าให้อยู่ในตำแหน่งที่กำหนด เช่น เมื่อเดินเต่าเดินหน้าไป 100 พิกเซล ให้ทำการปั๊มรูปเต่าไว้ตำแหน่งนั้น ตัวอย่างเช่น

turtle3.py - C:/Users/OFFBKK/Desktop/turtle3.py (3.6.5)

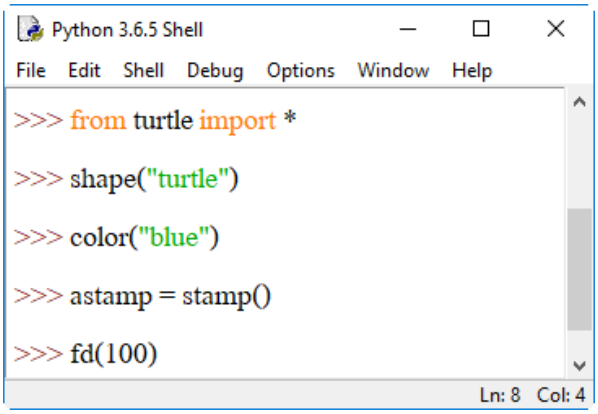
File Edit Format Run Options Window Help

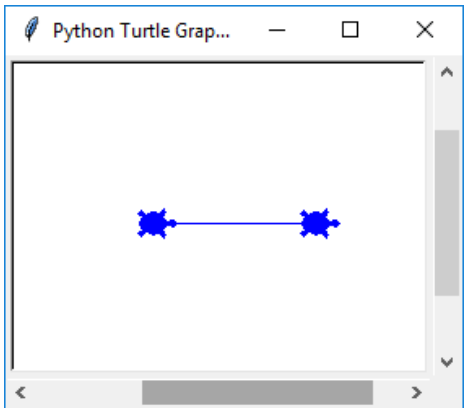
```
from turtle import *
pensize(5)              # ใส่ขนาดเส้น
shape("turtle") # ใส่ตัวละครเต่า turtle ลงไป

color("red")            # ใส่สีแดง ให้กับเส้น
forward(100)            # ลากเส้นตรงไปข้างหน้า 100 พิกเซล
stamp()

color("green")          # ใส่สีเขียว ให้กับเส้น
left(90)                # หันไปทางขวา ทำมุม 90 องศาจากมุมเดิม
forward(100)            # ลากเส้นตรงไปข้างหน้า 100 พิกเซล
stamp()

color("yellow")         # ใส่สีเหลือง ให้กับเส้น
left(90)                # หันไปทางขวา ทำมุม 90 องศาจากมุมเดิม
forward(100)            # ลากเส้นตรงไปข้างหน้า 100 พิกเซล
stamp()

color("blue")           # ใส่สีน้ำเงิน ให้กับเส้น
left(90)                # หันไปทางขวา ทำมุม 90 องศาจากมุมเดิม
forward(100)            # ลากเส้นตรงไปข้างหน้า 100 พิกเซล
stamp()
```

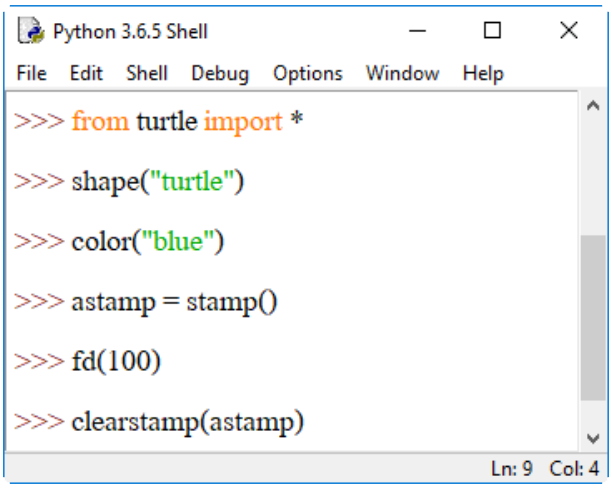
Ln: 24 Col: 10

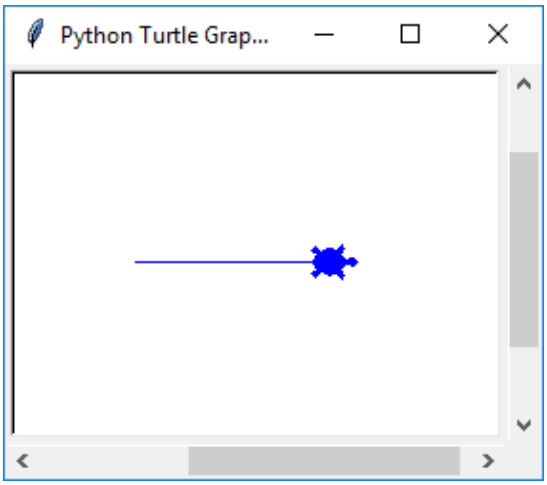


13. clearstamp() ลบตัวเต่าที่เราทำการปั๊มเอาไว้

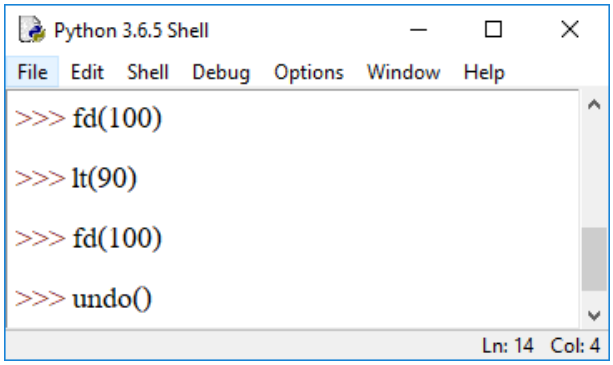








14. undo() ยกเลิก (ซ้ำๆ) การกระทำของเต่าครั้งล่าสุด

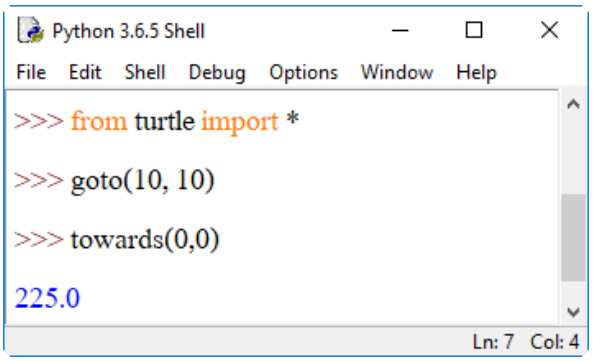


15. speed() ใส่ความเร็วในการเดินของตัวเต่า มีค่าตั้งแต่ 0 - 10 โดย 0 = เร็วที่สุด, 10 = เร็ว, 6 = ปกติ, 3 = ช้า, 1= ช้าที่สุด เช่น speed(3)

คำสั่งที่ใช้สอบถามค่าต่างๆ ของตัวเต่า

1. position() / pos() ส่งกลับค่าตำแหน่งปัจจุบันของเต่า (x, y)

2. towards(x, y) ส่งกลับค่ามุมระหว่างเส้นจากตำแหน่งเต่าไปยังตำแหน่งที่ระบุ



3. xcor() ส่งกลับค่าพิกัดแกน x ของตัวเต่า

4. ycor() ส่งกลับค่าพิกัดแกน y ของตัวเต่า

5. heading() ส่งกลับค่าองศาของหัวเต่า

6. distance(x, y) ส่งกลับค่าระยะทางจากเต่าไปยังตำแหน่ง (x, y) ที่อ้างอิง

คำสั่งควบคุมปากกา (Pen Control)

1. pendown() / pd() / down() เป็นคำสั่งวางปากกา

2. penup() / pu() / up() เป็นคำสั่งยกปากกา

3. pensize() เป็นคำสั่งสำหรับใช้กำหนดขนาดตัวเต่าและขนาดของเส้นเป็นหน่วยเป็นพิกเซล เช่น pensize(3)

4. pen() ตั้งค่าแอตทริบิวต์หลายๆ อย่างของปากกาในคำสั่งเดียว ซึ่งมีค่าดังต่อไปนี้ :

- shown แสดงหรือซ่อนปากกา มี 2 ค่า คือ True/False

- pendown วางปากกา มี 2 ค่า คือ True/False

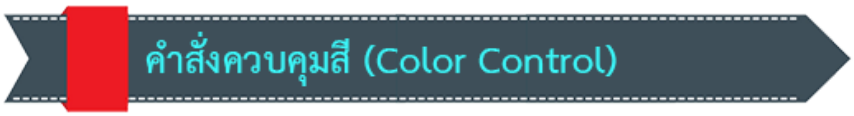
- pencolor ใส่สีปากกา โดยระบุชื่อสีลงไป เช่น pencolor = "red"

- fillcolor เติมสีวัตถุแบบปิด โดยระบุชื่อสีลงไป เช่น fillcolor = "red"

- pensize ใส่ขนาดปากกา

- speed ใส่ความเร็วในการวาดของปากกา มีค่าตั้งแต่ 0 - 10

5. isdown() ส่งกลับค่า True ถ้าสถานะเป็นวางปากกา และส่งกลับค่า False ถ้าสถานะเป็นยกปากกา



1. pencolor() ส่งกลับค่าหรือตั้งค่าสีปากกา โดยมี 3 รูปแบบ คือ

- pencolor() ส่งกลับค่า pencolor ปัจจุบัน ส่งค่ากลับมาเป็นสตริงระบุสี

- pencolor(ชื่อสี) ตั้งค่า pencolor เป็น colorstring เช่น "red", "yellow" หรือ "# 33cc8c"

- pencolor(r, g, b) ตั้งค่า pencolor เป็นสี RGB โดยแต่ละ R, G และ B ต้องอยู่ในช่วง 1.0 - 255

2. fillcolor() ส่งกลับค่าหรือตั้งค่าการเติมสี โดยมี 3 รูปแบบ คือ

- fillcolor() ส่งกลับค่า fillcolor ปัจจุบัน ส่งค่ากลับมาเป็นสตริงระบุสี

- fillcolor(ชื่อสี) ตั้งค่า fillcolor เป็น colorstring เช่น "red", "yellow" หรือ "# 33cc8c"

- fillcolor(r, g, b) ตั้งค่า fillcolor เป็นสี RGB โดยแต่ละ R, G และ B ต้องอยู่ในช่วง 1.0 - 255

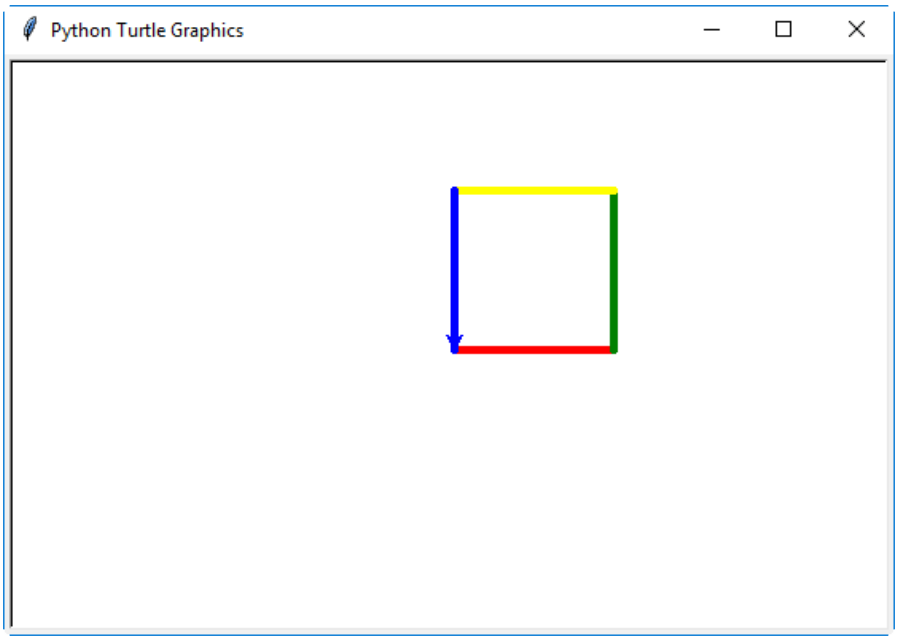
3. color() เป็นคำสั่งกำหนดสีให้กับเส้น (ชื่อสีในภาษาอังกฤษ) เช่น color("yellow") ตัวอย่างเช่น

turtle3.py - C:/Users/OFFBKK/Desktop/turtle3.py (3.6.5)

File Edit Format Run Options Window Help

```
from turtle import *
pensize(5)          # ใส่ขนาดเส้น

color("red")        # ใส่สีแดง ให้กับเส้น
forward(100)        # ลากเส้นตรงไปข้างหน้า 100 พิกเซล

color("green")      # ใส่สีเขียว ให้กับเส้น
left(90)            # หันไปทางขวา ทำมุม 90 องศาจากมุมเดิม
forward(100)        # ลากเส้นตรงไปข้างหน้า 100 พิกเซล

color("yellow")     # ใส่สีเหลือง ให้กับเส้น
left(90)            # หันไปทางขวา ทำมุม 90 องศาจากมุมเดิม
forward(100)        # ลากเส้นตรงไปข้างหน้า 100 พิกเซล

color("blue")       # ใส่สีน้ำเงิน ให้กับเส้น
left(90)            # หันไปทางขวา ทำมุม 90 องศาจากมุมเดิม
forward(100)        # ลากเส้นตรงไปข้างหน้า 100 พิกเซล

mainloop()
```

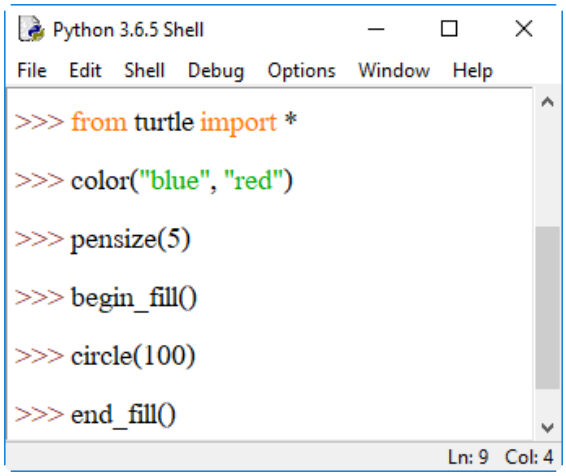
Ln: 20 Col: 0

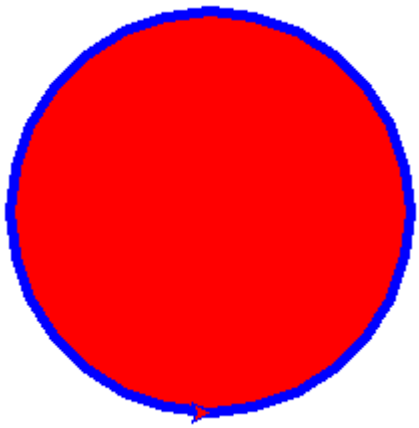


คำสั่งในการเติมสี

1. begin_fill() กำหนดจุดเริ่มต้นในการเติมสี โดยจะต้องใส่คำสั่งนี้ก่อนคำสั่งวาดรูปทรง

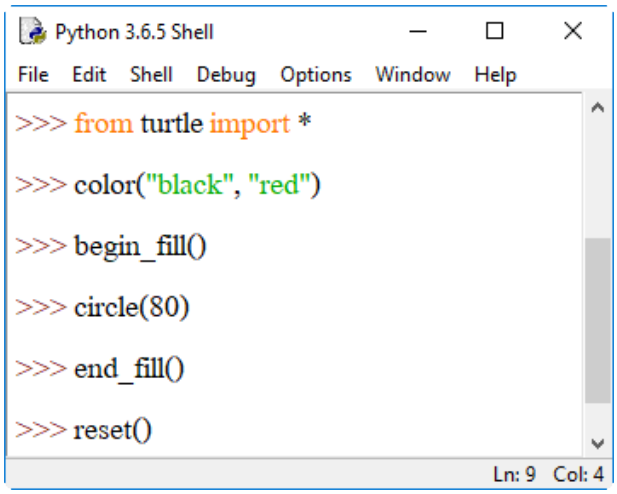
2. end_fill() เติมสีรูปทรง โดยจะต้องใส่คำสั่งนี้หลังคำสั่ง begin_fill()





คำสั่งควบคุมการวาดภาพเพิ่มเติม

1. reset() เป็นคำสั่งที่ใช้ลบภาพวาดของเต่าออกจากหน้าจอ ปรับตำแหน่งตัวเต่ามาอยู่ตรงกลางหน้าจออีกครั้ง และตั้งค่าตัวแปรให้เป็นค่าเริ่มต้น



2. clear() เป็นคำสั่งที่ใช้ลบภาพวาดของเต่าออกจากหน้าจอ แต่ตำแหน่งของเต่าและภาพวาดของเต่าอื่น ๆ จะไม่ได้รับผลกระทบ

3. write(arg, move, align, font) ใช้เขียนข้อความ เช่น write("Python", True, align = "left", font = ("Arial", 8, "normal"))

- arg การแทนสตริงของอาร์กิวเมนต์ เช่น "Python"

- move หากเป็น True ปากกาจะถูกย้ายไปที่มุมล่างขวาของข้อความ โดยค่าเริ่มต้นย้ายเป็น False

- align การจัดตำแหน่งข้อความ ได้แก่ "left", "center" และ "right" เช่น align = "left"

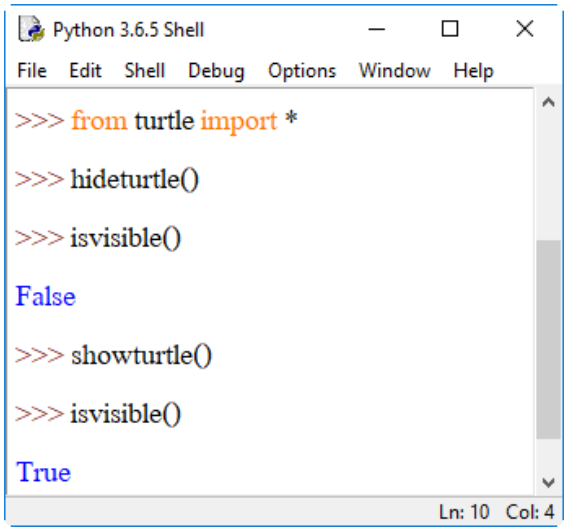
- font กำหนดรูปแบบตัวอักษร ได้แก่ fontname, fontsize, fonttype เช่น font = ("Arial", 8, "normal")

คำสั่งกำหนดค่าตัวเต่า

1. hideturtle() / ht() คำสั่งในการซ่อนเต่าไม่ให้มองเห็นตัวเต่า เมื่ออยู่ในระหว่างการทำภาพวาดที่ซับซ้อนบางส่วน เพราะการซ่อนเต่าจะช่วยให้การวาดภาพสามารถสังเกตได้ง่ายขึ้น

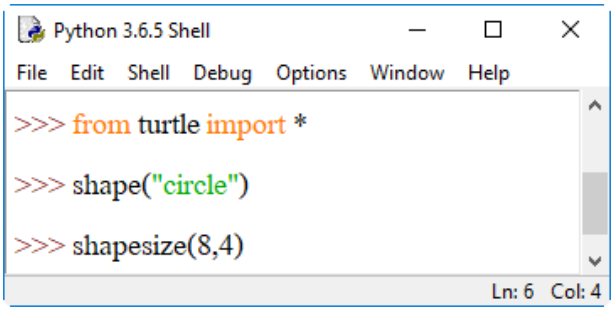
2. showturtle() / st() คำสั่งในการแสดงเต่า

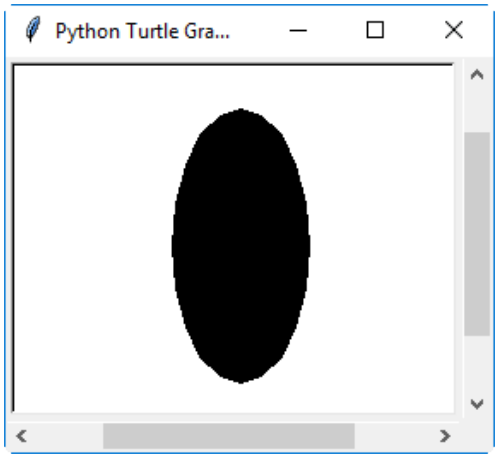
3. isvisible() ส่งกลับค่า True ถ้าอยู่ในสถานะแสดงตัวเต่า ส่งกลับค่า False ถ้าอยู่ในสถานะซ่อนตัวเต่า



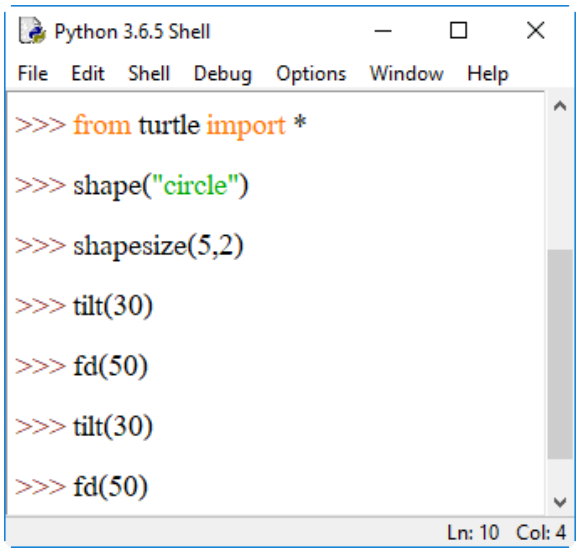
4. shape() เป็นการระบุรูปร่างของตัวเต่า โดยระบุรูปร่างเป็นสตริงอยู่ภายในวงเล็บ () ได้แก่ "arrow", "turtle", "circle", "square", "triangle", "classic" ลงไป เช่น shape("circle") หรือ shape("square") หากต้องการเปลี่ยนกลับเป็นรูปลูกศรให้ใช้ shape("classic")

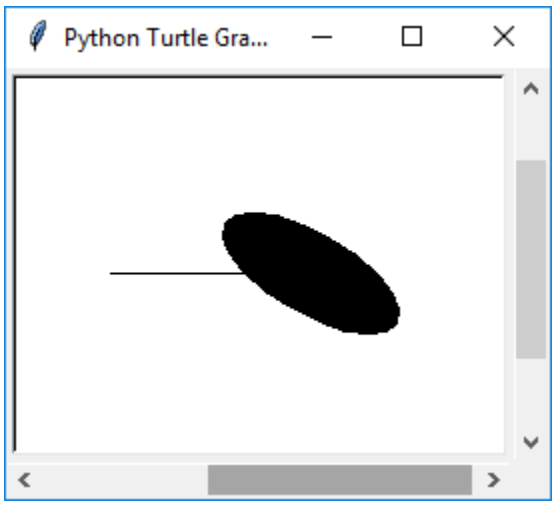
5. shapesize(กว้าง, ยาว) / turtlesize(กว้าง, ยาว) กำหนดขนาดรูปร่าง





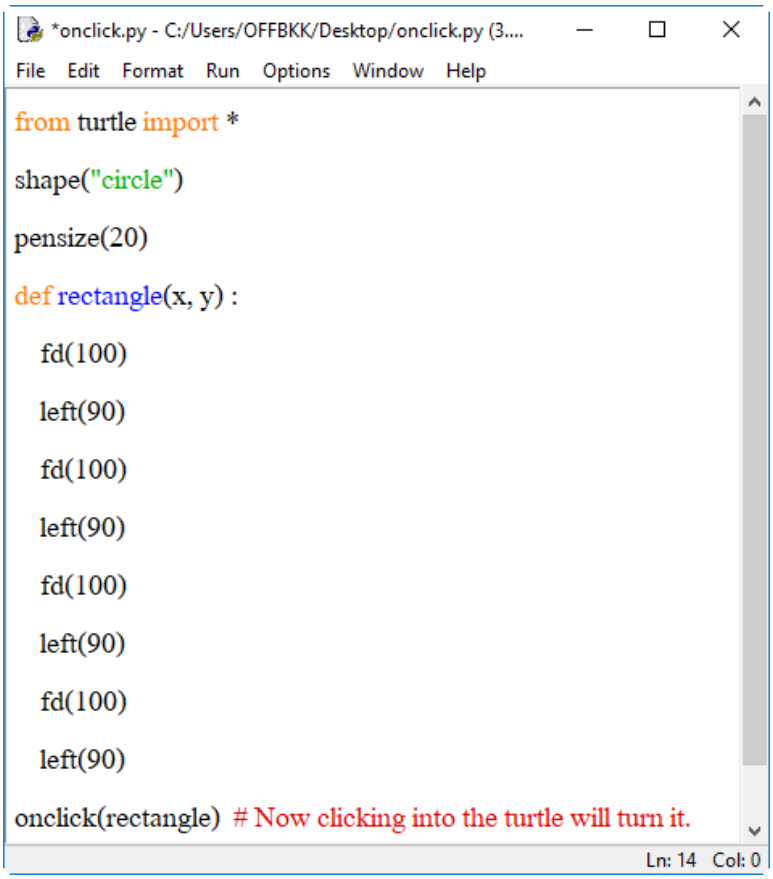
6. tilt(angle) หมุนรูปร่างของเต่าให้เอียงจากมุมของรูปร่างเดิมของเต่า แต่ไม่เปลี่ยนทิศทางของหัวเต่า

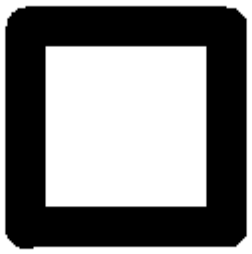




การกำหนดเหตุการณ์ (Event) ให้เต่าไพทอน

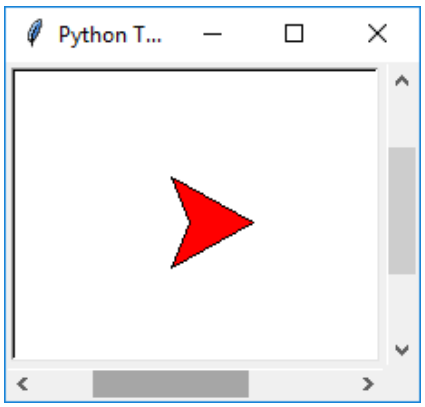
1. onclick() กำหนดเหตุการณ์เมื่อมีการคลิกเมาส์ที่ตัวเต่า โดยมีการส่งค่าพิกัด x, y ของจุดที่คลิกบนตัวเต่า เช่น เมื่อคลิกที่ตัวเต่าให้โปรแกรมเรียกฟังก์ชัน rectangle



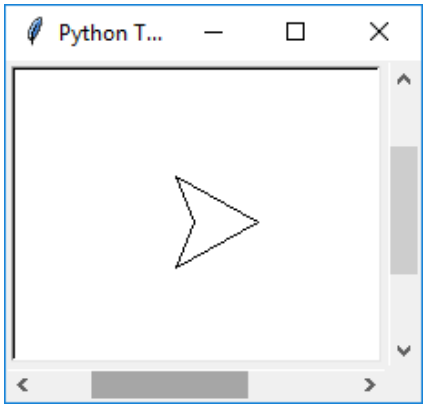


2. onrelease() กำหนดเหตุการณ์เมื่อมีการปล่อยเมาส์ที่ตัวเต่า

```
*onrelease.py - C:/Users/OFFBKK/Desktop/onrelease.py (3....
File  Edit  Format  Run  Options  Window  Help
from turtle import *
shapesize(5, 5)
def glow(x,y):
    fillcolor("red")
def unglow(x,y):
    fillcolor("")

onclick(glow)       # ถ้าคลิกเมาส์ที่ตัวเต่า จะเติมตัวเต่าเป็นสีแดง
onrelease(unglow)  # ถ้าปล่อยเมาส์ที่ตัวเต่า เต่าจะไม่มีสี
Ln: 10  Col: 0
```



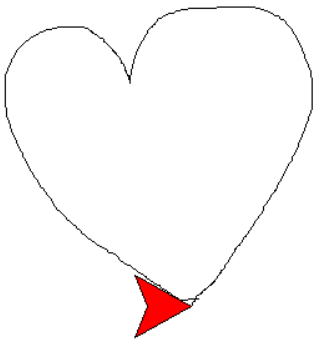
เมื่อคลิกเมาส์ที่ตัวเต่า



เมื่อปล่อยเมาส์

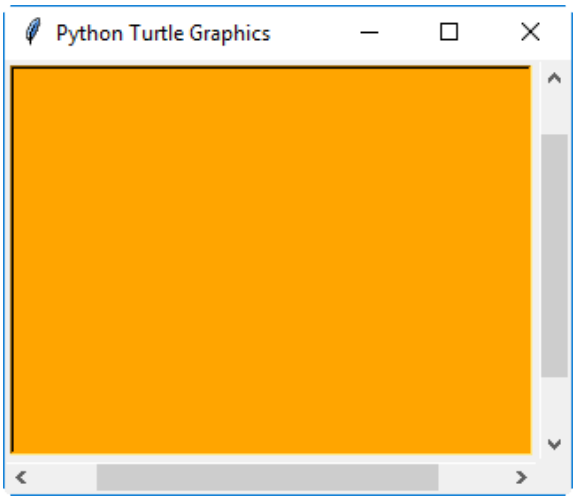
3. ondrag() กำหนดเหตุการณ์เมื่อมีการคลิกค้างที่ตัวเต่าแล้วลาก ตัวอย่างเช่น คำสั่ง ondrag(goto) เมื่อคลิกค้างที่เต่าแล้วลาก จะเป็นการวาดเส้นตามตำแหน่งของเต่าที่เคลื่อนที่ไป

```
*onrelease.py - C:/Users/OFFBKK/Desktop/onrelease.py (3....
File  Edit  Format  Run  Options  Window  Help
from turtle import *
shapesize(5, 5)
def glow(x,y):
    fillcolor("red")
def unglow(x,y):
    fillcolor("")

onclick(glow)       # ถ้าคลิกเมาส์ที่ตัวเต่า จะเติมตัวเต่าเป็นสีแดง
onrelease(unglow)   # ถ้าปล่อยเมาส์ที่ตัวเต่า เต่าจะไม่มีสี
ondrag(goto)        # ถ้าคลิกค้างที่ตัวเต่าแล้วลาก เต่าจะวาดเส้น
                                                        Ln: 11  Col: 0
```



คำสั่งควบคุมหน้าต่าง (Window control)

1. bgcolor() เป็นคำสั่งกำหนดสีภาพพื้นหลัง (ชื่อสีในภาษาอังกฤษ) เช่น bgcolor("orange")



2. clear() / clearscreen() ลบภาพวาดทั้งหมดและเต่าทั้งหมดออกจากหน้าจอ รีเซ็ตหน้าจอให้มีสถานะเริ่มต้น คือ พื้นหลังสีขาว

3. reset() / resetscreen() รีเซ็ตเต่าทั้งหมดบนหน้าจอให้อยู่ในสถานะเริ่มต้น

4. screensize() เป็นคำสั่งกำหนดขนาดให้กับหน้าต่าง มีหน่วยเป็นพิกเซล เช่น screensize(2000,1500)

5. title() เป็นคำสั่งกำหนดหัวข้อ title ของหน้าต่าง เช่น title("Rectangle")

6. mainloop() เป็นคำสั่งลูปค้างหน้าจอไว้สำหรับรอวาดรูปต่อ

การนำคำสั่ง for มาใช้ในการวาดรูป

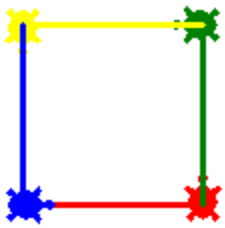
ตัวอย่างที่ 1

loop.py - C:/Users/OFFBKK/Desktop/lo...

File Edit Format Run Options Window Help

```
from turtle import *
title("square")
shape("turtle")
pensize(3)
speed(2)

for c in ['red', 'green', 'yellow', 'blue'] :
    color(c)          #สีเส้น
    forward(100)      #เดินหน้า 100
    left(90)          #หมุนไปทางซ้าย 90 องศา
    stamp()           #ปั๊มรูปเต่า

mainloop()
```

Ln: 14 Col: 0



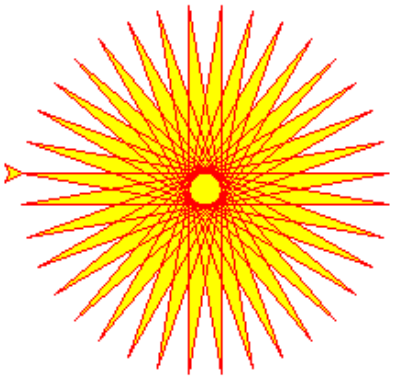
ตัวอย่างที่ 2

1.py - C:/Users/OFFBKK/Deskto...

File Edit Format Run Options Window Help

```
from turtle import *
color('red', 'yellow')
begin_fill()
for x in range(36) :
    forward(200)
    rt(170)
end_fill()
done()
```

Ln: 9 Col: 0



บทที่ 8 Python กับการเชื่อมต่อฐานข้อมูล

ภาษา Python มีช่องทางในการเชื่อมโยงกับฐานข้อมูล 2 แบบ คือ

1. การเชื่อมโยงกับฐานข้อมูลผ่านไลบรารีมาตรฐานประเภท DBM (Database Manager) เหมาะกับข้อมูลที่ไม่มีความซับซ้อนมาก เช่น ข้อมูลมีเพียง 1 ตาราง เป็นต้น

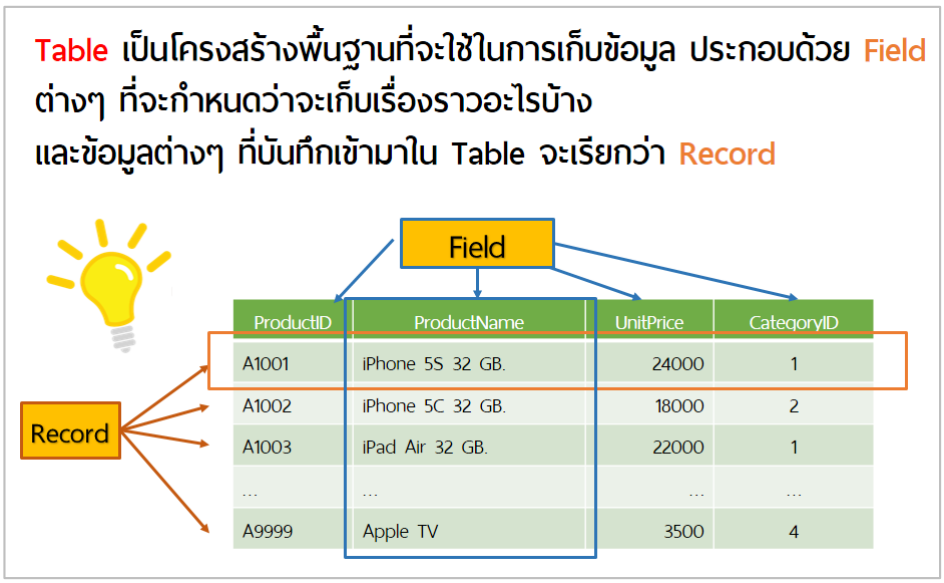
2. การเชื่อมโยงฐานข้อมูลผ่านไลบรารีมาตรฐาน DB-API 2.0 เหมาะกับฐานข้อมูลประเภท RDBMS (Relational Database Management System) เช่น Oracle, MySQL หรือ MS SQL Server เป็นต้น

โครงสร้างฐานข้อมูล

1. ตาราง (Table) สำหรับจัดเก็บกลุ่มข้อมูล เช่น ตารางลูกค้า ภายในตารางจะเป็นข้อมูลของลูกค้าทั้งหมด ตารางสินค้า จะเก็บกลุ่มรายละเอียดสินค้าแต่ละรายการไว้ในตารางสินค้าเท่านั้น

2. เรคอร์ด (Record) บางครั้งเรียกว่า Row รายละเอียดข้อมูลแต่ละรายการ เช่น เรคอร์ดของลูกค้าคนที่ 1 มีรายละเอียดดังนี้ คือ รหัส, ชื่อ, นามสกุล, ที่อยู่, เบอร์โทร, อีเมล์ หรือเรคอร์ดของสินค้าชิ้นที่ 1 จะมีรายละเอียดดังนี้ คือ รหัสสินค้า, ชื่อ, ยี่ห้อ, รุ่น, หมายเลขเครื่อง, สีสินค้า เป็นต้น

3. ฟิลด์ (Field) หรือบางครั้งเรียกว่า คอลัมน์ (Column) เป็นองค์ประกอบของเรคอร์ด เช่น เรคอร์ดของลูกค้าคนที่ 1 มีการเก็บรายละเอียด 6 รายการต่อการบันทึก 1 เรคอร์ด ภายในรายละเอียด 6 รายการดังกล่าว ประกอบด้วย รหัส, ชื่อ, นามสกุล, ที่อยู่, เบอร์โทร, อีเมล์ ซึ่งทุกรายการย่อยที่ยกตัวอย่างก็คือ Field ย่อยภายในเรคอร์ดนั่นเอง



ที่มาของภาพ : https://www.9experttraining.com

1. การเลือกใช้ฐานข้อมูลประเภท Flat File Database

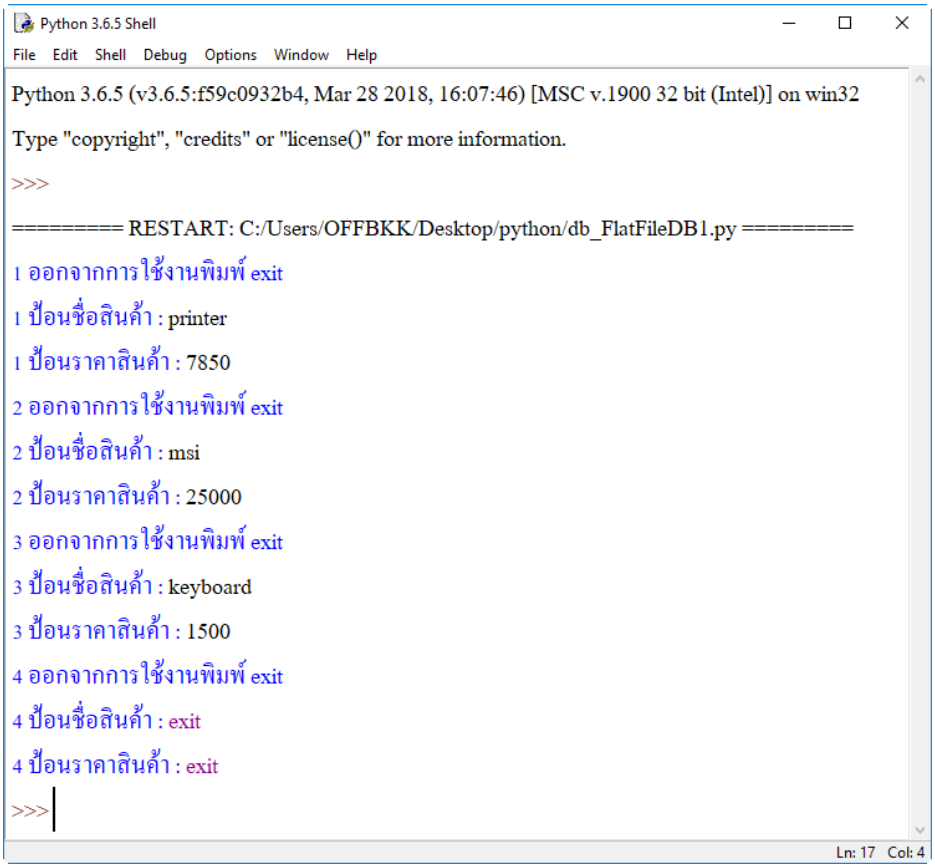
เป็นพื้นฐานการจัดเก็บข้อมูลเป็นไฟล์ Text หรือ Binary เหมาะกับข้อมูลขนาดเล็กจนถึงขนาดกลาง ไฟล์ชนิดนี้จะมีความจุที่ไม่จำกัด ขึ้นอยู่กับปริมาณของข้อมูลที่จัดเก็บ เมื่อเทียบกับอีก 2 ประเภท ไฟล์ฐานข้อมูลประเภทนี้จะมีความจุไฟล์น้อยที่สุด อีกทั้งยังง่ายต่อการสร้าง ค่าใช้จ่ายน้อย สามารถใช้งานร่วมกับโปรแกรม Spreadsheet ได้ แต่ข้อด้อยคือความเร็วในการเรียกใช้งานข้อมูล ยิ่งมีข้อมูลมาก ความเร็วในการค้นหาหรือดึงข้อมูลมาใช้ก็จะทำงานได้ช้า สำหรับภาษา Python สามารถใช้คำสั่งในการสร้าง อ่าน เขียนค่าข้อมูลที่ต้องการจัดเก็บในไฟล์ฐานข้อมูลประเภทนี้ได้

ตัวอย่างที่ 1 โปรแกรมจะทำการสร้างไฟล์ datatest.txt ขึ้นมาหน้า Desktop แล้วให้รับค่าการป้อนชื่อสินค้าและราคาสินค้าในหน้าต่าง Python Shell ไปเรื่อยๆ จนกว่าจะพิมพ์ exit ในช่องป้อนชื่อสินค้าและป้อนราคาสินค้า ข้อมูลชื่อสินค้าและราคาสินค้าจะถูกจัดเก็บในไฟล์ datatest.txt โดยอัตโนมัติ

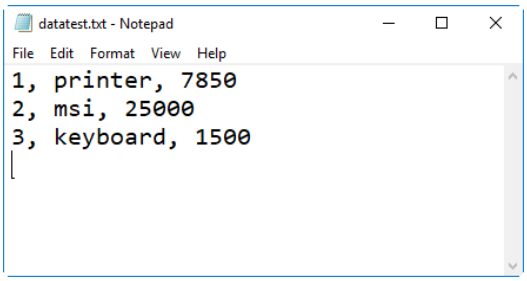
- เขียนคำสั่ง Python

```
lineno = 1
fdata = open('C:\\Users\OFFBKK\Desktop\datatest.txt', 'w')
while True :
   print('%d ออกจากการใช้งานพิมพ์ exit ' %lineno)
   title = input('%d ป้อนชื่อสินค้า : ' %lineno)
   cost = input('%d ป้อนราคาสินค้า : ' %lineno)
   if title == 'exit' or cost == 'exit' :
      break
   fdata.write('%d, %s, %s\n' %(lineno, title, cost))
   lineno = lineno + 1
fdata.close()
```

- ป้อนชื่อสินค้าและราคาสินค้าในหน้าต่าง Python Shell



- เปิดไฟล์ datatest.txt บนหน้า Desktop จะมีข้อมูลดังนี้



จากตัวอย่าง ให้ลองรันโปรแกรมอีกครั้ง เมื่อเราป้อนข้อมูลชื่อสินค้าและราคาสินค้าใหม่เข้าไป ข้อมูลใหม่นี้จะไปทับข้อมูลเดิม ทำให้ข้อมูลเดิมหายไป เพราะจากคำสั่งในบรรทัดที่ 2 คือ fdata = open('C:\\Users\OFFBKK\Desktop\datatest.txt', 'w') จะพบว่าเมื่อใช้อาร์กิวเมนต์ตัวอักษร 'w' จะทำให้ข้อมูลใหม่ไปทับข้อมูลเดิม หากต้องการเพิ่มข้อมูลเข้าไป โดยไม่ให้ข้อมูลเดิมสูญหาย ต้องเปลี่ยนจาก 'w' เป็น 'a'

ค่าคงที่สำหรับใช้งานกับอาร์กิวเมนต์	ลักษณะการทำงานร่วมกับคำสั่ง open
a	สร้างไฟล์หรือแก้ไขข้อมูลแบบเพิ่มเติม
w	สร้างไฟล์หรือแก้ไขข้อมูลแบบทับซ้ำ
r	อ่านข้อมูลในไฟล์ที่กำลังเปิดใช้งาน

จากตัวอย่างที่ 1 ประกาศตัวแปร lineno เพื่อใช้เก็บจำนวนครั้งในการป้อนข้อมูล โดยมีค่าเริ่มต้นเท่ากับ 1 และจะเพิ่มขึ้นทีละหนึ่งเพื่อรับข้อมูลชื่อสินค้าและราคาสินค้าชุดใหม่ โปรแกรมจะเชื่อมต่อกับไฟล์ datatest.txt ด้วยคำสั่ง open และสิ้นสุดการเชื่อมต่อด้วยคำสั่ง close ในการวนซ้ำเพื่อรับค่าการป้อนข้อมูลของสินค้าแต่ละรอบจะใช้คำสั่ง while True ซึ่งในแต่ละครั้งของการวนลูปจะเขียนข้อมูลลงในไฟล์ datatest.txt ด้วยคำสั่ง write

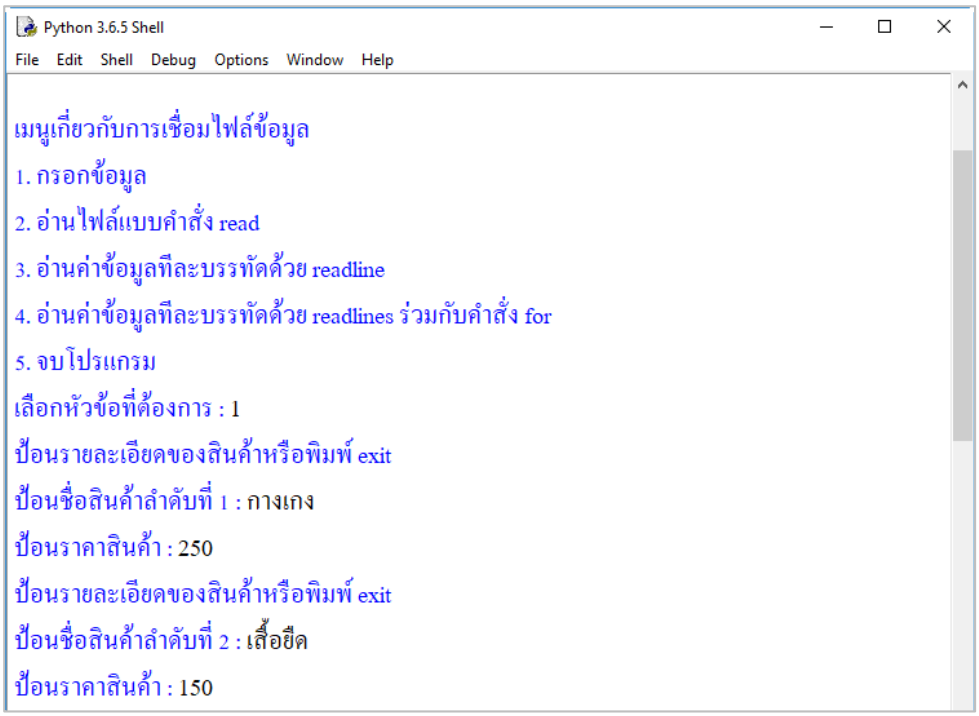
ตัวอย่างที่ 2 เขียนโปรแกรมรวมคำสั่งเขียนไฟล์ และอ่านค่าข้อมูลแบบ read, readline และ readlines

```
import os
datafilename = 'C:\\Users\OFFBKK\Desktop\productdata.txt'

###################ฟังก์ชันเขียนไฟล์#####################################
def writefile(filename) :
    lineno = 1
    if os.path.isfile(filename) :
        f = open(filename, 'a')
    else :
        f = open(filename, 'w')
    while True :
        print('ป้อนรายละเอียดของสินค้าหรือพิมพ์ exit')
        title = input('ป้อนชื่อสินค้าลำดับที่ %d : ' %lineno)
        cost = input('ป้อนราคาสินค้า : ')
        if title == 'exit' or cost == 'exit' :
            break
        f.write('%d, %s, %s\n' %(lineno, title, cost))
        lineno += 1
    f.close()
```

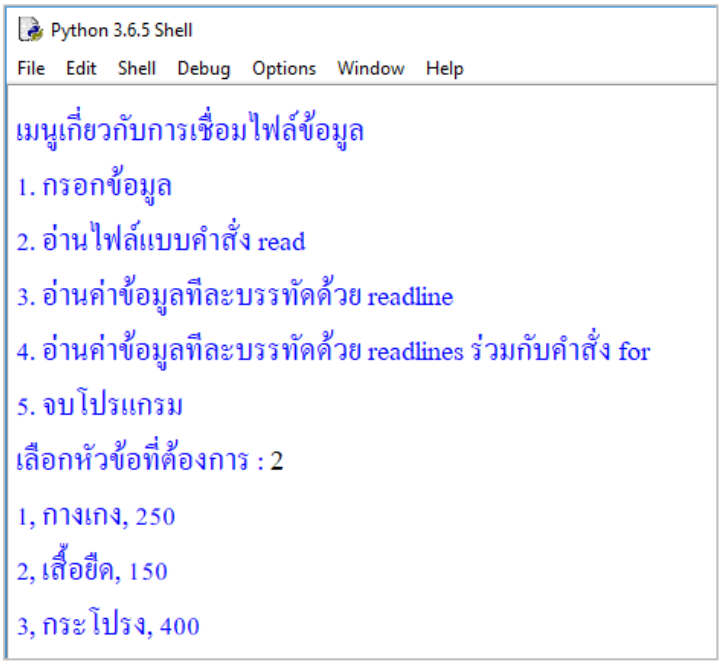
```
##################ฟังก์ชันอ่านค่าข้อมูลทั้งหมดในไฟล์###################################
def readallfile(filename) :
    f = open(filename,'r')
    data = f.read()
    print(data)
    f.close()

  ##################ฟังก์ชันอ่านค่าข้อมูลบรรทัดแรกในไฟล์###################################
def readonelinefile(filename) :
    f = open(filename,'r')
    data = f.readline()
    print(data)
    f.close()

  ##################ฟังก์ชันอ่านค่าข้อมูลทั้งหมดทีละชุดในไฟล์###################################
def readlineloopfile(filename) :
    f = open(filename,'r')
    data = f.readlines()
    for x in data :
       print(x)
    f.close()
```

```
###################เริ่มต้นโปรแกรม#####################################
while True :
   print('\nเมนูเกี่ยวกับการเชื่อมไฟล์ข้อมูล')
   print('''1. กรอกข้อมูล \n2. อ่านไฟล์แบบคำสั่ง read
3. อ่านค่าข้อมูลทีละบรรทัดด้วย readline
4. อ่านค่าข้อมูลทีละบรรทัดด้วย readlines ร่วมกับคำสั่ง for
5. จบโปรแกรม''')
   item = input('เลือกหัวข้อที่ต้องการ : ')
   if item == '1' :
      writefile(datafilename)
   elif item == '2' :
      readallfile(datafilename)
   elif item == '3' :
      readonelinefile(datafilename)
   elif item == '4' :
      readlineloopfile(datafilename)
   else :
      break
```

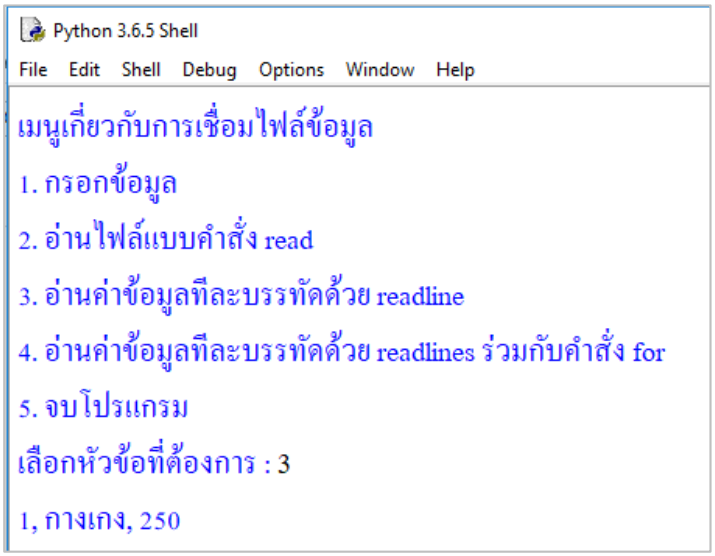
เมื่อพิมพ์ 1 จะทำตามคำสั่งฟังก์ชัน writefile() จนกว่าจะป้อนคำว่า exit ก็จะออกจากฟังก์ชัน



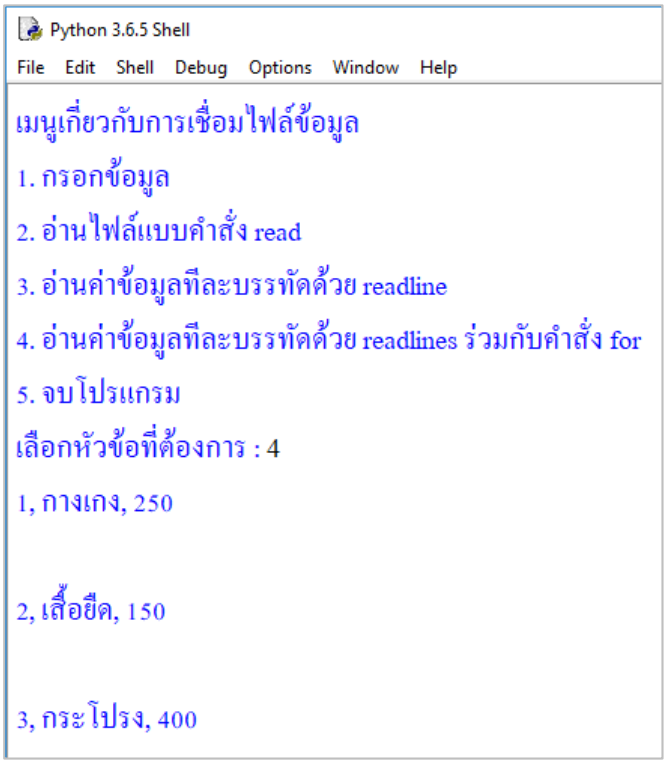
เมื่อพิมพ์ 2 จะทำตามคำสั่งฟังก์ชัน readallfile()



เมื่อพิมพ์ 3 จะทำตามคำสั่งฟังก์ชัน readonelinefile()



เมื่อพิมพ์ 4 จะทำตามคำสั่งฟังก์ชัน readlineloopfile ()

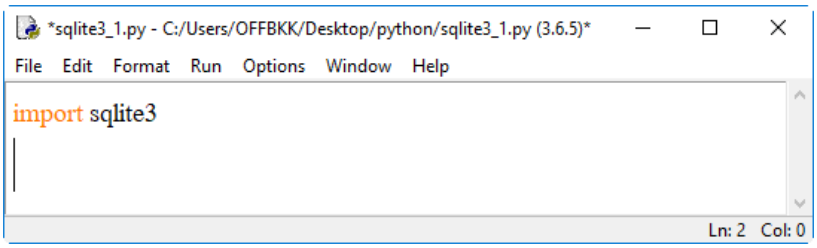


2. ฐานข้อมูลประเภท File Server-Based Database

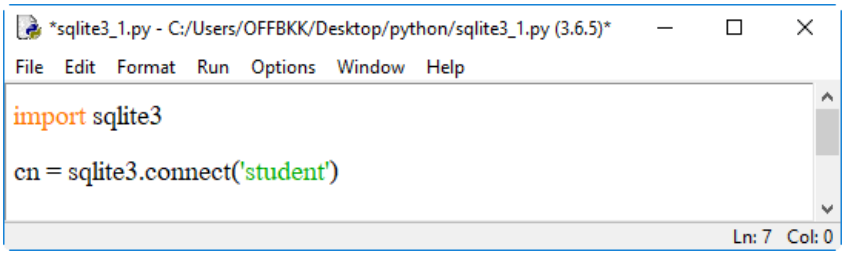
ภาษา Python มีโมดูลที่มีความสามารถในการจัดสร้างไฟล์ฐานข้อมูลแบบ RDBMS (Relational Database Management System) ที่มีขนาดของไฟล์เหมาะกับการจัดเก็บข้อมูลระดับเล็กถึงระดับกลาง สำหรับโมดูลของ Python ที่ใช้ดูแลไฟล์ฐานข้อมูลนี้ คือ SQLite3 ซึ่งถูกติดตั้งพร้อมกับโปรแกรม Python แล้ว โดยรองรับการใช้คำสั่ง select, insert, delete, update, create, drop, alter เป็นต้น

การใช้งาน SQLite3 แบ่งออกเป็นลำดับขั้นตอนดังนี้ คือ

2.1 การอ้างอิงโมดูลของ SQLite3



2.2 สร้างไฟล์ฐานข้อมูล มีรูปแบบการเขียน ดังนี้

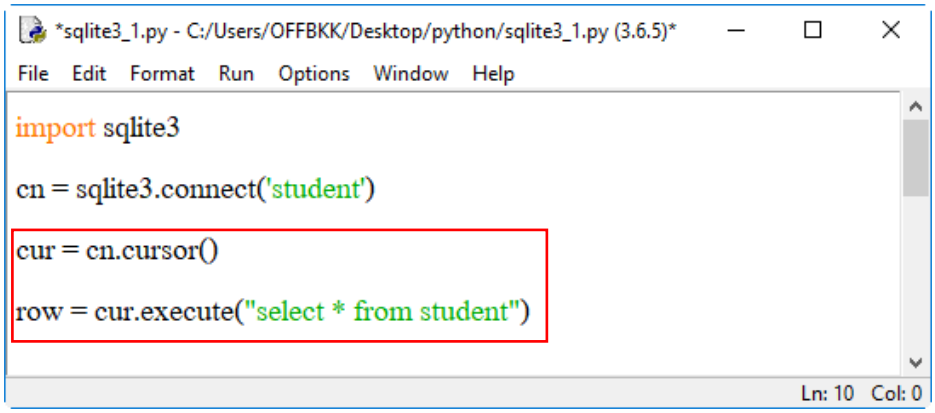


โดย cn เป็นตัวแปรของ Connect Object

sqlite3 เป็นประเภทของฐานข้อมูลที่ใช้งาน ในที่นี้คือโมดูล SQLite3

student เป็นที่อยู่ของไฟล์ฐานข้อมูล ซึ่งจะต้อง path ที่อยู่ของไฟล์ให้ถูกต้อง

2.3 การใช้ Cursor Object เมื่อเชื่อมต่อกับฐานข้อมูลได้แล้ว ต่อมาจะต้องใช้ Cursor Object เพื่อเข้าถึงตารางและข้อมูลในตาราง โดยต้องเขียนคำสั่ง cur = cn.cursor() เพื่อเชื่อม Cursor เข้ากับ Cursor Object



2.3.1 การใช้ Cursor Object กับการอ่านค่าข้อมูลด้วยคำสั่ง Select คำสั่งที่ใช้ในการอ่านค่าข้อมูลต่างๆ จากตาราง คือ คำสั่ง select โดยมีรูปแบบการเขียน ดังนี้ คือ

รูปแบบการใช้ select สำหรับอ่านเฉพาะ Field ที่ระบุ

รูปแบบ : select ชื่อฟิลด์ที่ 1, ชื่อฟิลด์ที่ 2 from ชื่อตาราง

ตัวอย่าง : select id, name, lastname from student

รูปแบบการใช้ select สำหรับอ่านทุกฟิลด์ในตาราง

รูปแบบ : select * from ชื่อตาราง

ตัวอย่าง : select * from student

รูปแบบการใช้ select พร้อมเงื่อนไขที่ต้องการอ่านเรคอร์ด

รูปแบบ : select * from ชื่อตาราง where เงื่อนไข

ตัวอย่าง : select * from student where age > 15

รูปแบบการใช้ select ร่วมกับการเรียงข้อมูล (Sort)

รูปแบบ : select * from ชื่อตาราง order by ชื่อฟิลด์

ตัวอย่าง (น้อยไปมาก) : select * from student order by age

ตัวอย่าง (มากไปน้อย) : select * from student order by age desc

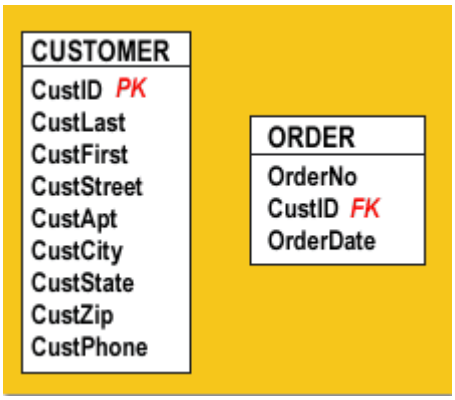
รูปแบบการใช้ select พร้อมเงื่อนไขและการเรียงข้อมูล

รูปแบบ : select * from ชื่อตาราง where เงื่อนไข order by ชื่อฟิลด์

ตัวอย่าง : select * from student where age > 15 order by age

รูปแบบการใช้ select กับการเชื่อมระหว่างตาราง รูปแบบคำสั่ง จะเขียนได้ดังนี้

select ชื่อตารางที่ 1.ชื่อฟิลด์, ชื่อตารางที่ 2.ชื่อฟิลด์
from ชื่อตารางที่ 1, ชื่อตารางที่ 2
where ชื่อตารางที่ 1.ชื่อฟิลด์ (คีย์รองหรือ FK) = ชื่อตารางที่ 2.ชื่อฟิลด์ (คีย์หลัก หรือ PK)



ตัวอย่าง : select ORDER.OrderNo, ORDER.OrderDate, CUSTOMER.CustFirst,

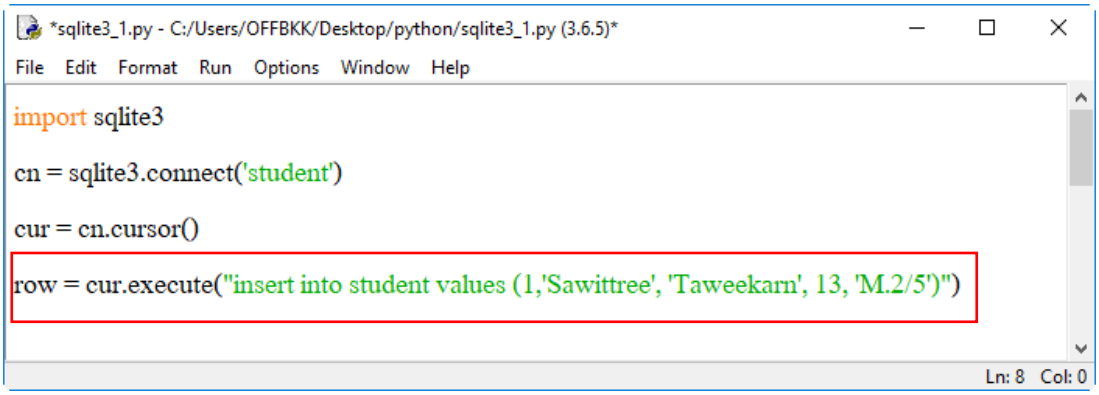
CUSTOMER.CustLast, CUSTOMER.CustPhone

from ORDER, CUSTOMER

where ORDER.CustID = CUSTOMER. CustID

2.3.2 การใช้ Cursor Object กับการเพิ่มข้อมูลใหม่ในตารางด้วยคำสั่ง Insert

รูปแบบการเขียนคำสั่ง insert สำหรับเพิ่มข้อมูลใหม่



รูปแบบการใช้คำสั่ง insert สำหรับเพิ่มข้อมูลลงทุกฟิลด์ของตาราง

รูปแบบ : insert into ชื่อตาราง values (ค่าข้อมูลที่ 1, ค่าข้อมูลที่ 2, ..., ค่าข้อมูลที่ n)

ตัวอย่าง : insert into student values (1,'Sawittree', 'Taweekarn', 13, 'M.2/5')

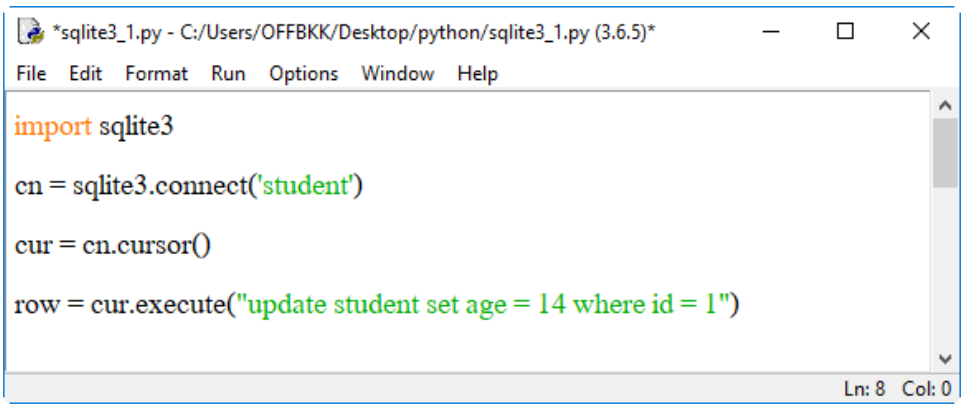
รูปแบบการใช้คำสั่ง insert สำหรับเพิ่มข้อมูลบางฟิลด์ของตาราง

รูปแบบ : insert into ชื่อตาราง(ชื่อฟิลด์ที่ 1, ชื่อฟิลด์ที่ 2)values(ค่าข้อมูลที่ 1, ค่าข้อมูลที่ 2)

ตัวอย่าง : insert into student (id, name, lastname) values (2,'Tawee', 'Meesuk')

2.3.3 การใช้ Cursor Object กับการแก้ไขข้อมูลในตารางด้วยคำสั่ง Update

รูปแบบการเขียนคำสั่ง update สำหรับแก้ไขข้อมูลในตาราง



รูปแบบการใช้คำสั่ง update สำหรับแก้ไขข้อมูลในฟิลด์ที่ระบุทุกเรคอร์ดในตาราง

รูปแบบ : update ชื่อตาราง set ชื่อฟิลด์ที่ 1 = ค่าข้อมูล, ชื่อฟิลด์ที่ 2 = ค่าข้อมูล

ตัวอย่าง : update student set age = age + 1

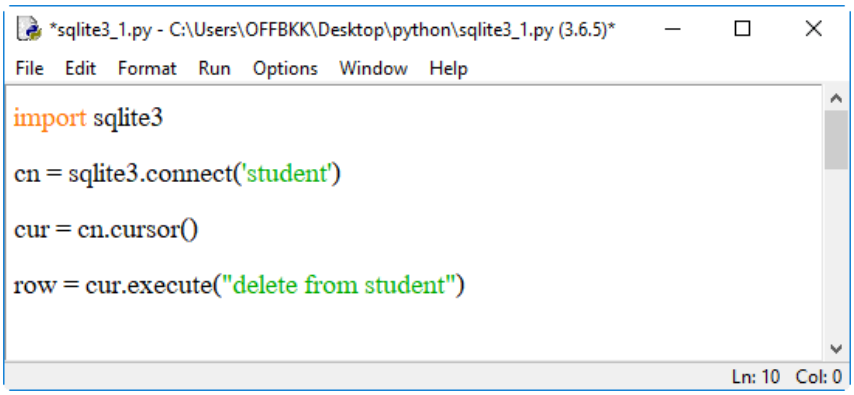
จากตัวอย่างจะเป็นการเพิ่มอายุนักเรียนทุกคนในตาราง โดยเพิ่มขึ้นคนละ 1 ปี

รูปแบบการใช้คำสั่ง update โดยใช้ร่วมกับคำสั่ง where เพื่อแก้ไขข้อมูลบางเรคอร์ด

รูปแบบ : update ชื่อตาราง set ชื่อฟิลด์ที่ 1 = ค่าข้อมูล, ชื่อฟิลด์ที่ 2 = ค่าข้อมูล
Where ชื่อฟิลด์ 1 = ค่าข้อมูล

ตัวอย่าง : update student set classroom = 'M.3/2' age = 14 where id = 1

2.3.4 การใช้ Cursor Object กับการลบเรคอร์ดข้อมูลจากตารางด้วยคำสั่ง Delete



รูปแบบการใช้คำสั่ง delete สำหรับลบเรคอร์ดทั้งหมดในตาราง

รูปแบบ : delete from ชื่อตาราง

ตัวอย่าง : delete from student

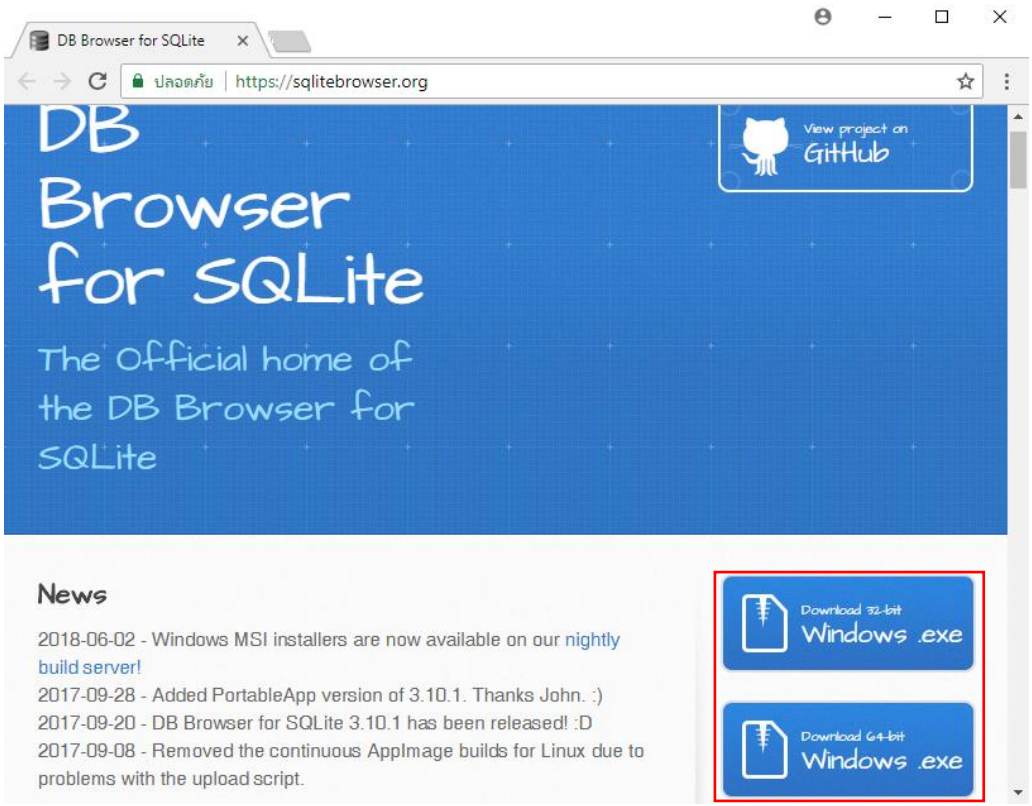
รูปแบบการใช้คำสั่ง delete ร่วมกับคำสั่ง where สำหรับลบบางเรคอร์ดในตาราง

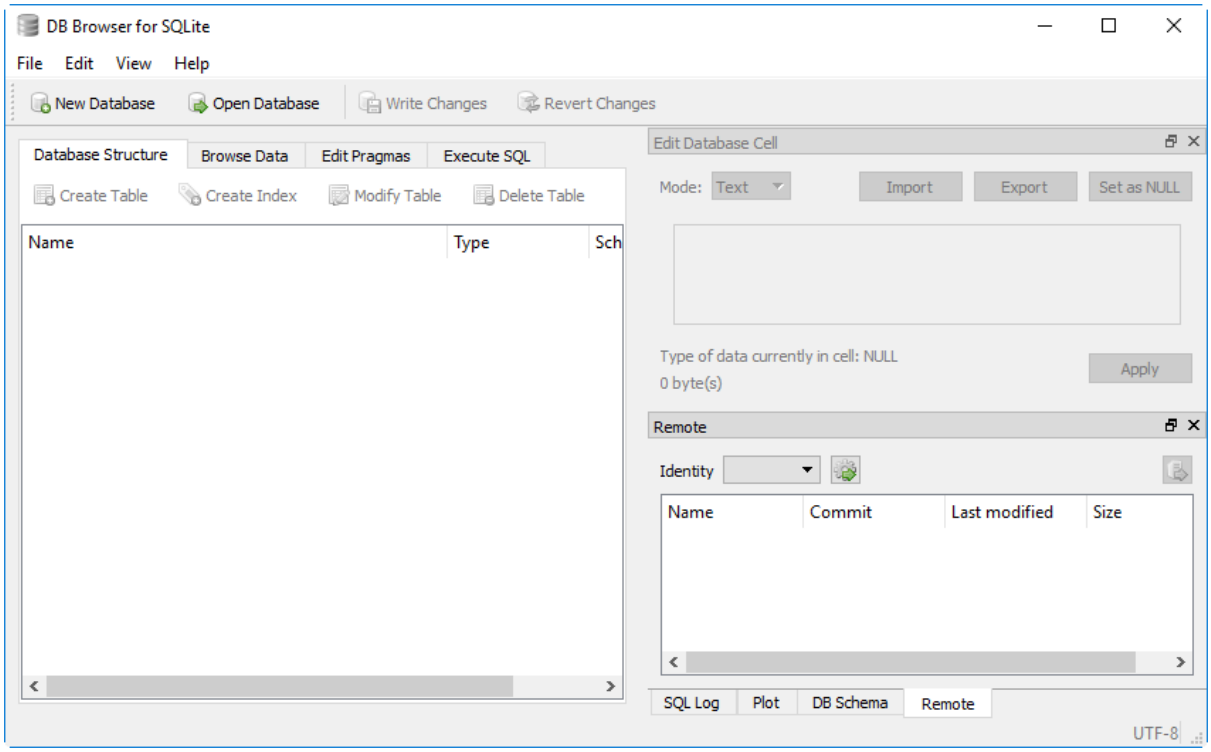
รูปแบบ : delete from ชื่อตาราง where ชื่อฟิลด์ = ค่าข้อมูล

ตัวอย่าง : delete from student where id = 1

2.4 การสร้างตารางและความสัมพันธ์ของตาราง (Relationship)

การสร้างตารางในไฟล์ฐานข้อมูล มี 2 วิธี คือ การเขียนโค้ดคำสั่งสร้างตาราง และการใช้โปรแกรมสำเร็จรูปสร้างตาราง โดยให้ทำการดาวน์โหลดโปรแกรม DB Browser for SQLite จากเว็บไซต์ https://sqlitebrowser.org/ และทำการติดตั้งโปรแกรมเพื่อใช้จัดการฐานข้อมูล ดังนี้





รูปแบบการเขียนคำสั่ง create เพื่อสร้างตาราง :

create table ชื่อตาราง (

ชื่อฟิลด์ที่ 1 ชนิดข้อมูล,

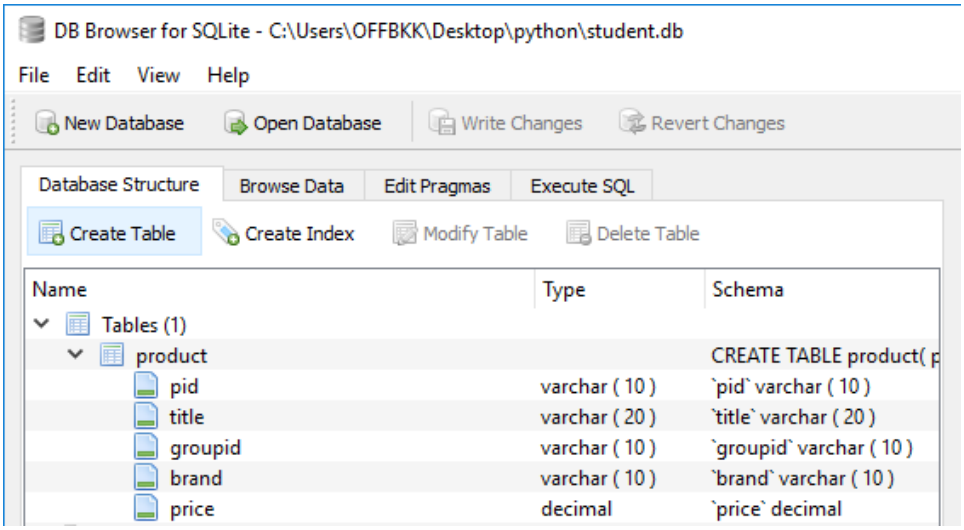
ชื่อฟิลด์ที่ 2 ชนิดข้อมูล,

ชื่อฟิลด์สุดท้าย ชนิดข้อมูล)

ตัวอย่าง :

```
import sqlite3
cn = sqlite3.connect('product.db')
cur = cn.cursor()
cur.execute('''
   create table product(
   pid varchar(10),
   title varchar(20),
   groupid varchar(10),
   brand varchar(10),
   price decimal)''')
```

เปิดโปรแกรม DB Browser for SQLite ขึ้นมา แล้วคลิกที่ปุ่ม open Database เปิดไฟล์ฐานข้อมูล product.db ที่สร้างไว้ขึ้นมา เพื่อดูตารางที่สร้างขึ้นจากการเขียนโค้ดคำสั่ง



รูปแบบการเขียนคำสั่ง create เพื่อสร้างตาราง โดยระบุ Primary Key:

create table ชื่อตาราง (

ชื่อฟิลด์ที่ 1 ชนิดข้อมูล,

ชื่อฟิลด์ที่ 2 ชนิดข้อมูล,

ชื่อฟิลด์สุดท้าย ชนิดข้อมูล,

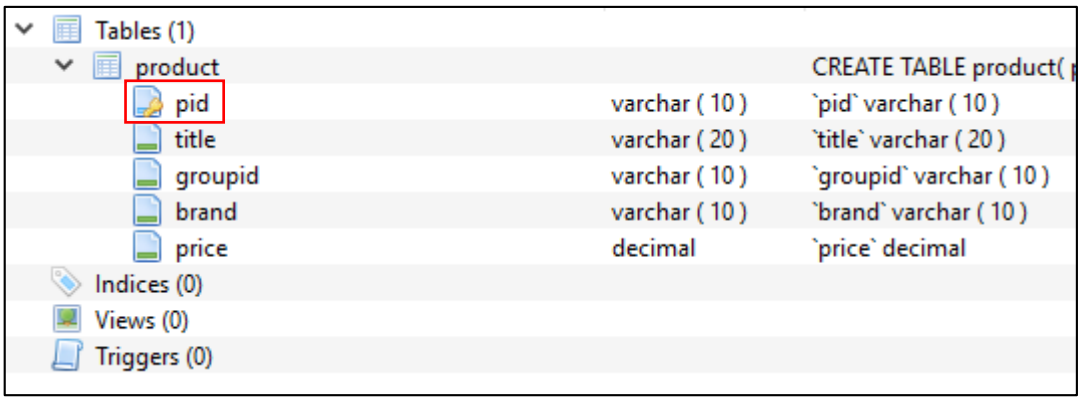
constraint[]

primary key (ชื่อฟิลด์ที่เป็นคีย์หลัก PK))

ตัวอย่าง :

```
import sqlite3
cn = sqlite3.connect('product.db')
cur = cn.cursor()
cur.execute('''
   create table product(
   pid varchar(10),
   title varchar(20),
   groupid varchar(10),
   brand varchar(10),
   price decimal,
   constraint[]
   primary key (pid))
   ''')
```

เปิดโปรแกรม DB Browser for SQLite ขึ้นมา แล้วคลิกที่ปุ่ม open Database เปิดไฟล์ฐานข้อมูล student.db ที่สร้างไว้ขึ้นมา เพื่อดูตารางที่สร้างขึ้นจากการเขียนโค้ดคำสั่ง



รูปแบบการเขียนคำสั่ง create เพื่อสร้างตารางที่มีความสัมพันธ์ (Relationship) กับตารางอื่น :

create table ชื่อตาราง (

ชื่อฟิลด์ที่ 1 ชนิดข้อมูล,

ชื่อฟิลด์ที่ 2 ชนิดข้อมูล,

ชื่อฟิลด์ที่ 3 ชนิดข้อมูล, constraint [ชื่อฟิลด์ที่เป็น Foreign Key] references

ชื่อตารางที่อ้างอิง Primary Key(ฟิลด์ที่เป็น Primary Key)

on update cascade

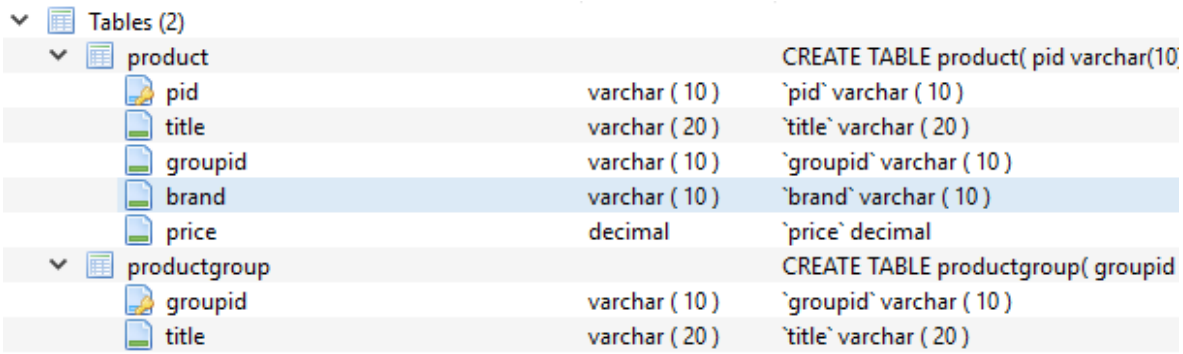
on delete cascade

)

ตัวอย่าง :

```
import sqlite3
cn = sqlite3.connect('product.db')
cur = cn.cursor()
cur.execute('''
   create table productgroup(
   groupid varchar(10),
   title varchar(20),
   constraint[]
   primary key (groupid))''')
cur.execute('''
   create table product(
   pid varchar(10),
   title varchar(20),
   groupid varchar(10) constraint[fkgroupid] references productgroup(groupid)
   on update cascade,
   brand varchar(10),
   price decimal,
   constraint[]
   primary key (pid))''')
```

เมื่อรันโปรแกรม จะได้ตาราง 2 ตาราง คือ product และ productgroup ดังรูป



2.5 เพิ่มเรคอร์ดข้อมูลลงในตาราง ด้วยคำสั่ง insert ของ SQL Statement

รูปแบบการใช้คำสั่ง insert เพิ่มเรคอร์ดในตาราง

insert into ชื่อตาราง values (ค่าข้อมูล, ค่าข้อมูล, ค่าข้อมูล)

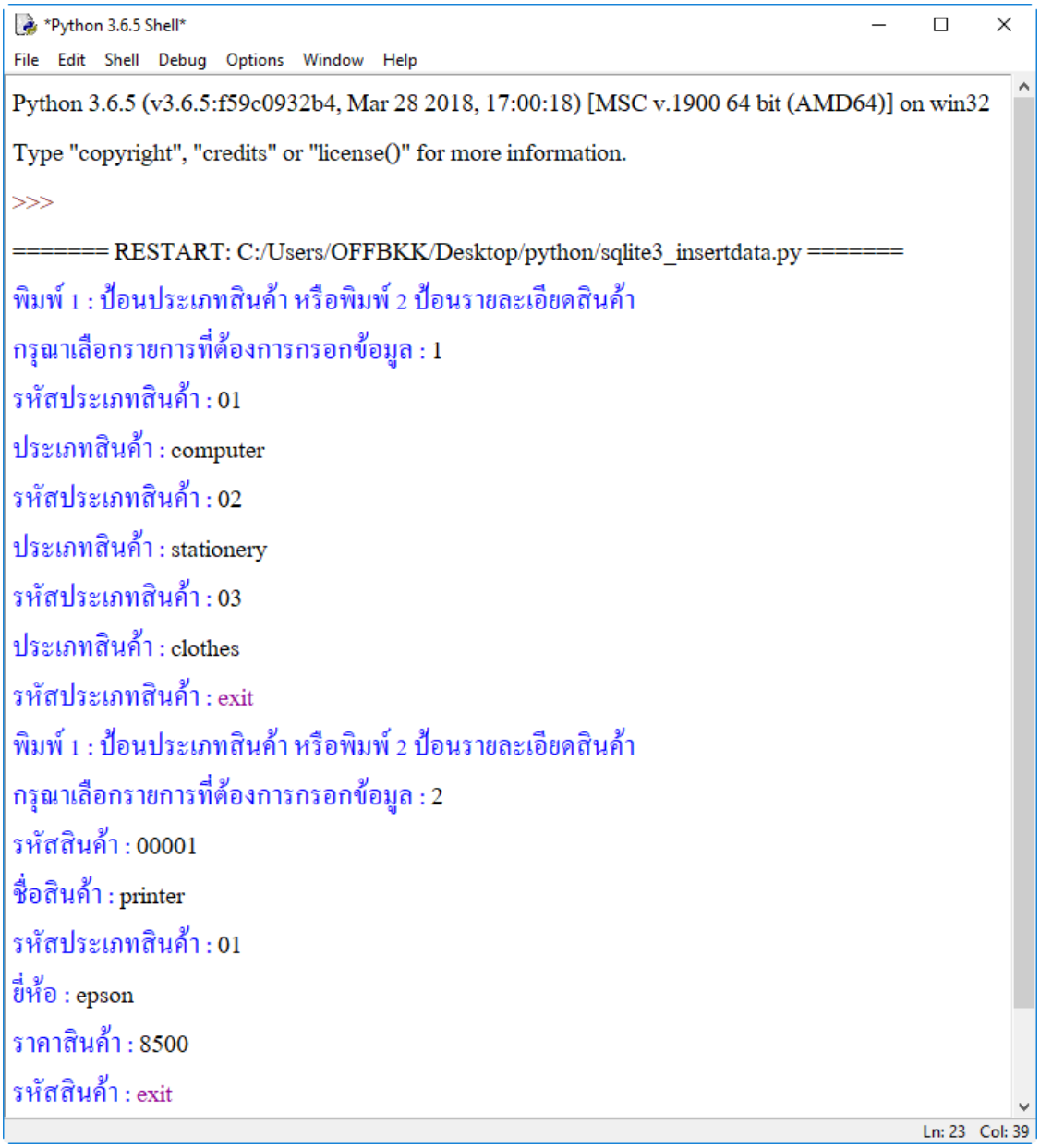
ตัวอย่าง :

```
import sqlite3
cn = sqlite3.connect('product.db')
cur = cn.cursor()

def productinput() :
  while True :
    a = input("รหัสสินค้า : ")
    if a == 'exit' :
      break
    b = input("ชื่อสินค้า : ")
    c = input("รหัสประเภทสินค้า : ")
    d = input("ยี่ห้อ : ")
    e = input("ราคาสินค้า : ")
    cur.execute("insert into product values(?,?,?,?,?)",(a,b,c,d,e))
    cn.commit()
```

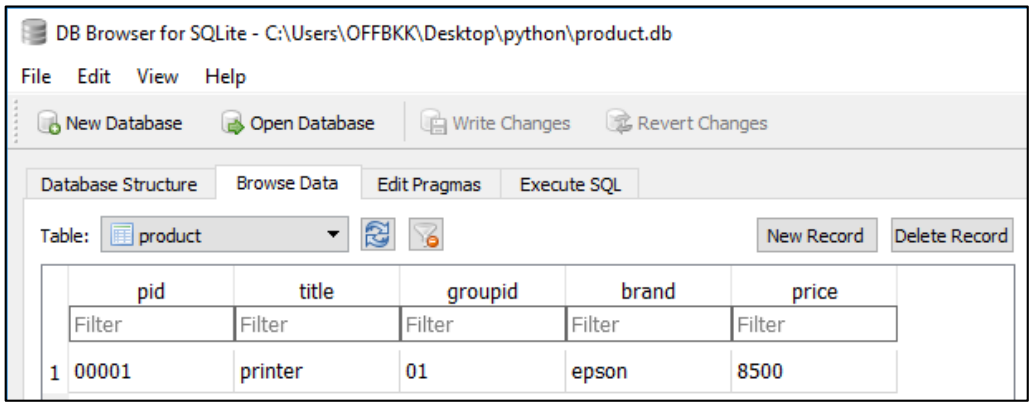
```
def groupinput() :
    while True :
        f = input("รหัสประเภทสินค้า : ")
        if f == 'exit' :
            break
        g = input("ประเภทสินค้า : ")
        cur.execute("insert into productgroup values(?,?)",(f,g))
        cn.commit()
while True :
    print("พิมพ์ 1 : ป้อนประเภทสินค้า หรือพิมพ์ 2 ป้อนรายละเอียดสินค้า")
    item = input("กรุณาเลือกรายการที่ต้องการกรอกข้อมูล : ")
    if item == '1' :
        groupinput()
    elif item == '2' :
        productinput()
    else :
        break
cn.commit()
cur.close()
cn.close()
```

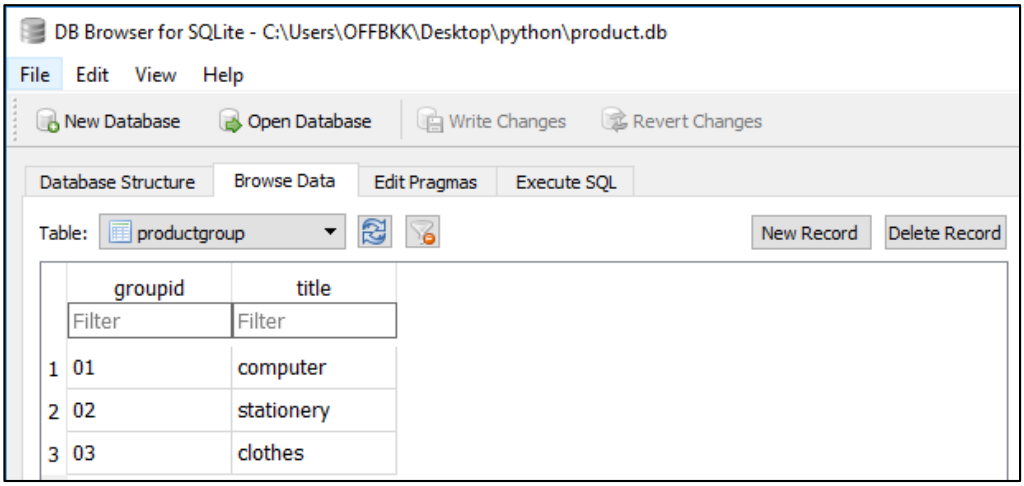
จากตัวอย่าง มีการสร้างฟังก์ชัน productinput() ขึ้นมาเพื่อรับค่าข้อมูลใส่ในตาราง product (สร้างเอาไว้ก่อนแล้วในขั้นตอนการสร้างตาราง) โดยรับค่าข้อมูลทางคีย์บอร์ดด้วยคำสั่ง input() แล้วบันทึกไว้ในตัวแปร a, b, c, d, e ตามลำดับ (ตามจำนวนฟิลด์ในตาราง) หลังจากนั้นใช้คำสั่ง cur.execute("insert into product values(?,?,?,?,?)",(a,b,c,d,e)) โดยใช้เครื่องหมาย ? อ้างอิงค่าข้อมูลของตัวแปรที่จะนำไปใส่ในแต่ละฟิลด์ของตาราง product

เมื่อรันโปรแกรม จะเข้าสู่หน้าจอ Python Shell ให้เราทำการกรอกข้อมูล ดังรูป



เมื่อเปิดโปรแกรม DB Browser for SQLite ก็จะได้เรคอร์ดข้อมูลในตาราง ดังรูป





2.6 อ่านค่าข้อมูลจากตารางด้วยคำสั่ง select มีหลายรูปแบบ ดังนี้

รูปแบบที่ 1 การอ่านเรคอร์ดระหว่างคำสั่ง cursor กับตัวแปร Python แบบส่งผ่านค่าข้อมูลไปยังตัวแปรโดยตรง

รูปแบบ : cur.execute("select * from ชื่อตาราง")

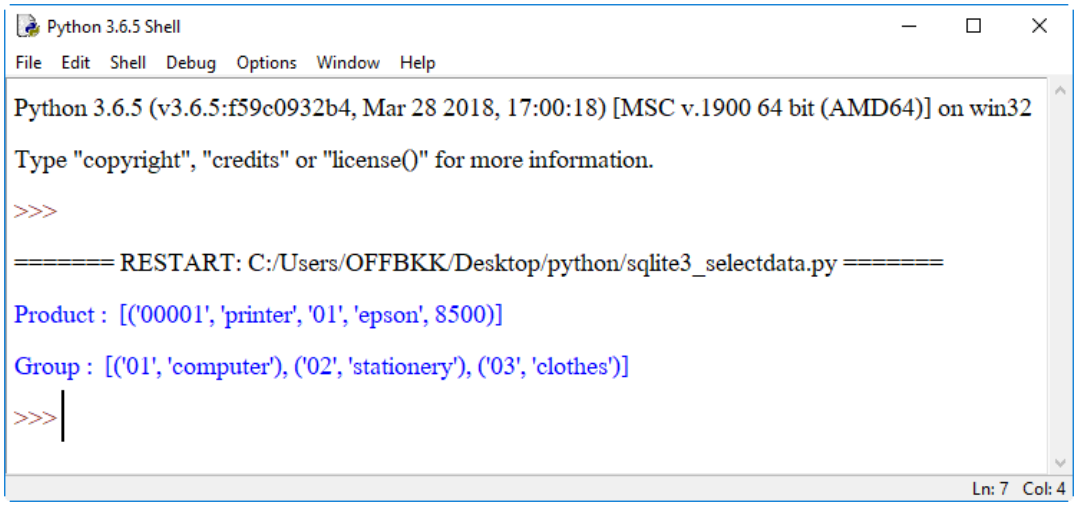
ชื่อตัวแปร = cur.fetchall()

print(ชื่อตัวแปร)

ตัวอย่าง :

```
import sqlite3
cn = sqlite3.connect('product.db')
cur = cn.cursor()

cur.execute("select * from product")
showproduct = cur.fetchall()
print("Product : ",showproduct)

cur.execute("select * from productgroup")
showgroup = cur.fetchall()
print("Group : ",showgroup)
```

ผลลัพธ์ :



เมื่อใช้คำสั่ง fetchall ข้อมูลจะออกมาเป็นชนิด List โดยสามารถอ่านค่าข้อมูลบางฟิลด์ของบางเรคอร์ดได้ โดยการอ้างหมายเลขลำดับของตัวแปร List เช่น

เมื่อใช้คำสั่ง showgroup = cur.fetchall() ผลลัพธ์ที่ได้คือ [('01', 'computer'), ('02', 'stationery'), ('03', 'clothes')] โดยสามารถระบุข้อมูลบางส่วนของเรคอร์ดได้ ดังนี้คือ

- showgroup[0] คือ เรคอร์ดที่ 1 ในที่นี้คือ ('01', 'computer') ถ้าระบุตำแหน่ง showgroup[0][0] คือ ฟิลด์แรกของเรคอร์ดแรก คือ '01' ส่วน showgroup[0][1] คือ ฟิลด์ที่ 2 ของเรคอร์ดที่ 1 คือ ' computer '
- showgroup[1] คือ เรคอร์ดที่ 2 ในที่นี้คือ ('02', 'stationery') ถ้าระบุตำแหน่ง showgroup[1][0] คือ ฟิลด์แรกของเรคอร์ดที่ 2 คือ '02' ส่วน showgroup[1][1] คือ ฟิลด์ที่ 2 ของเรคอร์ดที่ 2 คือ 'stationery'
- showgroup[2] คือ เรคอร์ดที่ 3 ในที่นี้คือ ('03', 'clothes') ถ้าระบุตำแหน่ง showgroup[2][0] คือ ฟิลด์แรกของเรคอร์ดที่ 3 คือ '03' ส่วน showgroup[2][1] คือ ฟิลด์ที่ 2 ของเรคอร์ดที่ 3 คือ 'clothes'

ดังนั้นถ้าเราต้องการเรียกดูข้อมูลบางส่วนของเรคอร์ด ก็ให้ระบุตำแหน่งไปเลย เช่น

```
cur.execute("select * from product")
showproduct = cur.fetchall()
print("Product : ",showproduct)
print("ชื่อสินค้าของเรคอร์ดที่ 1 : ",showproduct[0][1])
print("ยี่ห้อของเรคอร์ดที่ 1 : ",showproduct[0][3])
print("ราคาของสินค้าของเรคอร์ดที่ 1 : ",showproduct[0][4])
```

ผลลัพธ์ที่ได้คือ

```
Product :  [('00001', 'printer', '01', 'epson', 8500)]
ชื่อสินค้าของเรคอร์ดที่ 1 :  printer
ยี่ห้อของเรคอร์ดที่ 1 :  epson
ราคาของสินค้าของเรคอร์ดที่ 1 :  8500
```

รูปแบบที่ 2 การใช้คำสั่ง for ร่วมกับคำสั่ง Cursor Object สำหรับอ่านค่าข้อมูลในตาราง

รูปแบบ :

```
cur.execute("select * from ชื่อตาราง")
for ชื่อตัวแปร in cur.fetchall()
print(ชื่อตัวแปร)
```

ตัวอย่าง :

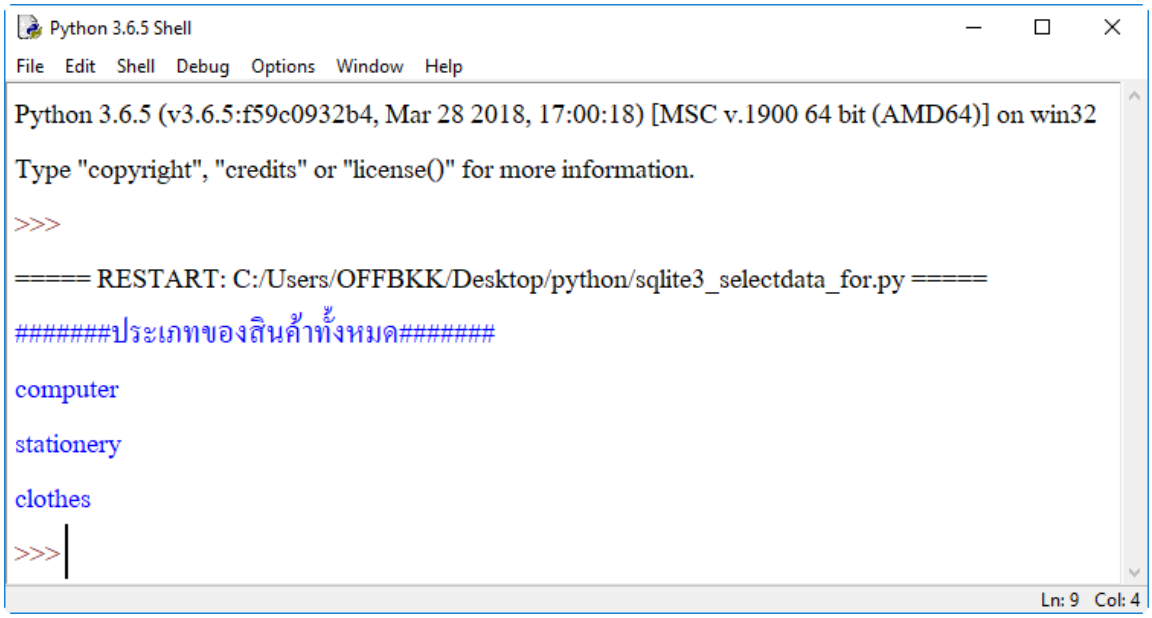
```
import sqlite3
cn = sqlite3.connect('product.db')
cur = cn.cursor()

cur.execute("select * from product")
print("#######Product #######")
for a in cur.fetchall() :
   print(a,'\n')

cur.execute("select * from productgroup")
print("#######Product Group #######")
for b in cur.fetchall() :
   print(b)
```

ผลลัพธ์ :

```
#######Product #######
('00001', 'printer', '01', 'epson', 8500)

#######Product Group #######
('01', 'computer')
('02', 'stationery')
('03', 'clothes')
```

ถ้าต้องการอ่านค่าข้อมูลบางฟิลด์ของเรคอร์ด ให้ระบุคำสั่งเป็น print(a[0]) แสดงค่าข้อมูลในฟิลด์แรก print(a[1]) แสดงค่าข้อมูลในฟิลด์ที่ 2 ไปเรื่อยๆ ตามลำดับ ตัวอย่างเช่น

```
cur.execute("select * from productgroup")
print("#######ประเภทของสินค้าทั้งหมด#######")
for b in cur.fetchall() :
    print(b[1])
```



รูปแบบที่ 3 การใช้คำสั่ง select อ่านค่าข้อมูลตารางมากกว่า 1 ตารางที่มีความสัมพันธ์กัน

ตัวอย่างเช่น :

```
cur.execute('''select pid, product.title, productgroup.title, product.price
                from product, productgroup
                where product.groupid = productgroup.groupid''')
for c in cur.fetchall() :
  print(c)
```

ผลลัพธ์ที่ได้ คือ จะแสดงข้อมูลในฟิลด์ pid, title, price ของตาราง product และฟิลด์ title ของตาราง productgroup โดยเรียงลำดับข้อมูลตามที่เรากำหนด ซึ่งสามารถแสดงข้อมูลจาก 2 ตารางได้ เพราะมีฟิลด์ที่สัมพันธ์กันคือ ฟิลด์ groupid นั่นเอง

```
('00001', 'printer', 'computer', 8500)
```

2.7 แก้ไขค่าข้อมูลเดิมในตารางด้วยคำสั่ง update

รูปแบบ : update ชื่อตาราง set ชื่อฟิลด์ = ค่าข้อมูลใหม่ where ชื่อฟิลด์ = ค่าข้อมูล

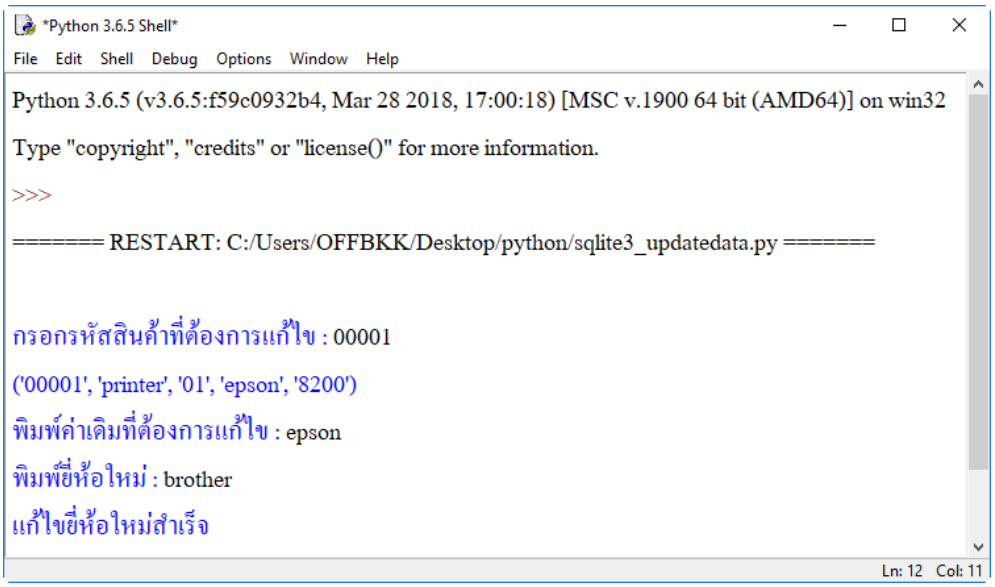
เช่น **:** update product set title = 'computer' where pid = '01'

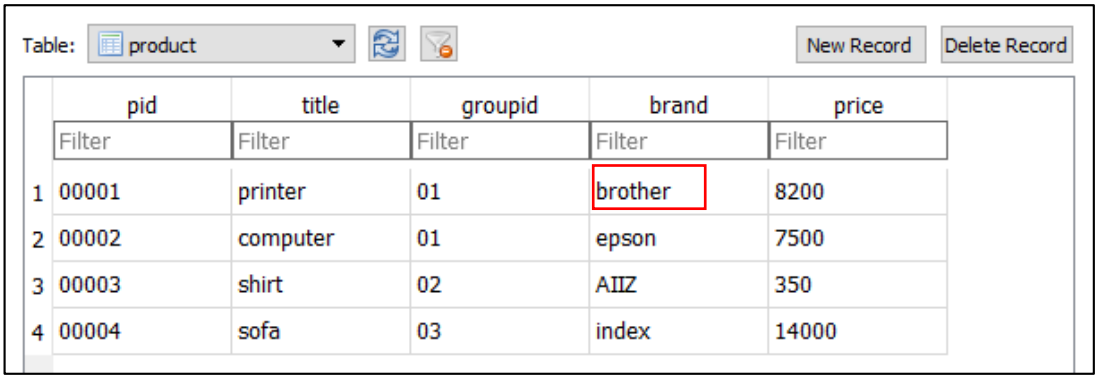
ตัวอย่าง :

```
import sqlite3
cn = sqlite3.connect('product.db')
cur = cn.cursor()
######################แก้ไขข้อมูลสินค้า#####################
while True :
  product_id = input("\nกรอกรหัสสินค้าที่ต้องการแก้ไข : ") #กรอกรหัสสินค้าที่ต้องการแก้ไข ถ้ากรอกว่า exit ให้ออกจากการทำงาน
  if product_id == 'exit' :
    break
  cur.execute("select * from product") #เรียกดูข้อมูลทั้งหมดในตาราง product โดยดูทีละเรคอร์ด
  for row in cur.fetchall() :
    if product_id == row[0] : #ถ้ารหัสที่กรอกเข้ามาตรงกับฟิลด์ pid (row[0])ในเรคอร์ดใด ให้พริ้นเรคอร์ดนั้นออกมา
      print(row)
      olddata = input("พิมพ์ค่าเดิมที่ต้องการแก้ไข : ") #พิมพ์ค่าเดิมที่ต้องการแก้ไข

      if row[0] == olddata : #ถ้าข้อมูลที่กรอกเข้ามาตรงกับฟิลด์ pid ให้ทำตามเงื่อนไขด้านล่าง
        newdata = input("พิมพ์รหัสสินค้าใหม่ : ") #พิมพ์ค่าใหม่ที่ต้องการแทนที่ข้อมูลเดิม
        cur.execute("update product set pid = ? where pid = ?",(newdata,row[0])) #เอาค่าในตัวแปร newdata ใส่เข้าไปในฟิลด์ pid
        print("แก้ไขรหัสสินค้าใหม่สำเร็จ")
```

```
elif row[1] == olddata :  #ถ้าข้อมูลที่กรอกเข้ามาตรงกับฟิลด์ title ให้ทำตามเงื่อนไขด้านล่าง
    newdata = input("พิมพ์ชื่อสินค้าใหม่ : ")
    cur.execute("update product set title = ? where pid = ?",(newdata,row[0]))
    print("แก้ไขชื่อสินค้าใหม่สำเร็จ")

elif row[2] == olddata :  #ถ้าข้อมูลที่กรอกเข้ามาตรงกับฟิลด์ groupid ให้ทำตามเงื่อนไขด้านล่าง
    newdata = input("พิมพ์รหัสประเภทสินค้าใหม่ : ")
    cur.execute("update product set groupid = ? where pid = ?",(newdata,row[0]))
    print("แก้ไขรหัสประเภทสินค้าใหม่สำเร็จ")

elif row[3] == olddata :  #ถ้าข้อมูลที่กรอกเข้ามาตรงกับฟิลด์ brand ให้ทำตามเงื่อนไขด้านล่าง
    newdata = input("พิมพ์ยี่ห้อใหม่ : ")
    cur.execute("update product set brand = ? where pid = ?",(newdata,row[0]))
    print("แก้ไขยี่ห้อใหม่สำเร็จ")

elif row[4] == olddata :  #ถ้าข้อมูลที่กรอกเข้ามาตรงกับฟิลด์ price ให้ทำตามเงื่อนไขด้านล่าง
    newdata = input("พิมพ์ราคาสินค้าใหม่ : ")
    cur.execute("update product set price = ? where pid = ?",(newdata,row[0]))
    print("แก้ไขราคาสินค้าใหม่สำเร็จ")
else :
    break
cn.commit()
```

ผลลัพธ์ที่ได้ คือ





	pid	title	groupid	brand	price
	Filter	Filter	Filter	Filter	Filter
1	00001	printer	01	brother	8200
2	00002	computer	01	epson	7500
3	00003	shirt	02	AIIZ	350
4	00004	sofa	03	index	14000

2.8 ลบค่าข้อมูลในตารางด้วยคำสั่ง delete

รูปแบบ : delete from ชื่อตาราง where ชื่อฟิลด์ = ค่าข้อมูล

เช่น : delete from product where pid = '01'

ตัวอย่าง

```
import sqlite3
cn = sqlite3.connect('product.db')
cur = cn.cursor()
######################ลบข้อมูลสินค้า######################
while True :
   product_id = input("\nกรอกรหัสสินค้าที่ต้องการลบ : ") #กรอกรหัสสินค้าที่ต้องการลบ
   if product_id == 'exit' :  #ถ้ากรอกว่า exit ให้ออกจากการทำงาน
      break
   cur.execute("select * from product") #เรียกดูข้อมูลทั้งหมดในตาราง product โดยดูทีละเรคอร์ด
   for row in cur.fetchall() :
      if product_id == row[0] : #ถ้ารหัสที่กรอกเข้ามาตรงกับฟิลด์ pid (row[0])ในเรคอร์ดใด ให้พริ้นเรคอร์ดนั้นออกมา
         print(row)
         cur.execute("delete from product where pid = ?",(product_id,))  #ลบเรคอร์ดนั้นออกไป
         cn.commit()
         print("ลบเรคอร์ดข้อมูลสินค้าสำเร็จ")
```

ผลลัพธ์ที่ได้

```
กรอกรหัสสินค้าที่ต้องการลบ : 00001
('00001', 'printer', '01', 'brother', '8200')
ลบเรคอร์ดข้อมูลสินค้าสำเร็จ
```

Table: product

	pid	title	groupid	brand	price
	Filter	Filter	Filter	Filter	Filter
1	00002	computer	01	epson	7500
2	00003	shirt	02	AIIZ	350
3	00004	sofa	03	index	14000

2.9 ปิดการเชื่อมต่อฐานข้อมูล

การปิดการเชื่อมต่อไฟล์ฐานข้อมูล มีรูปแบบการเขียนคำสั่ง ดังนี้คือ

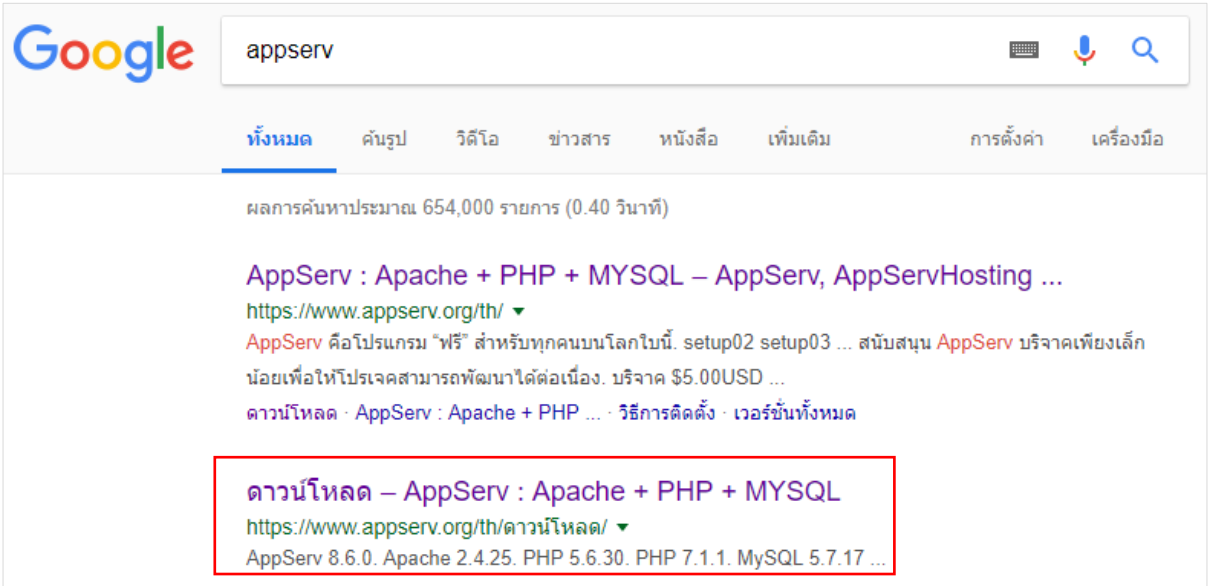
```
cn.close()
```

มักใช้คำสั่งนี้ก่อนสิ้นสุดการใช้งานของโปรแกรม เพื่อไม่ให้กระบวนการทำงาน Engine ด้านฐานข้อมูลตกค้างในระบบปฏิบัติการ เมื่อยุติการใช้โปรแกรมไปแล้ว

3. ฐานข้อมูลประเภท Database Server

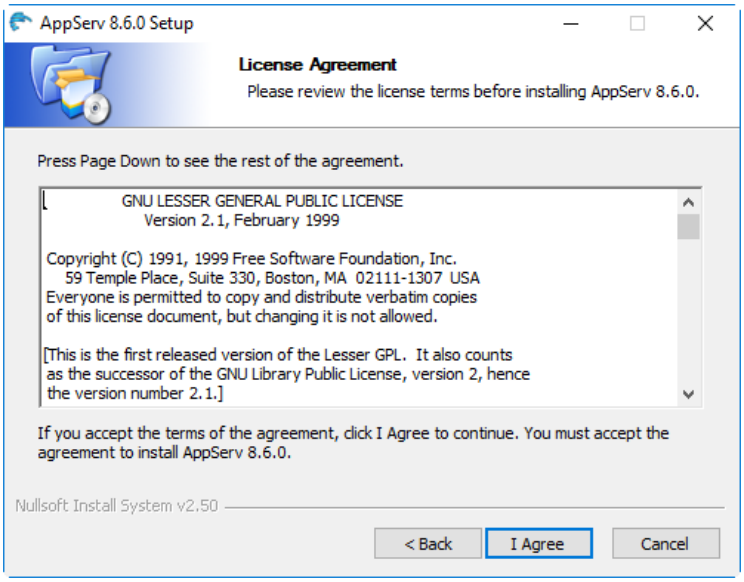
3.1 ดาวน์โหลดและติดตั้ง Appserv เพื่อใช้งาน localhost โดยมีขั้นตอนดังนี้

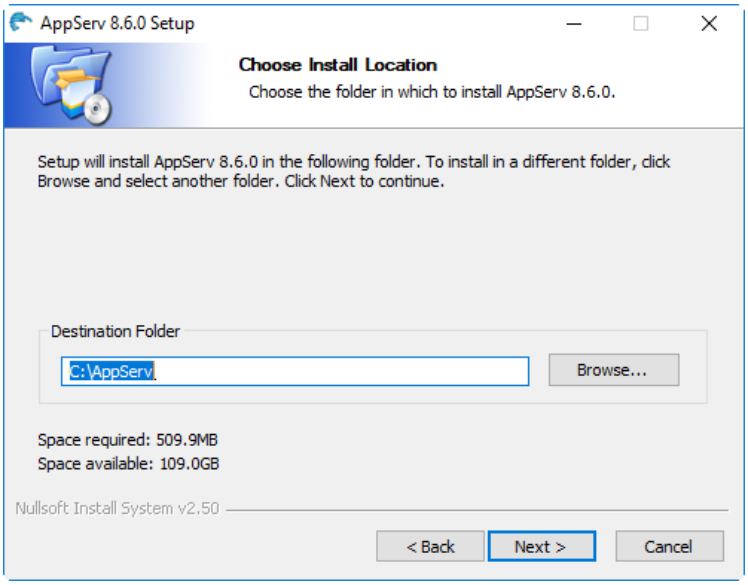
3.1.1 ดาวน์โหลด appserv จากเว็บไซต์ https://www.appserv.org/th/ ดังรูป

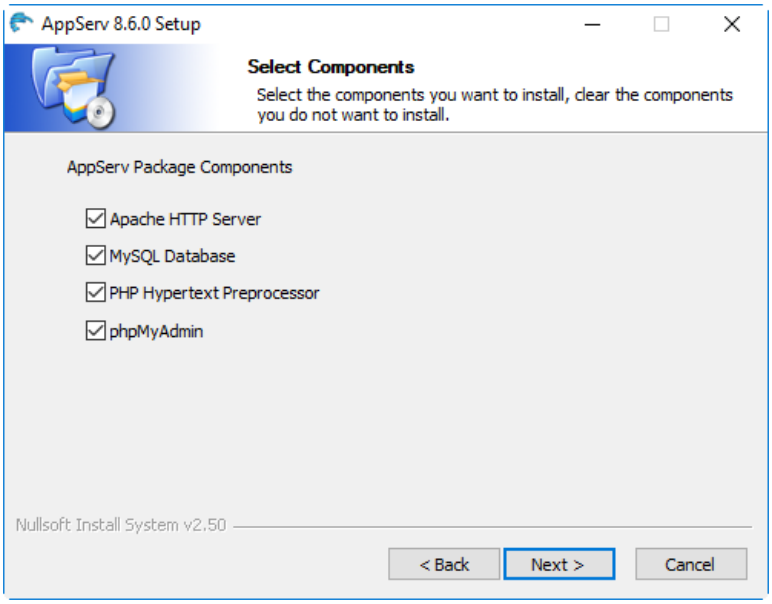


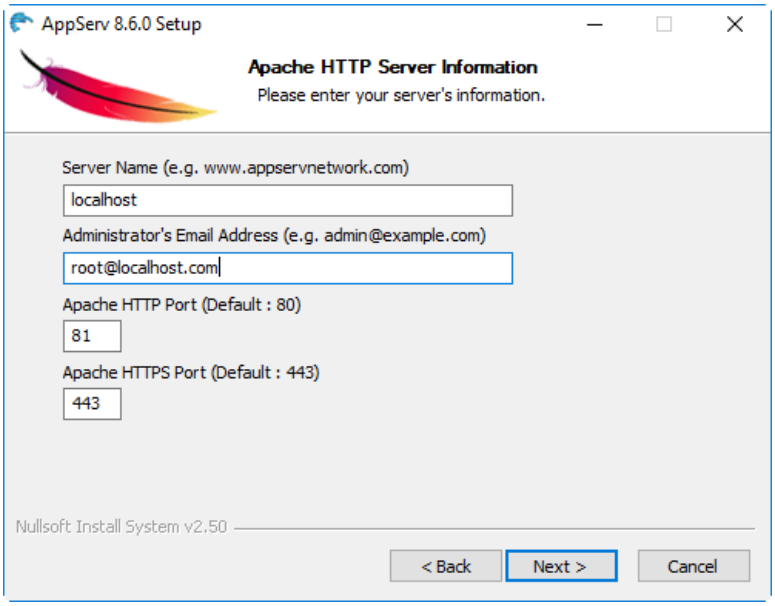
จะได้ไฟล์ติดตั้ง Appserv ชื่อว่า appserv-win32-8.6.0.exe ให้ดับเบิ้ลคลิกที่ไฟล์เพื่อทำการติดตั้งโปรแกรม

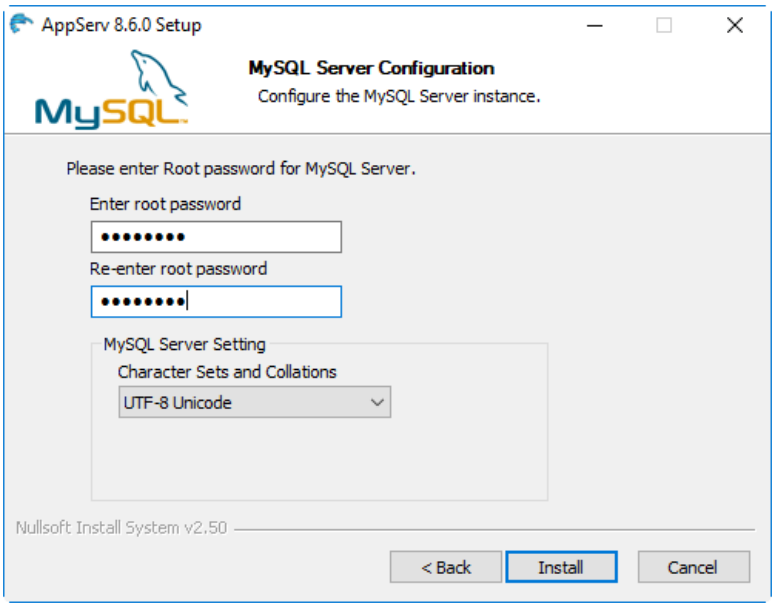
3.1.2 ติดตั้งโปรแกรม appserv โดยทำตามขั้นตอนต่อไปนี้

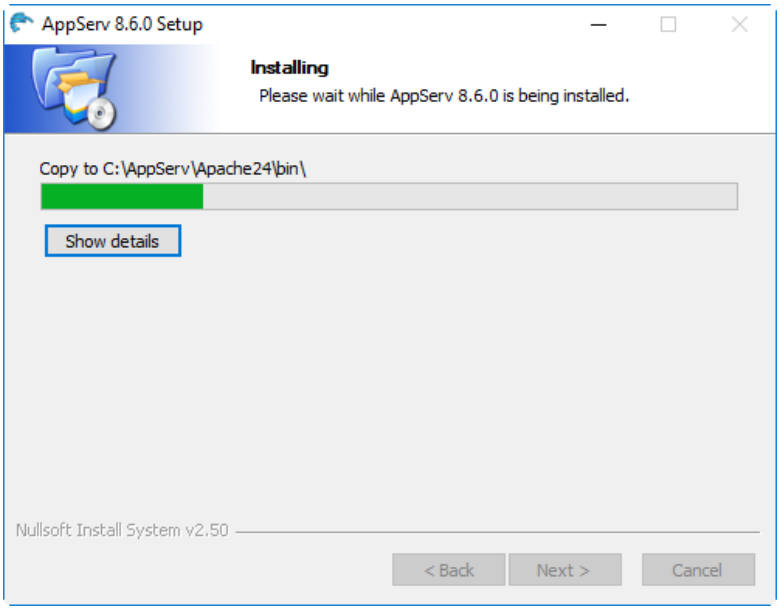

AppServ 8.6.0 Setup
License Agreement
Please review the license terms before installing AppServ 8.6.0.
Press Page Down to see the rest of the agreement.
GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999
Copyright (C) 1991, 1999 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
[This is the first released version of the Lesser GPL. It also counts
as the successor of the GNU Library Public License, version 2, hence
the version number 2.1.]
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install AppServ 8.6.0.
Nullsoft Install System v2.50
< Back
I Agree
Cancel

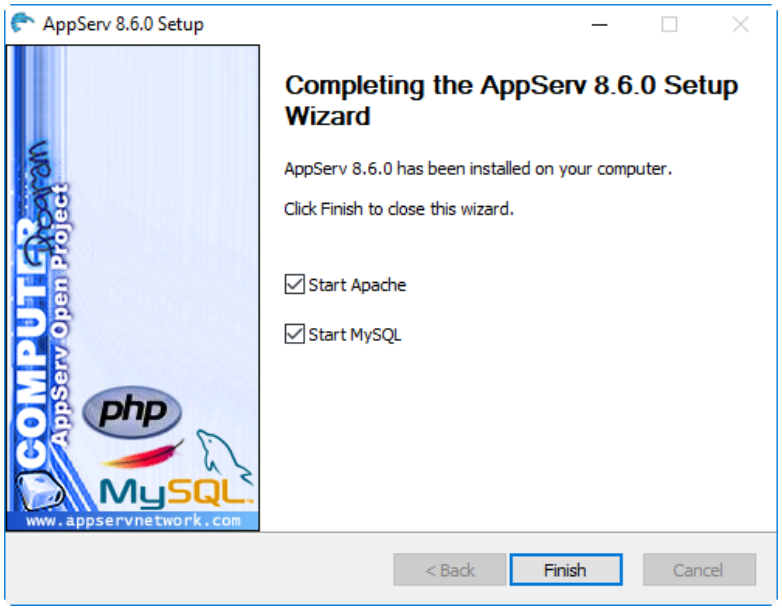

AppServ 8.6.0 Setup
Choose Install Location
Choose the folder in which to install AppServ 8.6.0.
Setup will install AppServ 8.6.0 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:\AppServ\
Browse...
Space required: 509.9MB
Space available: 109.0GB
Nullsoft Install System v2.50
< Back
Next >
Cancel


AppServ 8.6.0 Setup
Select Components
Select the components you want to install, clear the components you do not want to install.
AppServ Package Components
Apache HTTP Server
MySQL Database
PHP Hypertext Preprocessor
phpMyAdmin
Nullsoft Install System v2.50
< Back
Next >
Cancel

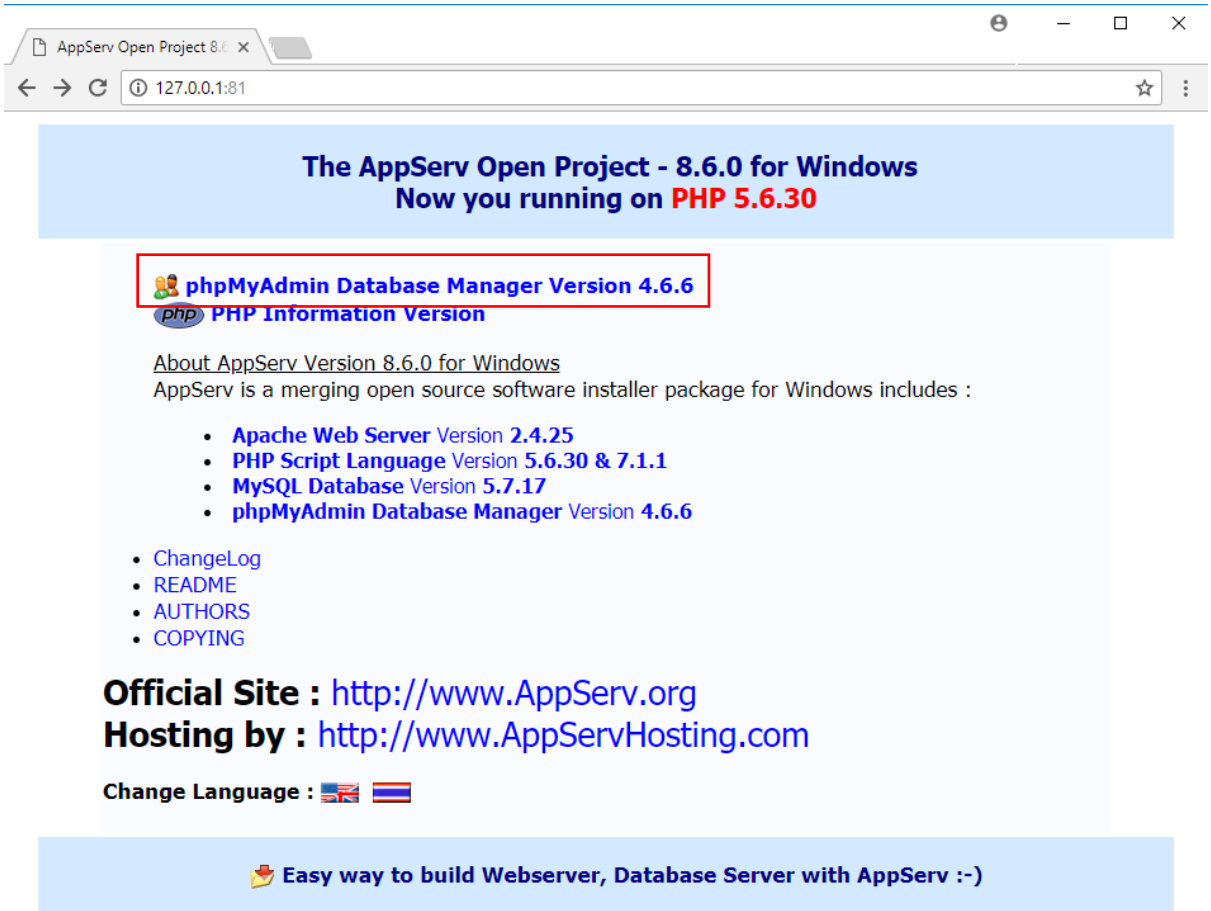

AppServ 8.6.0 Setup
Apache HTTP Server Information
Please enter your server's information.
Server Name (e.g. www.appservnetwork.com)
localhost
Administrator's Email Address (e.g. admin@example.com)
root@localhost.com
Apache HTTP Port (Default : 80)
81
Apache HTTPS Port (Default : 443)
443
Nullsoft Install System v2.50
< Back
Next >
Cancel


AppServ 8.6.0 Setup
MySQL Server Configuration
Configure the MySQL Server instance.
Please enter Root password for MySQL Server.
Enter root password
Re-enter root password
MySQL Server Setting
Character Sets and Collations
UTF-8 Unicode
Nullsoft Install System v2.50
< Back
Install
Cancel

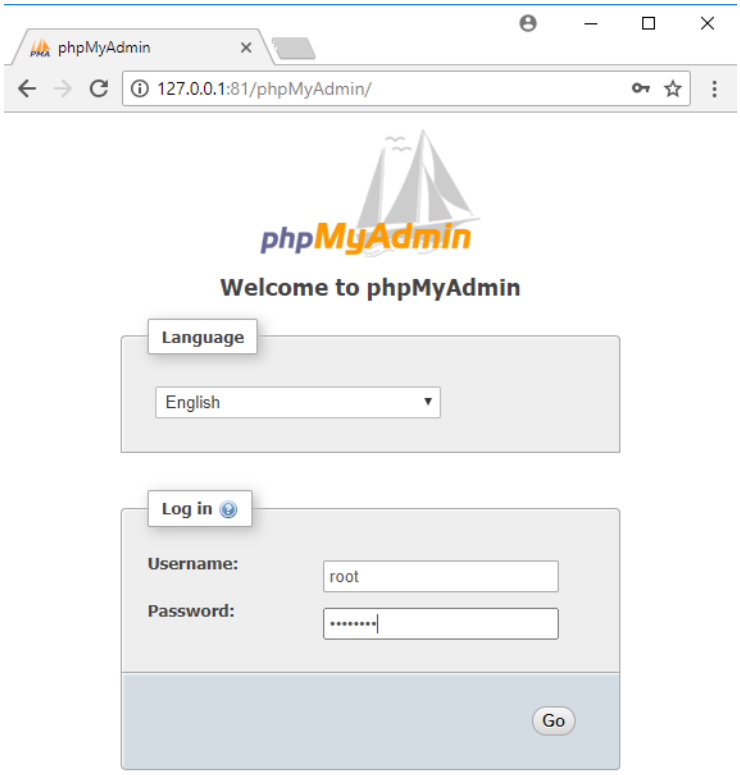

AppServ 8.6.0 Setup
Installing
Please wait while AppServ 8.6.0 is being installed.
Copy to C:\AppServ\Apache24\bin\
Show details
Nullsoft Install System v2.50
< Back
Next >
Cancel



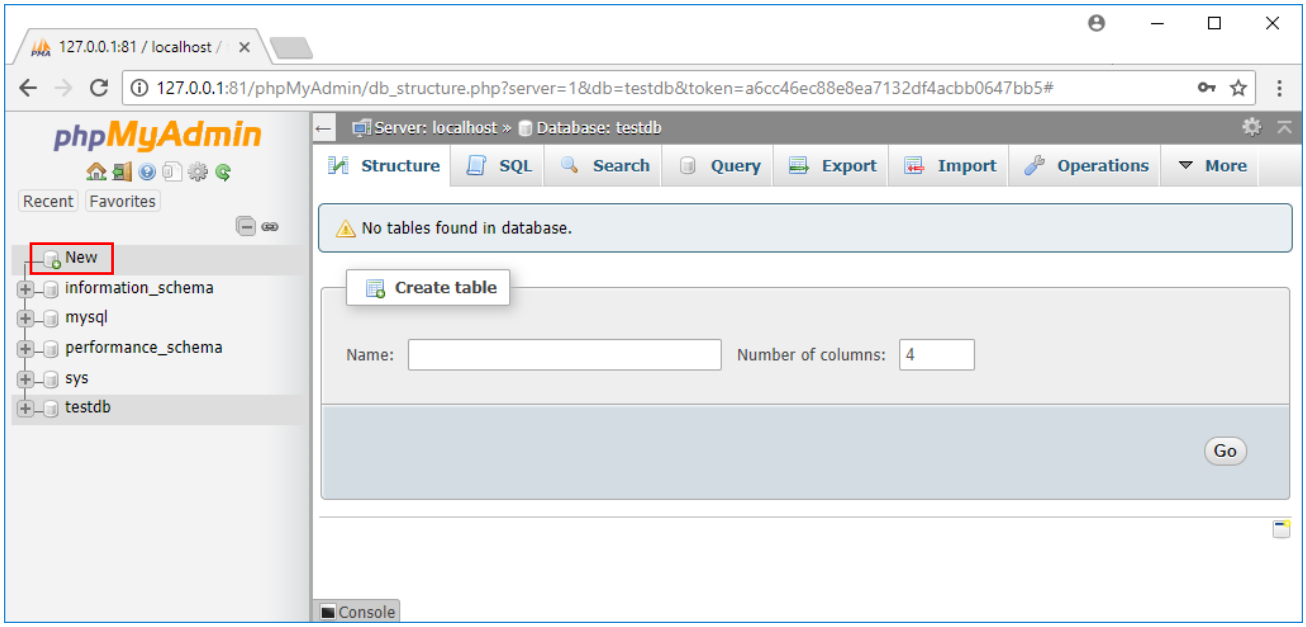
3.1.3 เปิดบราวเซอร์ Internet Explorer หรือ Google Chrome ขึ้นมา แล้วเข้าสู่หน้าแรกของ Appserv โดยพิมพ์คำว่า http://127.0.0.1:81 แล้วคลิกที่ phpMyAdmin Database Manager Version 4.6.6 ดังรูป



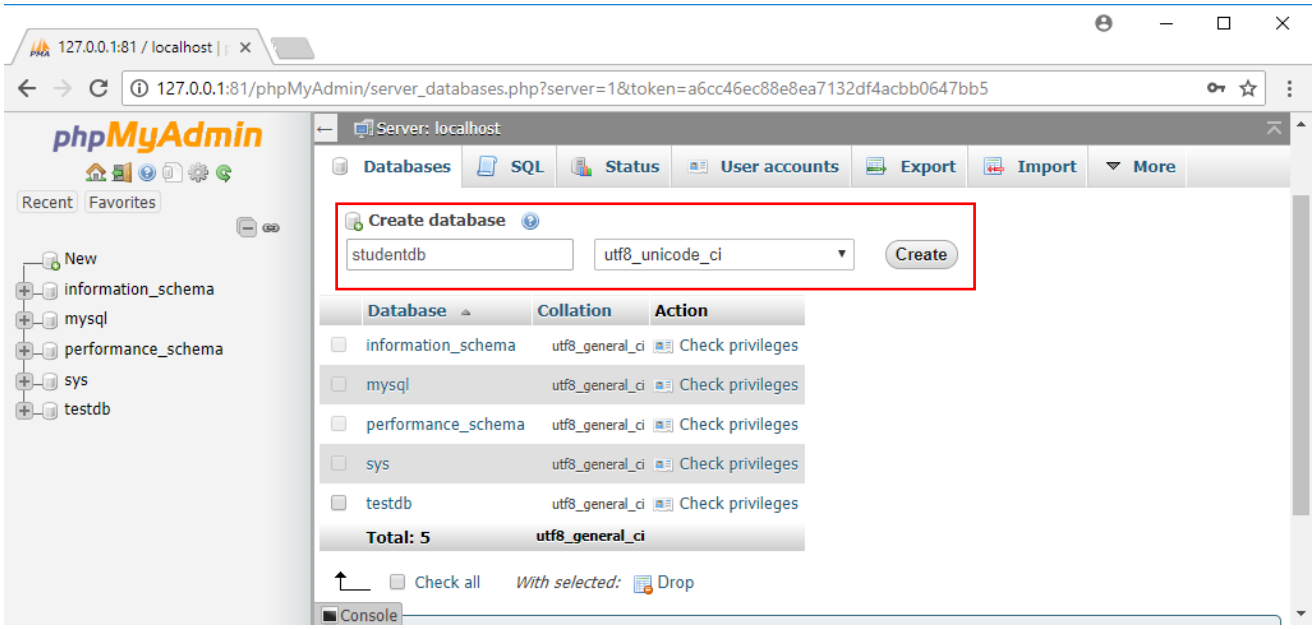
3.1.4 กรอก Username และ Password ที่ได้กำหนดไว้ในขั้นตอนการติดตั้ง แล้วคลิกปุ่ม Go



3.1.5 สร้างฐานข้อมูล หรือ Database โดยคลิกที่ New

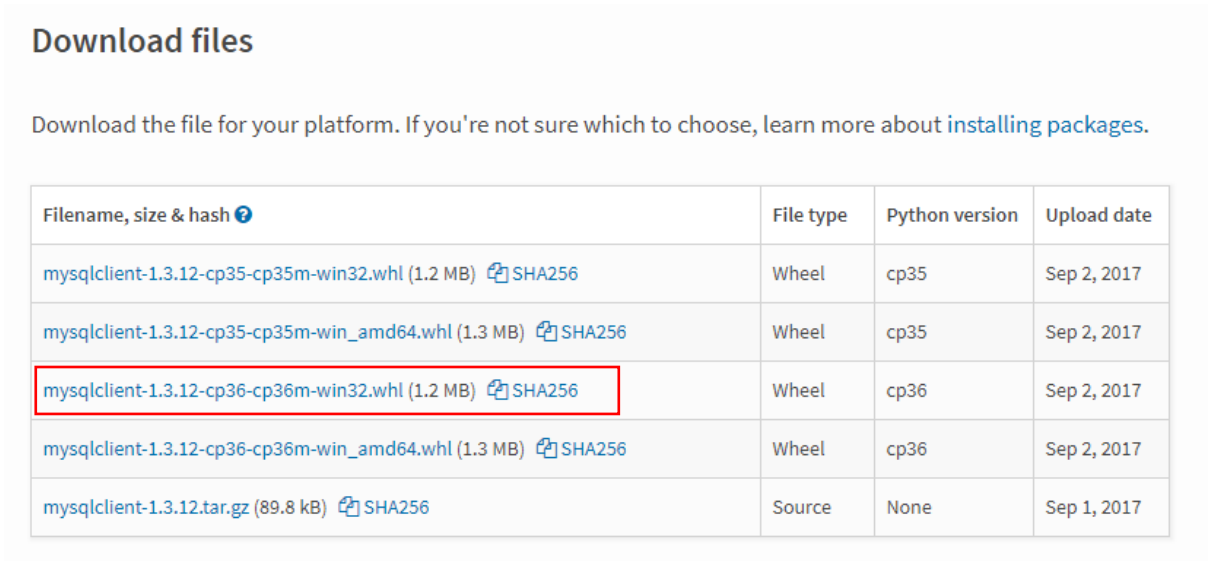


ใส่ชื่อ database และภาษา ซึ่งที่นี้จะเลือกเป็น utf8_unicode_ci เพื่อให้รองรับข้อมูลภาษาไทย แล้วกดปุ่ม Create



3.2 ดาวน์โหลดโมดูลพิเศษ MySQLdb

3.2.1 ทำการติดตั้งโมดูลพิเศษที่ชื่อว่า MySQLdb เพื่อเชื่อมต่อฐานข้อมูล MySQL กับภาษา Python จากเว็บไซต์ https://pypi.org/project/mysqlclient/#files โดยเลือกไฟล์ที่ชื่อว่า “mysqlclient-1.3.12-cp36-cp36m-win32.whl” ดังรูป



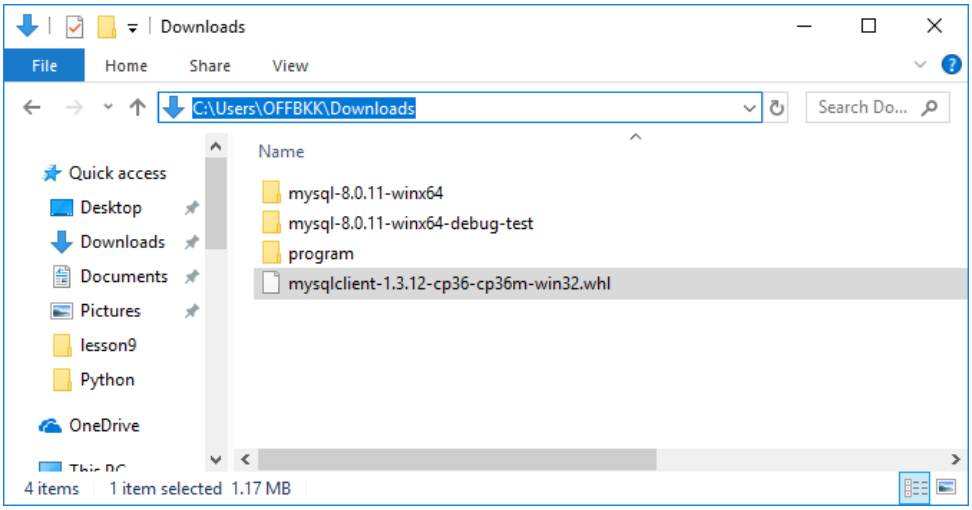
Download files

Download the file for your platform. If you're not sure which to choose, learn more about installing packages.

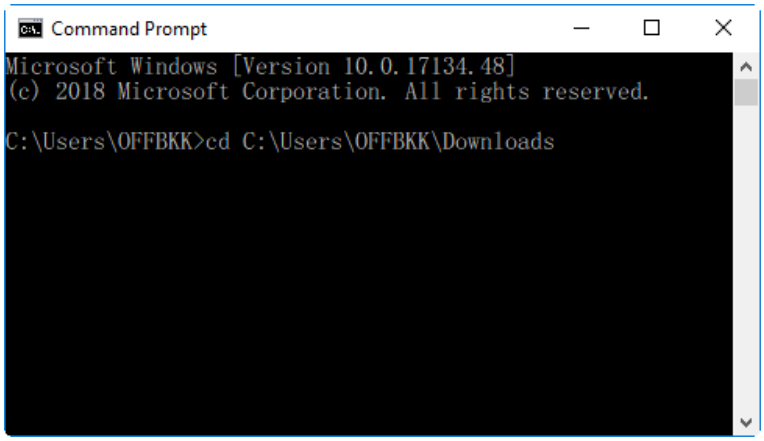
Filename, size & hash	File type	Python version	Upload date
mysqlclient-1.3.12-cp35-cp35m-win32.whl (1.2 MB) SHA256	Wheel	cp35	Sep 2, 2017
mysqlclient-1.3.12-cp35-cp35m-win_amd64.whl (1.3 MB) SHA256	Wheel	cp35	Sep 2, 2017
mysqlclient-1.3.12-cp36-cp36m-win32.whl (1.2 MB) SHA256	Wheel	cp36	Sep 2, 2017
mysqlclient-1.3.12-cp36-cp36m-win_amd64.whl (1.3 MB) SHA256	Wheel	cp36	Sep 2, 2017
mysqlclient-1.3.12.tar.gz (89.8 kB) SHA256	Source	None	Sep 1, 2017

3.2.2 เมื่อทำการดาวน์โหลดไฟล์เรียบร้อยแล้ว ไฟล์ติดตั้งจะอยู่ในโฟลเดอร์ Downloads ของเครื่องคอมพิวเตอร์ ให้ทำตามขั้นตอนดังต่อไปนี้

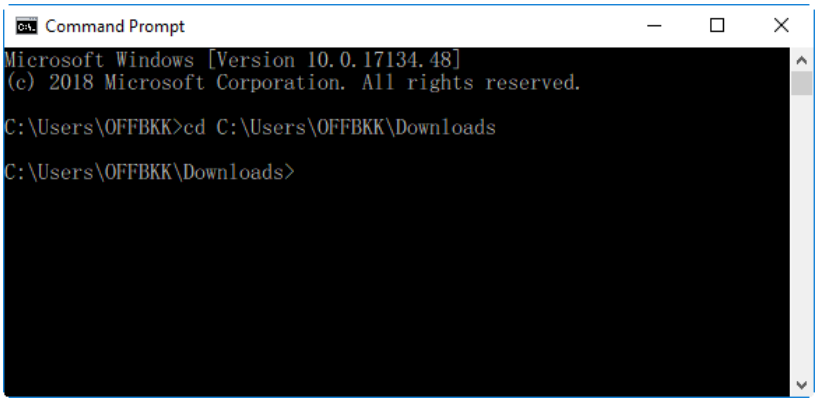
1) คัดลอกตำแหน่งที่อยู่ของไฟล์ที่เพิ่งดาวน์โหลดมา



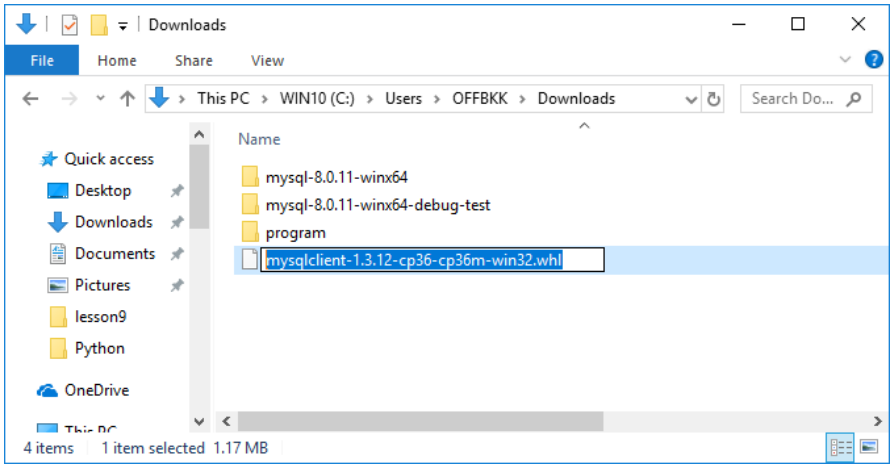
2) เปิดโปรแกรม Command Prompt ขึ้นมา แล้วพิมพ์ว่า cd ตามด้วยตำแหน่งไฟล์ที่เราคัดลอกมา ในที่นี้จะเป็น cd C:\Users\OFFBKK\Downloads ดังรูป



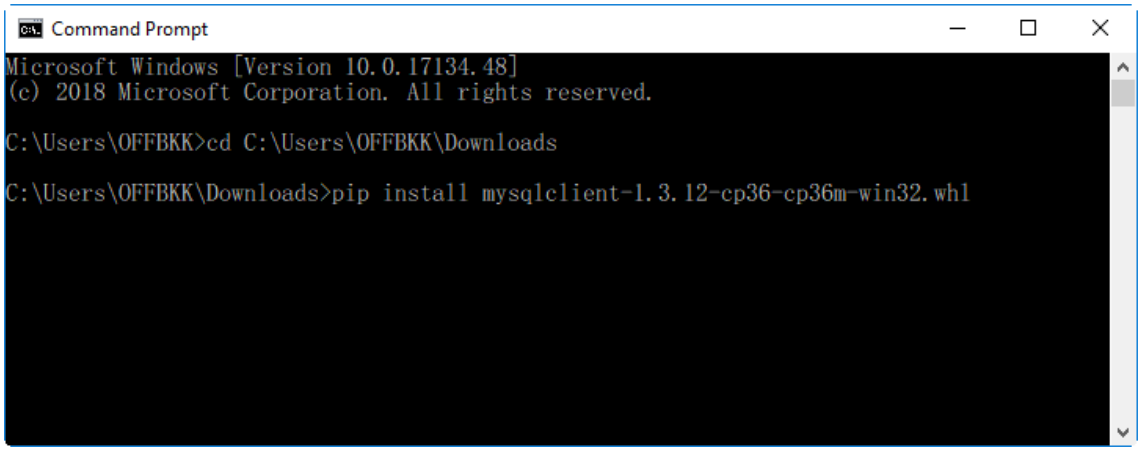
แล้วกดปุ่ม Enter จะได้ผลลัพธ์ดังรูป



3) หลังจากนั้นให้พิมพ์คำสั่ง pip install ตามด้วยชื่อไฟล์ (โดยคัดลอกชื่อไฟล์ ดังรูป) ในที่นี้ไฟล์ที่ดาวน์โหลดมาชื่อว่า mysqlclient-1.3.12-cp36-cp36m-win32.whl



จึงต้องพิมพ์ว่า pip install mysqlclient-1.3.12-cp36-cp36m-win32.whl แล้วกด Enter เพื่อทำการติดตั้ง เมื่อติดตั้งเรียบร้อยแล้ว จะมีข้อความว่า Successfully



3.2.3 เปิดโปรแกรม Python ขึ้นมา หลังจากนั้นให้ลองพิมพ์คำสั่ง import MySQLdb ถ้าไม่มี Error แสดงว่าได้ทำการติดตั้งโมดูล MySQLdb เรียบร้อยแล้ว

3.3 การเขียนโปรแกรมสำหรับเชื่อมต่อฐานข้อมูล MySQL ด้วยโมดูล MySQLdb

3.3.1 เรียกใช้โมดูล MySQLdb โดยใช้คำสั่ง

```
import MySQLdb
```

3.3.2 ใช้ Connect Object เชื่อมต่อกับฐานข้อมูล MySQL โดยจะต้องระบุชื่อผู้ใช้และรหัสผ่าน โดยมีรูปแบบการเขียนดังนี้ คือ

```
cn = MySQLdb.connect(host = "ชื่อ host", user = "ชื่อผู้ใช้", passwd = "รหัสผ่าน", db = "ชื่อฐานข้อมูล")
cur = cn.cursor()
```

ตัวอย่างเช่น

```
import MySQLdb
cn = MySQLdb.connect(host = "localhost", user = "root", passwd = "12345678",
                db = "studentdb")
print("เปิดฐานข้อมูลสำเร็จ")
cur = cn.cursor()
```

3.3.3 เขียนโปรแกรมสำหรับเชื่อมต่อฐานข้อมูล MySQL

การเขียนโปรแกรมสำหรับเชื่อมต่อฐานข้อมูล MySQL ด้วยโมดูล MySQLdb จะมีลักษณะเหมือนกับการใช้โมดูล SQLite ในหัวข้อที่ผ่านมา โดยมีคำสั่งในการจัดการฐานข้อมูล เช่น การสร้างตารางด้วยคำสั่ง create, การเพิ่มเรคอร์ดข้อมูลใหม่ด้วยคำสั่ง insert, การเรียกดูข้อมูลด้วยคำสั่ง select, การแก้ไขเรคอร์ดข้อมูลด้วยคำสั่ง update, การลบเรคอร์ดข้อมูลด้วยคำสั่ง delete เป็นต้น

1) การสร้างตารางด้วยคำสั่ง create

```
import MySQLdb
cn = MySQLdb.connect(host = "localhost", user = "root", passwd = "12345678",
                db = "studentdb")
print("เปิดฐานข้อมูลสำเร็จ")
cur = cn.cursor()

##########################สร้างตาราง#######################################
cur.execute('''create table student
    (id int PRIMARY KEY    NOT NULL,
    firstname  char(50)    NOT NULL,
    lastname char(50)    NOT NULL,
    classroom char(10)  NOT NULL,
    province char(20)  NOT NULL,
    phone  char(10) NOT NULL);''')
print("สร้างตารางสำเร็จ :D ")
cur.close()
```

2) การเพิ่มเรคอร์ดข้อมูลใหม่ด้วยคำสั่ง insert

```
import MySQLdb
cn = MySQLdb.connect(host = "localhost", user = "root", passwd = "12345678", db = "studentdb")
print("เปิดฐานข้อมูลสำเร็จ")
cur = cn.cursor()

############################เพิ่มข้อมูล#########################################
while True :
   i = input('Student ID : ')
   if i == 'exit' or i == '' :
      break
   f = input('Name : ')
   l = input('Lastname : ')
   c = input('Classroom : ')
   p = input('Province : ')
   t = input('Tel : ')
   cur.execute("insert into student values (%s,'%s','%s','%s','%s', '%s')" %(i,f,l,c,p,t))
   cn.commit()

cur.close()
cn.close()
```

ผลลัพธ์ที่ได้

```
เปิดฐานข้อมูลสำเร็จ
Student ID : 4
Name : Siriporn
Lastname : Larpsiriwat
Classroom : M.1/5
Province : Petchaburi
Tel : 0899551200
Student ID :
```

หมายเหตุ : %s ใช้สำหรับตัวแปรชนิด integer ส่วน '%s' ใช้สำหรับตัวแปรชนิด string

3) การเรียกดูข้อมูลด้วยคำสั่ง select

```
import MySQLdb
cn = MySQLdb.connect(host = "localhost", user = "root", passwd = "12345678", db = "studentdb")
print("เปิดฐานข้อมูลสำเร็จ")
cur = cn.cursor()

##########################เรียกดูข้อมูล#########################################
cur.execute("select firstname,lastname,classroom  from student")
# cur.execute("select *  from student")
for r in cur.fetchall() :
  print(r)
cur.close()
```

ผลลัพธ์ที่ได้

```
เปิดฐานข้อมูลสำเร็จ
('nattha', 'meesuk', 'M.2/1')
('Chakrit', 'Jaidee', 'M.5/2')
('Natthawut', 'suttichaikul', 'M.3/1')
('Siriporn', 'Larpsiriwat', 'M.1/5')
```

4) การแก้ไขเรคอร์ดข้อมูลด้วยคำสั่ง update

```
import MySQLdb
cn = MySQLdb.connect(host = "localhost", user = "root", passwd = "12345678", db = "studentdb")
print("เปิดฐานข้อมูลสำเร็จ")
cur = cn.cursor()

##############################เรียกดูข้อมูลนักเรียนทั้งหมด##############################
cur.execute("select * from student")
for row in cur.fetchall() :
   print(row)

##################################แก้ไขข้อมูลนักเรียน#################################
while True :
   student_id = int(input("\nกรอกรหัสนักเรียนที่ต้องการแก้ไข : "))
   if student_id == 'exit' :
      break
   cur.execute("select * from student")
   for row in cur.fetchall() :
      if student_id == row[0] :
         print(row)
         olddata = input("พิมพ์ค่าเดิมที่ต้องการแก้ไข : ")
```

```
if row[0] == int(olddata) :
    newdata = int(input("พิมพ์รหัสนักเรียนใหม่ : "))
    cur.execute("update student set id = %s where id = %s",(newdata,row[0]))
    print("แก้ไขรหัสนักเรียนใหม่สำเร็จ")

elif row[1] == olddata :
    newdata = input("พิมพ์ชื่อใหม่ : ")
    cur.execute("update student set firstname = '%s' where id = %s",(newdata,row[0]))
    print("แก้ไขชื่อใหม่สำเร็จ")

elif row[2] == olddata :
    newdata = input("พิมพ์นามสกุลใหม่ : ")
    cur.execute("update student set lastname = '%s' where id = %s",(newdata,row[0]))
    print("แก้ไขนามสกุลใหม่สำเร็จ")

elif row[3] == olddata :
    newdata = input("พิมพ์ห้องเรียนใหม่ : ")
    cur.execute("update student set classroom = '%s' where id = %s",(newdata,row[0]))
    print("แก้ไขห้องเรียนใหม่สำเร็จ")
elif row[4] == olddata :
    newdata = input("พิมพ์จังหวัดใหม่ : ")
    cur.execute("update student set province = '%s' where id = %s",(newdata,row[0]))
    print("แก้ไขจังหวัดใหม่สำเร็จ")

elif row[5] == olddata :
    newdata = input("พิมพ์เบอร์โทรใหม่ : ")
    cur.execute("update student set tel = '%s' where id = %s",(newdata,row[0]))
    print("แก้ไขเบอร์โทรใหม่สำเร็จ")
else :
    break
cn.commit()
```

ผลลัพธ์ที่ได้

```
เปิดฐานข้อมูลสำเร็จ
(1, 'nattha', 'meesuk', 'M.2/1', 'bangkok', '0894513633')
(2, 'Chakrit', 'Jaidee', 'M.5/2', 'Nonthaburi', '0872512055')
(3, 'Natthawut', 'suttichaikul', 'M.3/1', 'Chiang-Mai', '0856478544')
(4, 'Siriporn', 'Larpsiriwat', 'M.1/5', 'Petchaburi', '0899551200')

กรอกรหัสนักเรียนที่ต้องการแก้ไข : 1
(1, 'nattha', 'meesuk', 'M.2/1', 'bangkok', '0894513633')
พิมพ์ค่าเดิมที่ต้องการแก้ไข : 1
พิมพ์รหัสนักเรียนใหม่ : 5
แก้ไขรหัสนักเรียนใหม่สำเร็จ
```

ลักษณะการทำงานของโปรแกรม

รับค่ารหัสนักเรียนทางคีย์บอร์ดด้วยคำสั่ง input() และแปลงค่าเป็น integer เพราะจะต้องนำค่าตัวแปร student_id ไปเปรียบเทียบกับฟิลด์ id (row[0]) ของตาราง ซึ่งเป็นชนิด integer หลังจากนั้นใช้คำสั่ง cur.execute("select * from student") เพื่อเรียกดูข้อมูลทั้งหมดในตาราง วนลูปคำสั่ง for row in cur.fetchall() เพื่อดูข้อมูลทีละเรคอร์ด ในระหว่างการวนลูปให้ทำการตรวจสอบเงื่อนไขว่า รหัสนักเรียนที่กรอกเข้ามาตรงกับฟิลด์ id ในตารางหรือไม่ (student_id == row[0]) ถ้าตรงกันให้แสดงเรคอร์ดนั้นมาทางหน้าจอด้วยคำสั่ง print(row) จากนั้นให้กรอกค่าเดิมที่ต้องการแก้ไข แล้วนำค่ามาเปรียบเทียบว่าตรงกับฟิลด์ใด เช่น ถ้าตรงกับฟิลด์ id ก็ให้ทำการแทนที่ข้อมูลของฟิลด์ id ด้วยคำสั่ง cur.execute("update student set id = %s where id = %s",(newdata, row[0])) โดย %s ตัวแรกคือตัวแปร newdata ก็คือข้อมูลใหม่ที่ต้องใส่เข้าไป ส่วน %s ตัวที่สองคือ row[0] เป็นการระบุตำแหน่งของเรคอร์ดที่ต้องการแก้ไข

5) การลบเรคอร์ดข้อมูลด้วยคำสั่ง delete

```
import MySQLdb
cn = MySQLdb.connect(host = "localhost", user = "root", passwd = "12345678", db = "studentdb")
print("เปิดฐานข้อมูลสำเร็จ")
cur = cn.cursor()
##############################เรียกดูข้อมูลนักเรียนทั้งหมด##############################
cur.execute("select * from student")
for r in cur.fetchall() :
   print(r)
##################################แก้ไขข้อมูลนักเรียน#################################
while True :
   student_id = int(input("\nกรอกรหัสนักเรียนที่ต้องการลบ : "))
   cur.execute("select * from student")
   for row in cur.fetchall() :
      if student_id == row[0] :
         print(row)
         confirm = input("ยืนยันการลบข้อมูลนักเรียน กรุณาพิมพ์ Y : ")
         if confirm == 'y' or 'Y' :
            cur.execute("delete from student where id = %s", (row[0],))
            cn.commit()
            print("ลบข้อมูลนักเรียนสำเร็จ")
         else :
            break
```

ผลลัพธ์ที่ได้

```
เปิดฐานข้อมูลสำเร็จ
(1, 'nuttapon', 'Phokiat', 'M.2/5', 'Bangkok', '0958742104')
(2, 'Suksan', 'Poonlap', 'M02/4', 'Nonthaburi', '0894523655')
(3, 'Panya', 'Jaidee', 'M.5/4', 'Bangkok', '0875692541')
(5, 'nattha', 'meesuk', 'M.2/1', 'bangkok', '0894513633')

กรอกรหัสนักเรียนที่ต้องการลบ : 1
(1, 'nuttapon', 'Phokiat', 'M.2/5', 'Bangkok', '0958742104')
ยืนยันการลบข้อมูลนักเรียน กรุณาพิมพ์ Y : y
ลบข้อมูลนักเรียนสำเร็จ
```

ลักษณะการทำงานของโปรแกรม

รับค่ารหัสนักเรียนทางคีย์บอร์ดด้วยคำสั่ง input() และแปลงค่าเป็น integer เพราะจะต้องนำค่าตัวแปร student_id ไปเปรียบเทียบกับฟิลด์ id (row[0]) ของตาราง ซึ่งเป็นชนิด integer หลังจากนั้นใช้คำสั่ง cur.execute("select * from student") เพื่อเรียกดูข้อมูลทั้งหมดในตาราง วนลูปคำสั่ง for row in cur.fetchall() เพื่อดูข้อมูลทีละเรคอร์ด ในระหว่างการวนลูปให้ทำการตรวจสอบเงื่อนไขว่า รหัสนักเรียนที่กรอกเข้ามาตรงกับฟิลด์ id ในตารางหรือไม่ (student_id == row[0]) ถ้าตรงกันให้แสดงเรคอร์ดนั้นมาทางหน้าจอด้วยคำสั่ง print(row) จากนั้นให้ทำการยืนยันการลบ ถ้ามีการพิมพ์ y หรือ Y ทางคีย์บอร์ดให้ทำการลบเรคอร์ดนั้นด้วยคำสั่ง cur.execute("delete from student where id = %s", (row[0],))

บทที่ 9 พื้นฐานการใช้โมดูล tkinter สำหรับ Graphical User Interface (GUI)

รูปแบบการเขียนคำสั่งภาษาไพธอนมี 2 รูปแบบ คือ แบบป้อนคำสั่งทีละบรรทัด (Command line interface และแบบกราฟิก คือ การติดต่อสื่อสารกับระบบคอมพิวเตอร์โดยอาศัยอินเตอร์เฟสแบบกราฟิก ซึ่งประกอบไปด้วยเมนู รูปภาพ ไอคอน โดยการใช้เมาส์คลิกเพื่อสั่งงาน ซึ่งในบทนี้จะอธิบายการสร้างอินเตอร์เฟสแบบกราฟิกด้วยภาษาไพธอนเป็นหลัก

ไพธอนได้รับการสนับสนุนโมดูลสำหรับพัฒนา GUI (Graphic User Interface) ไว้มากมาย เช่น tkinter, wxPython, JPython, wxWidgets, Qt, Gtk+, FLTK, FOX และ OpenGL เป็นต้น ซึ่งมีรายละเอียดเฉพาะโมดูลที่สำคัญๆ ดังนี้คือ

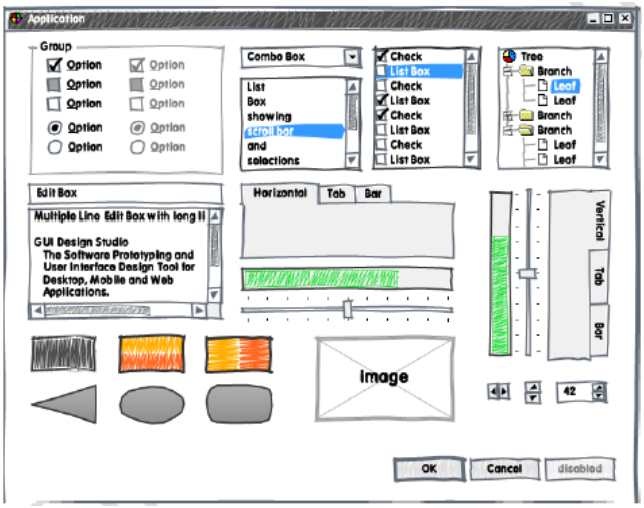
1. Tkinter เป็นโมดูลที่พัฒนามาจาก Tk GUI Toolkit ซึ่งทางานอยู่บนระบบปฏิบัติการยูนิกซ์มาก่อน ไพธอนได้เลือกโมดูลนี้ในการพัฒนากราฟิกบนไพธอนเป็นหลัก

2. wxPython เป็นโมดูลที่ถูกพัฒนาขึ้นเพื่อให้ทางานอยู่บนระบบปฏิบัติการวินโดวส์เป็นหลัก และเป็นลิขสิทธิ์แบบ Open source

3. JPython เป็นโมดูลที่พัฒนาขึ้นเพื่อให้ไพธอนสามารถทำงานร่วมกับกราฟิกของภาษาจาวาได้

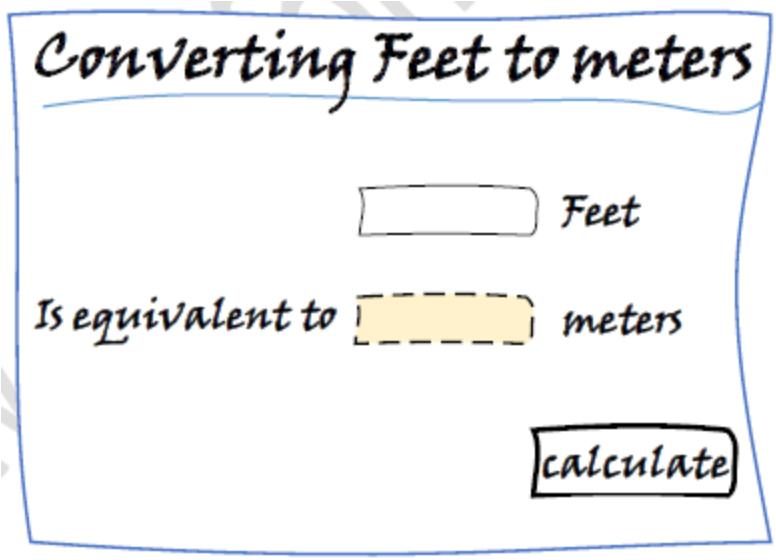
1. แนวความคิดในการออกแบบ GUI

ก่อนการพัฒนา GUI ด้วยไพธอน จะต้องออกแบบหน้าตาของ GUI เสียก่อน โดยการสเก็ตช์ภาพบนกระดาษ หรือออกแบบด้วยโปรแกรมสำหรับสร้างงานกราฟิกด้วยคอมพิวเตอร์อย่างคร่าวๆ เสียก่อน ดังรูป



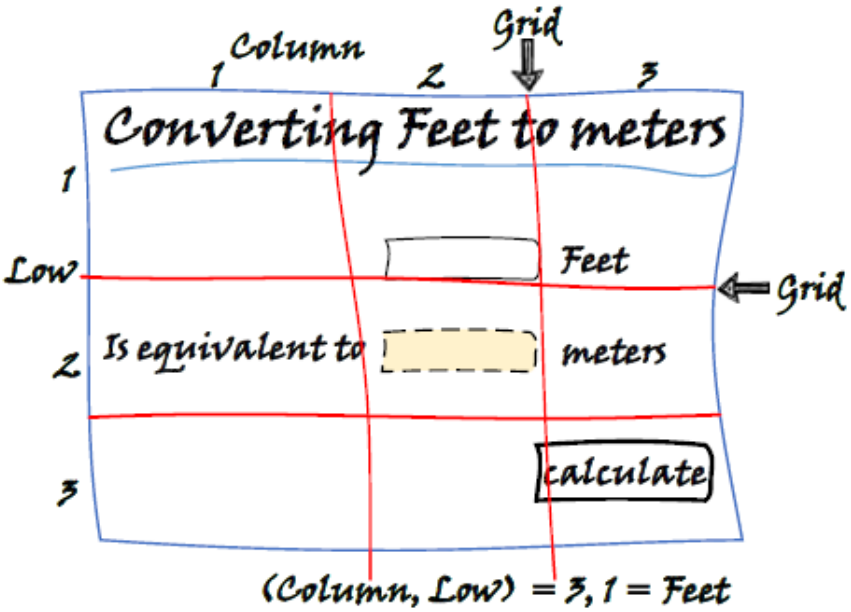
รูปที่ 9.1 การสเก็ตช์ GUI อย่างคร่าวๆ บนกระดาษ

ยกตัวอย่างการเขียนโปรแกรมแปลงค่าหน่วยวัดระยะทางจากฟุต (feet) เป็นเมตร (meters)



รูปที่ 9.2 แสดงภาพสเก็ตช์ของโปรแกรมแปลงจาก feet เป็น meters

จากรูปที่ 9.2 ผู้ใช้จะป้อนตัวเลขจำนวนจริงในช่อง Feet เมื่อกดปุ่ม calculate โปรแกรมจะทำการแปลงค่าจำนวนจริงที่ป้อนให้กับโปรแกรมเป็นเมตรและแสดงใน meters ขั้นตอนต่อไปให้ผู้เขียนโปรแกรมทำการวาดเส้นกริดเพื่อใช้กำหนดขอบเขตสำหรับจัดวางตำแหน่ง (Layout) ของอ๊อปเจ็กต์ต่างๆ ในโปรแกรม จากตัวอย่างด้านบน จะทำการกำหนดเส้นกริดเป็น 3 แถว 3 คอลัมน์ดังรูปที่ 9.3

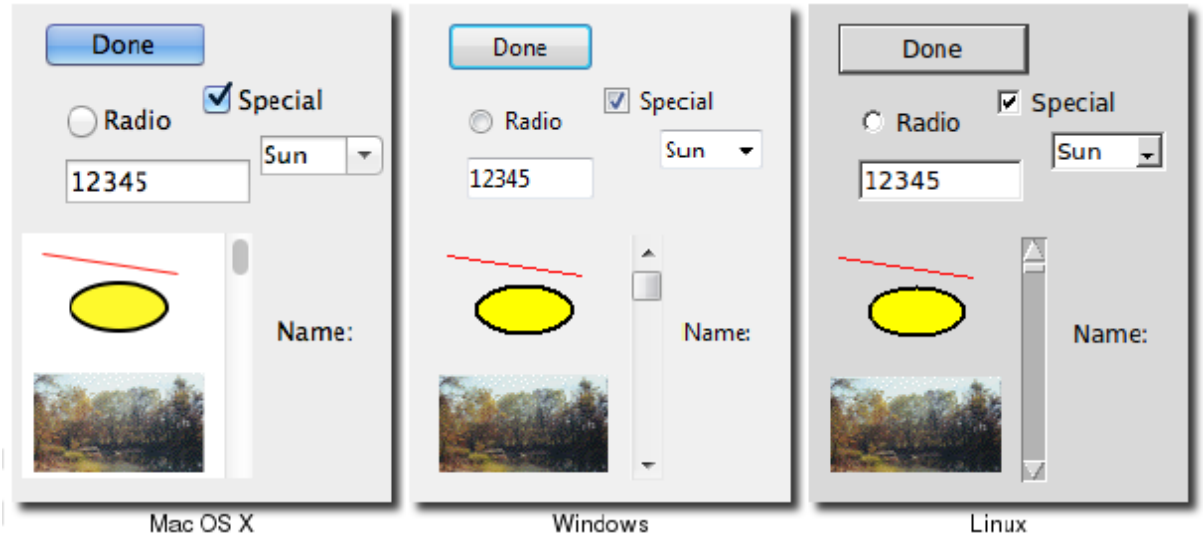


รูปที่ 9.3 แสดงการวาดเส้นกริด 3 x 3 ลงบนภาพสเก็ตช์

2. การสร้าง GUI ด้วย Tk

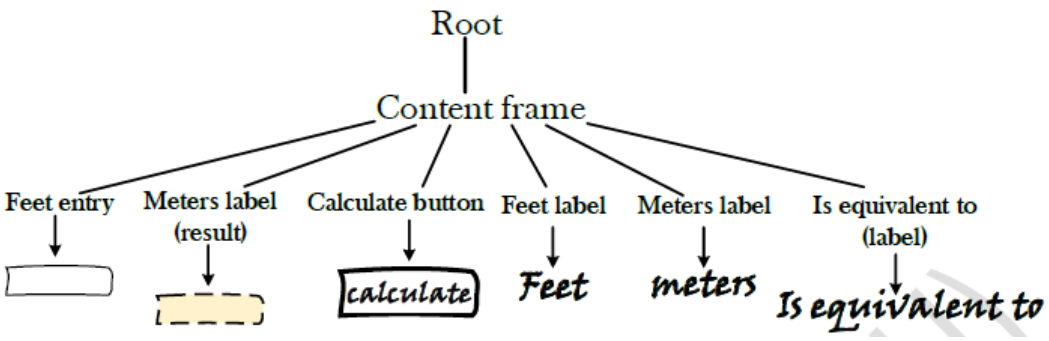
ก่อนการสร้างกราฟิกจาก Tk หรือ Tkinter ผู้เขียนโปรแกรมจำเป็นต้องเข้าใจถึงแนวความคิดและหลักการทำงานของ Tk เสียก่อน เพื่อเป็นพื้นฐานในการสร้าง GUI ที่สมบูรณ์ต่อไป Tk ประกอบไปด้วย 3 ส่วนที่สำคัญคือ widgets, geometry management และ event handling ซึ่งมีรายละเอียดดังนี้

2.1 Widgets คือสิ่งต่างๆ (หรือเรียกว่า อ๊อปเจ็กต์) ที่ปรากฏอยู่บนจอภาพ เช่น Button, Label, Frame, checkbox, tree views, scrollbars, text areas เป็นต้น ดังรูปที่ 9.4



รูปที่ 9.4 แสดงตัวอย่าง Widgets บนระบบปฏิบัติการต่างๆ

Widgets ต่างๆ เหล่านี้ถูกออกแบบให้มีลักษณะการทำงานแบบลำดับชั้น นั่น คือ การสร้าง GUI ใดๆ ให้ปรากฏบนจอภาพ จะเริ่มต้นสร้างจากลำดับชั้น ที่เรียกว่า root window (root) ขึ้นเสียก่อน ดังรูปที่ 9.5 ลำดับชั้นที่ 2 จึงสร้างเฟรม (Content frame) หรือผืนผ้า (Canvas) เพื่อบรรจุ widgets ต่างๆ ลงบน root window ในลำดับชั้นที่ 3 เป็นการเพิ่ม Widgets ต่างๆ ที่ได้ออกแบบไว้บนกระดาษลงบนเฟรมหรือ Canvas ที่ได้จัดเตรียมไว้ สำหรับการควบคุมการทำงานของWidgets จะเป็นแบบลำดับชั้นเช่นเดียวกัน



รูปที่ 9.5 แสดงลำดับของการสร้าง GUI ด้วย Tk

จากรูปที่ 9.5 แสดงลำดับชั้นการสร้าง GUI ของโปรแกรมแปลงหน่วยวัดระยะทางจากฟุตเป็นเมตร โดยเริ่มต้นจากการสร้างหน้าต่างหลัก (Root) ขึ้นก่อนเสมอ เพื่อรองรับ Widgets ที่ทำหน้าที่ต่างๆ กัน โดย Widgets จะถูกสร้างขึ้นลงบนเฟรม (Content frame) อีกที ซึ่งเฟรมดังกล่าวเปรียบเสมือนหน้ากระดาษบนคอมพิวเตอร์นั่นเอง

2. การเรียกใช้งาน Widgets โดย Widgets ในภาษาไพธอนถูกเขียนขึ้นด้วยโปรแกรมเชิงวัตถุ ดังนั้นทุกๆ Widget ที่สร้างขึ้นจะถูกเรียกว่า อ๊อปเจ็กต์ (Object) หรือวัตถุ เมื่อทำการสร้างอินสแตนซ์ของ Widget ใดๆ ขึ้นจะต้องส่งพารามิเตอร์ให้กับคลาสแม่ตามลำดับชั้นตามที่กล่าวมาแล้ว ยกเว้น Root ซึ่งเป็นคลาสแม่ที่อยู่ในตำแหน่งบนสุด (Top level window) ของลำดับชั้น คลาสลูกทุกๆ คลาสจะถูกสร้างภายใน root เท่านั้น จากตัวอย่างต่อไปนี้เป็นตัวอย่างการสร้าง Root, Content frame และ Widgets

```
root = Tk() #Create root window
content = Frame(root) #Create content frame
button = Button(content) #Create button in frame
```

Widget แต่ละประเภทมีหน้าที่การทำงานที่ต่างกัน ดังนั้นเมื่อสร้าง Widget ขึ้นมาแล้วจำเป็นที่จะต้องการทำการปรับแต่งคุณสมบัติของ Widgets (Configuration option) แต่ละตัวให้เหมาะสมกับงานที่จะทำ ตัวอย่างเช่น Label และ Button สามารถกำหนดข้อความที่แสดงผลด้วยออฟชัน text = "Textname" ในขณะที่ scrollbar ไม่จำเป็นต้องใช้ออฟชันดังกล่าว และถ้ามีการกดปุ่ม Button จะต้องเรียกใช้ฟังก์ชันหรือเมธอดที่เหมาะกับงานดังกล่าวโดยใช้คำสั่ง command แต่สำหรับ Label ไม่จำเป็นต้องใช้คำสั่ง command ในการทำงาน เป็นต้น

Configuration option จะถูกกำหนดในขณะที่ทำการสร้าง Widget โดยการส่งเป็นพารามิเตอร์ที่เหมาะสมให้กับแต่ละ Widget ที่จะถูกสร้างขึ้น ตัวอย่างเช่น เมื่อต้องการสร้างปุ่ม Button โดยปุ่มดังกล่าวประกอบด้วยชื่อของปุ่ม และเหตุการณ์ที่เกิดขึ้นหลังจากมีการกดปุ่มดังกล่าว ดังนี้

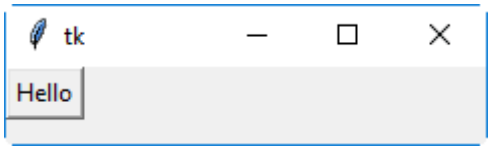
1. นำเข้าโมดูล tkinter

```
from tkinter import *
root = Tk() #Create root window
```

2. สร้างปุ่ม Button โดยส่งพารามิเตอร์ให้ 2 ค่า คือ ข้อความที่แสดงบนปุ่ม (text = "Hello") และวิธีการตอบสนองเมื่อปุ่มดังกล่าวถูกกด (command = "buttonpressed") พร้อมกับกำหนดค่ากริด

```
from tkinter import *
root = Tk() #Create root window
button = Button(root, text="Hello", command="buttonpressed")
button.grid()
```

ผลลัพธ์ที่ได้ คือ



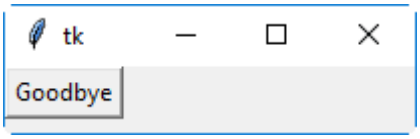
3. ทดสอบเปลี่ยนข้อความที่แสดงบนปุ่มจาก "Hello" เป็น "Goodbye"

```
from tkinter import *
root = Tk() #Create root window
button = Button(root, text="Hello", command="buttonpressed")
button.grid()
button['text'] = 'Goodbye'
```

หรือใช้เมธอด configure() เพื่อเปลี่ยนข้อความบนปุ่มแทนก็ได้

```
from tkinter import *
root = Tk() #Create root window
button = Button(root, text = "Hello", command = "buttonpressed")
button.grid()
button.configure(text = 'Goodbye')
```

ผลลัพธ์ที่ได้ คือ



3. การจัดการรูปทรงเรขาคณิตให้กับ Widgets (Geometry management) จากที่กล่าวมาแล้วข้างต้นว่าการวาง Widgets ลงบนเฟรมนั้น จะต้องกำหนดตำแหน่งในการวาง โดยอาศัยศาสตร์ทางด้านเรขาคณิตเข้าช่วย เพื่อให้ Widgets ที่จะวางอยู่ในตำแหน่งที่เหมาะสม ซึ่งไพธอนมี 3 เมธอดในการจัดการเกี่ยวกับเรขาคณิตของ Widgets ประกอบไปด้วยเมธอด pack(), grid() และ place() ซึ่งจะกล่าวในหัวข้อการจัดวาง Widgets ด้วยเรขาคณิต

สำหรับปัญหาการวาง Widgets ที่เกิดขึ้นเสมอ คือ การปรับขนาดของเฟรมหรือ Widgets จะส่งผลกระทบซึ่งกันและกัน เช่น ถ้าผู้ใช้งานย่อหรือขยายขนาดของหน้าต่างหลัก จะส่งผลให้กระทบกับอ๊อปเจ็กต์ต่างๆ ที่อยู่ภายในหน้าต่างหลักนั้นๆ ทันที ซึ่งอาจจะทำให้ปุ่ม ตัวอักษร ลาเบล เกิดความผิดเพี้ยนไปจากเดิม ปัญหาต่างๆ เหล่านี้จะถูกจัดการด้วย Geometry management ที่อยู่ใน Tk โดยใช้เทคนิคที่เรียกว่า Master and Slave โดย Master คืออ๊อปเจ็กต์ที่ทำหน้าที่รองรับ Widgets ต่างๆ ที่จะทำงาน เช่น root หรือ content frame สำหรับ Slave คือ Widgets ต่างๆ ที่วาดหรือวางลงบน Master นั่นเอง

สำหรับการทำงานของ Geometry management นั้นจะใช้วิธีสอบถามไปยัง Widgets ต่างๆ ที่กำลังจะทำงานว่าแต่ละ Widgets ต้องการพื้นที่ๆ ใช้สำหรับการทำงานมากน้อยเพียงใด จากนั้น Geometry management จะคำนวณพื้นที่ทั้งหมดในภาพรวม เพื่อจัดวาง Widgets เหล่านั้นในตำแหน่งที่เหมาะสมต่อไป จากตัวอย่าง เมื่อผู้ใช้ต้องการเพิ่ม Widget อันใดอันหนึ่งลงบนเฟรมจะเรียกใช้เมธอด grid() และตามด้วยพารามิเตอร์คอลัมน์ (Column) และแถว (Low) ในการกำหนดตำแหน่งการวาง พร้อมกับพารามิเตอร์ "sticky" เพื่อบอกว่าให้วาง Widget ตามขนาดจริงที่ปรากฏ เช่น ความกว้างและยาวของปุ่มจะมีขนาดตามฟอนต์ที่ปรากฏบนปุ่ม เป็นต้น หรืออาจจะใช้เมธอด columnconfigure() หรือ rowconfigure() เพื่อปรับขนาดของ Widget ตามที่ผู้ใช้งานต้องการก็ได้

4. การจัดการกับเหตุการณ์ต่างๆ (Event Handling) Event handling คือ เหตุการณ์ต่างๆ ที่ผู้ใช้งานกระทำกับ Widgets ใดๆ บน GUI เช่น การกดปุ่ม การกดปุ่มใดๆ บนแป้นพิมพ์ การเคลื่อนเมาส์ การปรับขนาดของหน้าต่างวินโดวส์ เป็นต้น ซึ่งเหตุการณ์ต่าง ๆ เหล่านี้จะถูกจัดการโดย Tk ซึ่งเรียกว่า event loop โดยจะทำงานร่วมกับระบบปฏิบัติการโดยตรง เช่น เมื่อเคลื่อนเมาส์ไปยังปุ่มจะส่งผลให้ปุ่มดังกล่าวจะเปลี่ยนสี และเมื่อเคลื่อนเมาส์ออกจากปุ่มจะทำให้สีของปุ่มกลับไปเป็นสีเดิม เป็นต้น

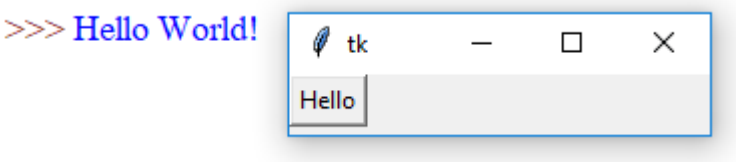
5. การตอบสนองต่อเหตุการณ์ที่เกิดขึ้น (Command Callbacks) มีหลาย Widgets จำเป็นต้องกระทำอย่างใดอย่างหนึ่งเมื่อมีการคลิกหรือกระทำกับ Widgets เหล่านั้น เช่น เมื่อกดปุ่ม Save as… จะส่งผลให้มีการเปิดหน้าต่างวินโดวส์ เพื่อให้ผู้ใช้เลือกไดเรคทอรีที่ต้องการบันทึกผลลงฮาร์ดดิสก์ เป็นต้น ในไพธอนจะใช้คำสั่ง "command" หรือเรียกว่า "Callbacks" ในการตอบสนองต่อเหตุการณ์ต่างๆ ที่เกิดขึ้นกับ Widgets โดยมีรูปแบบคำสั่งคือ

command = functionName

ตัวอย่างเช่น เมื่อคลิกปุ่ม Hello แล้วไปยังฟังก์ชัน helloCallBack เพื่อพิมพ์คำว่า Hello World!

```
from tkinter import *
root = Tk() #Create root window

def helloCallBack():
   print("Hello World!")
btn1 = Button(root, text="Hello", command=helloCallBack)
btn1.grid()
```

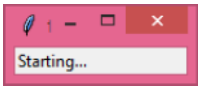
ผลลัพธ์ที่ได้ คือ



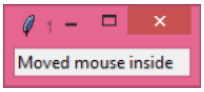
6. การผูกเหตุการณ์ต่างๆ เข้ากับไพธอน Widgets มีคำสั่ง "command" ซึ่งมาพร้อมกับขณะที่มีการสร้าง Widget อยู่แล้ว เพื่อผูกเหตุการณ์ต่างๆ เข้ากับการกระทำอย่างใดอย่างหนึ่งในโปรแกรม แต่ผู้ใช้สามารถเรียกใช้คำสั่ง bind เพื่อเป็นทางเลือกในการดักจับเหตุการณ์ต่างๆ เหมือนกับการใช้คำสั่ง command ได้เช่นเดียวกัน ดังตัวอย่างต่อไปนี้จะแสดงการจัดการกับ Widget ชนิด Label ด้วยคำสั่ง bind ดังนี้

```
from tkinter import *
root = Tk() #Create root window
label1 = Label(root, text="Starting...")
label1.grid()
label1.bind('<Enter>', lambda e: label1.configure(text='Moved mouse inside'))
label1.bind('<Leave>', lambda e: label1.configure(text='Moved mouse outside'))
label1.bind('<1>', lambda e: label1.configure(text='Clicked left mouse button'))
label1.bind('<Double-1>', lambda e: label1.configure(text='Double clicked'))
label1.bind('<B3-Motion>', lambda e: label1.configure(text='right button drag to %d,%d' % (e.x, e.y)))
root.mainloop()
```

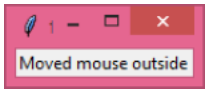
ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม คือ



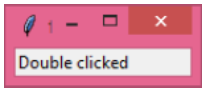
เริ่มโปรแกรม



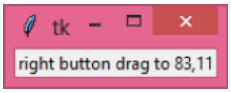
เคลื่อนเมาส์ไปในวินโดวส์



เคลื่อนเมาส์ออกจากวินโดวส์



ดับเบิลคลิกเมาส์ในวินโดวส์



คลิกขวาและลากเมาส์

จากภาพด้านบนแสดงการทำงานของโปรแกรม โดยเริ่มจากบรรทัดที่ 2 โปรแกรมทำการสร้างหน้าต่างหลัก (root) โดยเรียกคลาส Tk() ขึ้นมาทำงาน จากนั้นบรรทัดที่ 3 โปรแกรมทำการสร้าง Label ลงบนหน้าต่างหลัก โดยมีพารามิเตอร์ 2 ตัวคือ หน้าต่างหลักที่จะเพิ่ม Label ลงไป ในที่นี้คือ root และข้อความที่จะให้ปรากฏบน Label คือ text="Starting..." บรรทัดที่ 4 โปรแกรมเรียกใช้เมธอดกริด (เมื่อไม่กำหนดคอลัมน์และแถวจะมีค่าเป็น 1 คอลัมน์ และ 1 แถว) บรรทัดที่ 6 ทำการผูกเหตุการณ์เมื่อผู้ใช้มีการกระทำใดๆ บน Label ดังกล่าวด้วยคำสั่ง bind โดยจะทำงานก็ต่อเมื่อผู้ใช้เคลื่อนเมาส์เข้าไปในวินโดวส์ โดยจะพิมพ์ข้อความว่า "Move mouse inside" บรรทัดที่ 6, 7, 8 และ 9 จะทำงานก็ต่อเมื่อผู้ใช้เคลื่อนเมาส์ออกจากวินโดวส์ คลิกเมาส์ซ้าย ดับเบิ้ลคลิก และการคลิกพร้อมกับลากเมาส์ตามลำดับ ผลลัพธ์แสดงดังรูปด้านบน ส่วนบรรทัดที่ 10 โปรแกรมจะทำการวนลูปเพื่อรอรับเหตุการณ์ต่างๆ ของผู้ใช้งานไปเรื่อยๆ จนกว่าจะปิดโปรแกรม

7. สรุปขั้นตอนการสร้าง GUI ด้วย tkinter มีขั้นตอนดังนี้

1. นำเข้าโมดูล Tkinter ด้วยคำสั่ง from tkinter import *

2. สร้างหน้าต่าง GUI หลัก ด้วยคำสั่ง root = Tk()

3. เพิ่มWidgets หรืออ๊อปเจ็กต์ลงในหน้าต่างหลัก เช่น Button, Canvas หรือ Label เป็นต้น

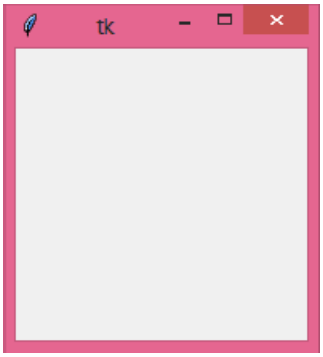
4. เขียนโปรแกรมสำหรับตรวจสอบเหตุการณ์ต่างๆ (Events) ที่เกิดขึ้นจากผู้ใช้งานกระทำกับอ๊อปเจ็กต์ใดๆ เช่น การคลิก การลาก การปล่อย เป็นต้น

5. สั่งให้โปรแกรมวนลูปรับคำสั่งจากผู้ใช้งานไปเรื่อยๆ จนกว่าจะปิดโปรแกรม ด้วยคำสั่ง root.mainloop()

- ลำดับการสร้างหน้าต่าง GUI ด้วย Tk

```
from tkinter import *
root = Tk() #Create root window
# Code to add widgets will go here...
root.mainloop()
```

- แสดงหน้าต่างหลัก (root window)



3. การจัดวาง Widgets ด้วยเรขาคณิต (Geometry management)

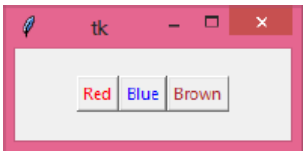
ทุกๆ Widgets จะต้องอาศัยเรขาคณิตช่วยในการจัดวาง (Geometry management) ซึ่งไพธอนได้จัดเตรียมไว้ 3 เมธอดคือ pack(), grid() และ place() ดังนี้

3.1 เมธอด pack() ทำหน้าที่จัดวาง Widgets ให้อยู่ในกลุ่ม (block) ก่อนวางลงใน Widget แม่ ซึ่งมีรูปแบบคำสั่งดังนี้

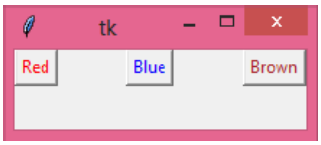
```
widget.pack( pack_options )
```

pack_options มี 3 รูปแบบคือ expand, fill และ side

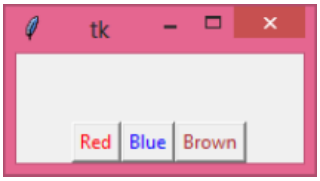
1) expand เมื่อกำหนดให้เป็น True หรือ YES โปรแกรมจะขยายขนาด Widget โดยการเพิ่มช่องว่างเข้าไปแทน และอยู่ตรงกลางหน้าต่างหลัก เช่น widget.pack(expand = True)



2) fill กำหนดให้ขยายขนาดของ Widget ตามขนาดจริงที่น้อยที่สุด หรือสามารถกำหนดเป็น Y (vertically : ขยายทางแนวตั้ง) หรือกำหนดเป็น X (horizontally : ขยายทางแนวนอน) หรือทั้งคู่ก็ได้ แต่ค่าดีฟอลต์ (default) เป็น NONE เช่น fill = X, fill = Y หรือ fill = BOTH ตัวอย่างเช่น widget.pack(fill = BOTH)



3) side กำหนดตำแหน่งในการวาง Widget คือ TOP (ดีฟอลต์) ด้านบน, BOTTOM ด้านล่าง, LEFT ด้านซ้าย และ RIGHT ด้านขวา เช่น side = BOTTOM เช่น widget.pack(side = BOTTOM)



ตัวอย่างการใช้งานเมธอด pack() ดังนี้

```
from tkinter import *
root = Tk()


frame = Frame(root, bd="3", relief=GROOVE, padx=10, pady=10)
frame.pack()
redbutton = Button(frame, text="Red", fg="red")
redbutton.pack(side = LEFT)
greenbutton = Button(frame, text="Brown", fg="brown")
greenbutton.pack(side = RIGHT)
bluebutton = Button(frame, text="Blue", fg="blue")
bluebutton.pack(side = BOTTOM)
bottomframe = Frame(root, bd="3", relief=GROOVE, padx=10, pady=10)
bottomframe.pack(side = BOTTOM)
blackbutton = Button(bottomframe, text="Black", fg="black")
blackbutton.pack(side = BOTTOM)


root.mainloop()
```

ผลลัพธ์ที่ได้ คือ

จากตัวอย่างโปรแกรม แสดงการใช้เมธอด pack() ในการจัดวาง Widgets โดยโปรแกรมสร้างเฟรม (บรรทัดที่ 4) ด้วยเมธอด Frame(root) อ๊อปเจ็กต์ที่สร้างขึ้นจะเก็บไว้ในตัวแปรชื่อ frame จากนั้นบรรทัดที่ 5 โปรแกรมเรียกเมธอด pack() โดยไม่มีพารามิเตอร์ ซึ่งส่งผลให้เฟรมดังกล่าวจะวางอยู่ในตำแหน่งบนสุดของหน้าต่างหลัก (root window) บรรทัดที่ 6 – 11 โปรแกรมสร้างปุ่มสีแดง น้ำเงิน และน้ำตาลลงบนเฟรมดังกล่าว

บรรทัดที่ 13 และ 14 โปรแกรมสร้างเฟรมชื่อ bottomframe โดยวางอยู่ในตำแหน่งด้านล่างของหน้าต่างหลัก บรรทัดที่ 15 และ 16 โปรแกรมสร้างปุ่มสีดำ และวางลงในเฟรมชื่อ bottomframe

2. เมธอด grid() ทำหน้าที่จัดวาง Widgets ลงในหน้าต่างหลัก (root window) โดยอยู่ในรูปแบบของตาราง ซึ่งมีรูปแบบคำสั่ง ดังนี้

widget.grid(grid_options)

grid_options มีรูปแบบ ดังนี้คือ

1. column ตำแหน่งคอลัมน์ที่ต้องการวาง Widget ค่าดีฟอลต์คือ 0 (คอลัมน์ซ้ายสุด)

2. columnspan จำนวนคอลัมน์ที่ต้องการใช้ ค่าดีฟอลต์คือ 1 บรรทัด

3. ipadx, ipady ขนาดภายในขอบของ Widget ในแนวตั้งและแนวนอน มีหน่วยเป็นพิกเซล

4. padx, pady ขนาดภายนอกขอบของ Widget ในแนวตั้งและแนวนอน มีหน่วยเป็นพิกเซล

5. row ตำแหน่งแถวที่ต้องการวาง Widget

6. rowspan จำนวนแถวที่ต้องการใช้ ค่าดีฟอลต์คือ 1 คอลัมน์

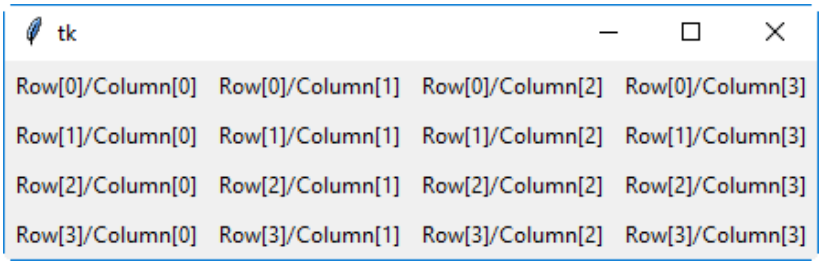
7. sticky ใช้เมื่อต้องการวาง Widget ในตำแหน่งที่ต้องการ โดยค่าดีฟอลต์จะอยู่ตรงกลาง แต่สามารถกำหนดตำแหน่งด้วยการบอกทิศทาง โดยใช้อักษรต่างๆ ดังนี้ N (ทิศเหนือ), E (ตะวันออก), S (ใต้),

W (ตะวันตก), NE (ทิศตะวันออกเฉียงเหนือ), NW (ทิศตะวันตกเฉียงเหนือ), SE (ทิศตะวันออกเฉียงใต้) และ SW (ทิศตะวันตกเฉียงใต้) เป็นต้น

ตัวอย่างการใช้งานเมธอด grid() ดังนี้

```
from tkinter import *
root = Tk( )
for r in range(4):
   for c in range(4):
      Label(root, text='Row[%s]/Column[%s]'%(r,c), borderwidth=5).grid(row=r,column=c)
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูป



จากตัวอย่างโปรแกรม แสดงการใช้เมธอด grid() ในการจัดวาง Widgets โดยบรรทัดที่ 5 โปรแกรมสร้างเลเบลที่ทำการพิมพ์ตำแหน่งแถวและคอลัมน์ลงบนหน้าต่างหลัก โดยใช้เมธอด grid() ในการกำหนดตำแหน่งของเลเบล ผลลัพธ์ที่ได้แสดงดังรูปด้านบน

3. เมธอด place() ทำหน้าที่จัดวาง Widgets ลงในหน้าต่างหลัก (root window) โดยการระบุตำแหน่ง ซึ่งมีรูปแบบคำสั่ง ดังนี้

widget.place(place_options)

place_options มีรูปแบบดังนี้ คือ

1. anchor กำหนดตำแหน่งการวาง Widget โดยอาศัยทิศ เช่น N (ทิศเหนือ), E (ตะวันออก), S, W, NE, NW, SE หรือ SW เป็นต้น

2. bordermode กำหนดตำแหน่งการวาง Widget โดยอาศัยขอบด้านใน (INSIDE) และขอบด้านนอก (OUTSIDE) ค่าดีฟอลต์คือ INSIDE

3. height, width กำหนดขนาดความกว้างและความยาว มีหน่วยเป็นพิกเซล

4. relheight, relwidth กำหนดขนาดความกว้างและความยาว โดยใช้เลขจำนวนจริงระหว่าง 0.0 – 1.0

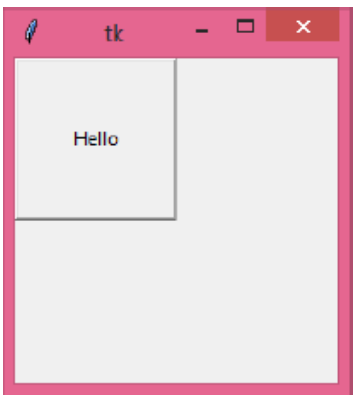
5. relx, rely ขนาดแนวตั้งและแนวนอนของ offset โดยใช้เลขจำนวนจริงระหว่าง 0.0 – 1.0

6. x, y ขนาดแนวตั้งและแนวนอนของ offset มีหน่วยเป็นพิกเซล

ตัวอย่างการใช้งานเมธอด place() ดังนี้

```
from tkinter import *
import tkinter.messagebox
root = Tk( )

def helloCallBack():
   tkinter.messagebox.showinfo("Hello Python", "Hello World")

B = tkinter.Button(root, text ="Hello", command = helloCallBack)
B.pack()
B.place(bordermode=OUTSIDE, height=100, width=100)

root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



จากตัวอย่างโปรแกรม แสดงการใช้เมธอด place() ในการจัดวาง Widgets โดยบรรทัดที่ 6 โปรแกรมสร้างเมธอดชื่อว่า helloCallBack() ทำหน้าที่แสดงกล่องข้อความ (messagebox) โดยพิมพ์ข้อความว่า "Hello World" เมธอดดังกล่าวนี้จะถูกเรียกใช้เมื่อมีการกดปุ่ม Hello บรรทัดที่ 8 โปรแกรมสร้างปุ่มซึ่งมีข้อความบนปุ่มคือ "Hello" โดยมีพารามิเตอร์ 3 ตัวคือ อ๊อปเจ็กต์ของหน้าต่างหลัก (root), ข้อความที่ต้องการแสดงบน

ปุ่ม ("Hello") และการกระทำ (action) เมื่อปุ่มถูกกด ในที่นี้โปรแกรมจะเรียกเมธอด helloCallBack() มาทำงาน บรรทัดที่ 10 โปรแกรมจะวางปุ่มดังกล่าวนอกขอบ (border = OUTSIDE) ที่เมธอด pack() ได้กำหนดไว้ โดยปุ่มมีขนาดความกว้างและความยาวเท่ากับ 100 พิกเซล

4. คุณสมบัติพื้นฐานของ Widgets

Widgets ต่างๆ ที่ใช้สำหรับออกแบบ GUI มีคุณสมบัติพื้นฐานหลายประการที่สามารถใช้งานร่วมกันได้ เช่น ขนาด สี และฟอนต์ เป็นต้น ซึ่งมีรายละเอียดของคุณสมบัติต่างๆ ดังนี้

1. Dimension (ขนาด): ขนาดของ Widgets สามารถกำหนดได้หลายแบบดังนี้

- กำหนดขนาดด้วยตัวอักษร เช่น 'c' = เซนติเมตร (Centimeters), 'i' = นิ้ว (Inches), 'm' = มิลลิเมตร (Millimeters), 'p' = Printer's points (ประมาณ 1/72)

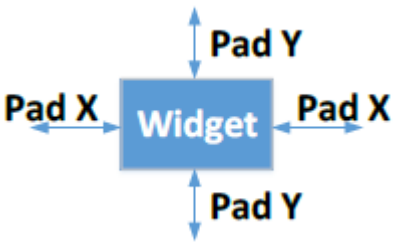
- กำหนดขนาดด้วยตัวเลข มีหน่วยเป็นพิกเซล (Pixels) เช่น 1, 3, หรือ 5 เป็นต้น

Dimension ถูกนำไปใช้กำหนดคุณสมบัติของ Widgets ดังต่อไปนี้

- **borderwidth** ความกว้างเส้นขอบของ Widget

- **highlightthickness** ความกว้างของรูปสี่เหลียมที่ใช้สำหรับเน้นความสนใจ (highlight rectangle) เมื่อ Widget ถูกโฟกัส

- **padX padY** ขนาดพื้นที่ว่างเพิ่มเติมรอบๆ Widget



- **selectborderwidth** ความกว้างเส้นขอบเมื่อ Widget ถูกเลือก

- **wraplength** ความยาวสูงสุดที่เกิดจากการครอบคำหรือข้อความ

- **height** กำหนดความสูงของ Widget

- **width** ความกว้างของ Widget

- **underline** ขีดเส้นใต้ตัวอักษรที่ต้องการ (0 คืออักษรตัวแรกของข้อความ)

2. Color (สี) การกำหนดสีให้กับตัวอักษรหรือ Widgets ได้ 2 รูปแบบคือ

- กำหนดค่าของสีโดยใช้เลขฐาน 16 ตัวอย่างเช่น #ffff คือสีขาว, #0000 คือสีดำ หรือ #00ffff คือสีฟ้า เป็นต้น

- กำหนดค่าของสีโดยใช้ข้อความ เช่น "red" = สีแดง, "green" = สีเขียว หรือ "white" = สีขาว เป็นต้น ดังรูปด้านล่าง

Color ถูกนำไปใช้กำหนดคุณสมบัติของ Widgets ดังต่อไปนี้

- **activebackground** กำหนดสีพื้นหลังของ Widgets เมื่อ Widgets ทำงาน (Active)

- **activeforeground** กำหนดสีด้านหน้าของ Widgets เมื่อ Widgets ทำงาน (Active)

- **background** กำหนดสีพื้นหลังของ Widgets หรือเขียนย่อเป็น bg

- **foreground** กำหนดสีด้านหน้าของ Widgets หรือเขียนย่อเป็น fg

- **disabledforeground** กำหนดสีด้านหน้าของ Widgets เมื่อ Widgets ถูก Disable

- **highlightbackground** กำหนดสีพื้นหลังเมื่อ Widgets ถูกโฟกัส

- **highlightcolor** กำหนดสีด้านหน้าเมื่อ Widgets ถูกโฟกัส

- **selectbackground** กำหนดสีพื้นหลังเมื่อมีการเลือกรายการใดรายการหนึ่งใน Widget เช่น เลือกรายการจาก Dropdown

- **selectforeground** กำหนดสีด้านหน้าเมื่อมีการเลือกรายการใดรายการหนึ่งใน Widget

3. Font (ฟอนต์) การกำหนดรูปแบบของฟอนต์ให้กับ Widgets สามารถกำหนดได้ 2 รูปแบบ คือ

- กำหนดฟอนต์โดยใช้เครื่องหมายวงเล็บครอบ เช่น ("Helvetica", "16") หรือ ("Times", "24", "bold italic")

- กำหนดฟอนต์โดยใช้เมธอด font ที่มากับ Tk ซึ่งมีรูปแบบคือ

myFont = font.Font(option, ...)

สำหรับ option มีรายละเอียดดังนี้

- **family** ชื่อของฟอนต์ เช่น 'Helvetica', 'Verdana', 'Times' ตัวอย่าง family = 'Helvetica'

- **size** ขนาดของฟอนต์ เช่น size = 15

- **weight** ขนาดความหนาของฟอนต์ เช่น bold คือฟอนต์ตัวหนา, normal คือ ความหนาปกติ ตัวอย่าง weight = 'bold'

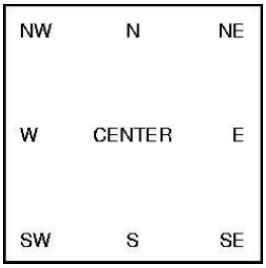
- **slant** ตัวอักษรเอียง เช่น italic คือตัวอักษรเอียง, roman คือ ตัวอักษรปกติ

- **underline** ขีดเส้นใต้ เช่น underline = 1 คือการขีดเส้นตัวอักษร, 0 คือ ตัวอักษรปกติ

- **overstrike** ขีดเส้นทับตัวอักษร เช่น overstrike = 1 คือขีดเส้นทับตัวอักษร, 0 คือไม่ขีดเส้นทับ

4. Anchors กำหนดจุดอ้างอิงที่ใช้สำหรับจัดวาง Widgets มีรูปแบบคือ NW: ทิศตะวันตกเฉียงเหนือ, N: ทิศเหนือ, NE: ทิศตะวันออกเฉียงเหนือ, W: ทิศตะวันตก, CENTER: จุดกลางของแผนที่, E: ทิศตะวันออก, SW: ทิศตะวันตกเฉียงใต้, S: ทิศใต้, SE: ทิศตะวันออกเฉียงใต้

ตัวอย่างเช่น anchor = NE, anchor = SE สำหรับตัวอย่างตำแหน่งทิศต่างๆ แสดงดังรูป



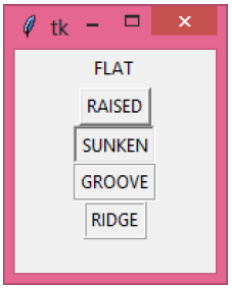
5. Relief styles กำหนดลักษณะภาพในรูปแบบ 3 มิติ (3D) รอบๆ บริเวณ Widgets มีรูปแบบคือ

- **FLAT** แสดงภาพในลักษณะแบนราบ (ไม่เป็นภาพนูน)
- **RAISED** แสดงภาพในลักษณะยกขึ้น หรือนูนขึ้น
- **SUNKEN** แสดงภาพในลักษณะจมลงไป
- **GROOVE** แสดงภาพในลักษณะกรอบเป็นร่องลึก
- **RIDGE** แสดงภาพในลักษณะกรอบนูนขึ้น

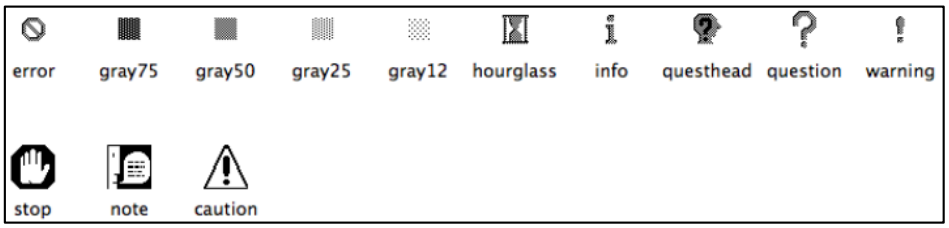
ตัวอย่างการใช้งาน Relief แสดงในโปรแกรม

```
from tkinter import *
root = Tk()

B1 = Button(root, text ="FLAT",relief=FLAT)
B2 = Button(root, text ="RAISED",relief=RAISED)
B3 = Button(root, text ="SUNKEN",relief=SUNKEN)
B4 = Button(root, text ="GROOVE",relief=GROOVE)
B5 = Button(root, text ="RIDGE",relief=RIDGE)
B1.pack()
B2.pack()
B3.pack()
B4.pack()
B5.pack()
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



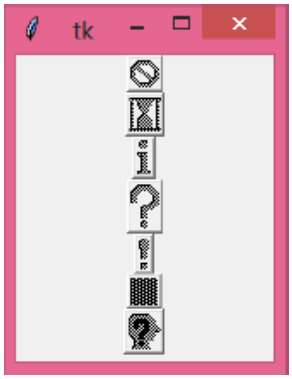
6. Bitmaps (บิตแมป) Bitmap คือภาพที่เกิดจากจุดสีที่เรียกว่า Pixel (พิกเซล) ประกอบกันเป็นรูปร่างบนพื้นที่ที่มีลักษณะเป็นเส้นตาราง (กริด) แต่ละพิกเซลจะมีค่าของตำแหน่ง และค่าสีของตัวเอง ภาพหนึ่งภาพ จะประกอบด้วยพิกเซลหลายๆ พิกเซลผสมกัน ไฟล์ภาพเหล่านี้มีหลายรูปแบบ อาทิ เช่น BMP, TIF, JPG, PCT เป็นต้น ไพธอนมีภาพบิตแมปให้เลือกใช้งานดังรูป



ตัวอย่างการใช้งาน Bitmaps

```
from tkinter import *


root = Tk()
B1 = Button(root, relief=RAISED, bitmap="error")
B2 = Button(root, relief=RAISED, bitmap="hourglass")
B3 = Button(root, relief=RAISED, bitmap="info")
B4 = Button(root, relief=RAISED, bitmap="question")
B5 = Button(root, relief=RAISED, bitmap="warning")
B6 = Button(root, relief=RAISED, bitmap="gray75")
B7 = Button(root, relief=RAISED, bitmap="questhead")
B1.pack()
B2.pack()
B3.pack()
B4.pack()
B5.pack()
B6.pack()
B7.pack()
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม

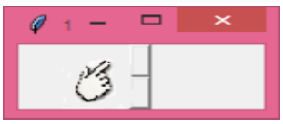


7. Cursors เคอร์เซอร์หรือตัวชี้เมาส์ คือ สัญลักษณ์แสดงตำแหน่งของเมาส์บนจอภาพ ไพธอนเตรียมสัญลักษณ์สำหรับใช้เป็นเคอร์เซอร์ให้เลือกใช้งานดังรูป

ตัวอย่างการใช้งานเคอร์เซอร์

```
from tkinter import *
root = Tk()
B1 = Button(root, relief=RAISED, cursor="hand1")
B2 = Button(root, relief=RAISED, cursor="heart")
B1.pack()
B2.pack()
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



5. การสร้างและใช้งาน Widgets พื้นฐาน

ในหัวข้อนี้จะกล่าวถึงวิธีการสร้าง Widgets พื้นฐานที่สำคัญๆ สำหรับใช้ในการสร้าง GUI เช่น frames, labels, buttons, checkbuttons, radiobuttons, entries และ comboboxes เป็นต้น

1. Frame เฟรมเป็น Widget ที่มีลักษณะเป็นรูปสี่เหลี่ยม โดยปกติเฟรมจะถูกใช้สำหรับจัดกลุ่มหรือบรรจุ Widgets อื่นๆ ที่เกี่ยวข้องหรือสัมพันธ์กันเข้าไว้ด้วยกัน

รูปแบบคำสั่งสำหรับการสร้างเฟรม คือ f = Frame(root, option=value, ...)

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของเฟรม แสดงในตารางด้านล่าง

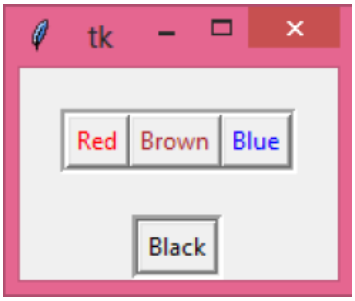
Option	คำอธิบาย
bd	กำหนดขนาดความกว้างของขอบเฟรม มีหน่วยเป็นพิกเซล ค่าเริ่มต้นเท่ากับ 2 พิกเซล เช่น Frame(root, bd = 5, height = 50, width = 100, relief = GROOVE)
bg	กำหนดสีพื้นด้านหลังของเฟรม เช่น Frame(root, bd = 5, height = 50, width = 100, relief = GROOVE, bg = "green")

cursor	กำหนดรูปแบบของเคอร์เซอร์ เคอร์เซอร์จะเปลี่ยนรูปเมื่อเคลื่อนเมาส์ทับบนเฟรม เช่น Frame(root, bd = 3, height = 50, width = 100, relief = GROOVE, cursor = "hand1")
height	กำหนดความสูงของเฟรม มีหน่วยเป็นพิกเซล เช่น Frame(root, bd = 3, height = 50, width = 100, relief = GROOVE)
width	กำหนดความกว้างของเฟรม ถ้าไม่กำหนดไพธอนจะกำหนดขนาดเท่ากับความกว้างของฟอนต์แทน เช่น Frame(root, bd = 3, height = 50, width = 100)
highlightbackground	กำหนดแถบสีพื้นหลังเมื่อเฟรมได้รับความสนใจ (Focus) เช่น Frame(root, bd = 3, height = 50, width = 100, highlightbackground = "green")
highlightcolor	กำหนดแถบสีเมื่อเฟรมได้รับความสนใจ เช่น Frame(root, bd = 3, height = 50, width = 100, highlightcolor = "green")
highlightthickness	กำหนดความกว้างจากขอบของเฟรม เช่น Frame(root, bd = 3, height = 50, width = 100, highlightthickness = 2)
relief	กำหนดลักษณะเฟรมในรูปแบบ 3 มิติ เช่น Frame(root, bd = 5, height = 50, width = 100, relief = GROOVE)

ตัวอย่างการใช้งานเฟรม (ตัวอย่างที่ 1)

```
from tkinter import *
root = Tk()

frame = Frame(root, bd=3, height=300, width=500, relief=GROOVE, cursor="hand1", highlightthickness=20)
frame.pack(expand=True)
redbutton = Button(frame, text="Red", fg="red")
redbutton.pack(side=LEFT)
greenbutton = Button(frame, text="Brown", fg="brown")
greenbutton.pack(side=LEFT)
bluebutton = Button(frame, text="Blue", fg="blue")
bluebutton.pack(side=LEFT)
bottomframe = Frame(root, bd=3, cursor="hand1", relief=SUNKEN)
bottomframe.pack(side = BOTTOM)
blackbutton = Button(bottomframe, text="Black", fg="black")
blackbutton.pack(side = BOTTOM)
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



จากตัวอย่างโปรแกรม แสดงการสร้างเฟรมเพื่อรอบรับการวาง Widget ชนิดปุ่มลงในเฟรมดังกล่าว บรรทัดที่ 4 เป็นการสร้างเฟรมชื่อ frame ให้มีขนาดสูงเท่ากับ 300 และกว้างเท่ากับ 500 โดยมีความหนาของขอบเฟรมเท่ากับ 3 (bd = 3) กรอบของเฟรมเป็นแบบร่องลึก (relief = GROOVE) มีขนาดความกว้างจากขอบเฟรมเท่ากับ 20 (highlightthickness = 20) เมื่อเลื่อนเมาส์เข้าใกล้เฟรมดังกล่าว เคอร์เซอร์จะเปลี่ยนเป็นรูปมือ (cursor = "hand1") และเฟรมที่สร้างขึ้นจะเขียนลงบนหน้าต่างหลักของโปรแกรม (root) บรรทัดที่ 5 เป็นการกำหนดให้เฟรมดังกล่าววางลงตรงกลางหน้าต่างหลักด้วยเมธอด frame.pack(expand=True) บรรทัดถัดไปโปรแกรมทำการสร้างปุ่ม มีข้อความสีแดง น้ำตาล และน้ำเงินลงบนเฟรมดังกล่าวตามลำดับ สังเกตว่าขนาดของเฟรมจะเท่ากับขนาดของปุ่ม เนื่องจากเมธอด pack() จะบีบอัดให้เฟรมมีขนาดเท่ากับผลรวมความกว้างและความยาวของ Widgets ที่อยู่ในเฟรมนั่นเอง (โปรแกรมจะไม่สนใจความสูงและความกว้างที่กำหนดในเฟรม)

บรรทัดที่ 12 โปรแกรมทำการสร้างเฟรมอีกครั้ง ชื่อ bottomframe โดยเฟรมดังกล่าวมีลักษณะของกรอบที่จมลึกลงไป (relief = SUNKEN) และเฟรมดังกล่าวถูกจัดวางให้อยู่ในตำแหน่งด้านล่างของหน้าต่างหลักด้วยเมธอด bottomframe.pack(side = BOTTOM) จากนั้นโปรแกรมสร้างปุ่มที่มีข้อความสีดำ ใส่ลงในเฟรม bottomframe โดยกำหนดให้วางในตำแหน่งด้านล่างของเฟรม (side = BOTTOM) ผลลัพธ์ที่ได้แสดงในรูปด้านบน ผู้เขียนโปรแกรมสามารถเลือกใช้เมธอด grid() แทนเมธอด pack() ในการจัดวาง Widgets ก็ได้ ดังตัวอย่างโปรแกรมด้านล่าง

ตัวอย่างการใช้งานเฟรม (ตัวอย่างที่ 2)

```
from tkinter import *
root = Tk()

frame = Frame(root, bd=3, height=300, width=500, relief=GROOVE, cursor="hand1", highlightthickness=20)
frame.pack()

Button(frame, text="Red", fg="red").grid(column=2, row=3)
Label(frame, text=" ", fg="red").grid(column=3, row=3)
Button(frame, text="Brown", fg="brown").grid(column=4, row=3)
Label(frame, text=" ", fg="red").grid(column=5, row=3)
Button(frame, text="Blue", fg="blue").grid(column=6, row=3)

root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม

2. Button (ปุ่ม) Button คือ Widget ที่มีลักษณะเป็นรูปสี่เหลี่ยมผืนผ้า มีหน้าที่ตอบสนองกับผู้ใช้งานโดยวิธีการกดลงและปล่อย การแสดงผลบนปุ่มจะเป็นได้ทั้งข้อความหรือรูปภาพก็ได้ เมื่อกดปุ่มและปล่อย (เกิด event) จะส่งผลให้เกิดการเรียกใช้งานฟังก์ชันหรือเมธอดที่ฝังอยู่กับปุ่มได้ เพื่อทำหน้าที่อย่างใดอย่างหนึ่ง เช่น กดปุ่ม Cancel จะทำให้โปรแกรมยกเลิกคำสั่งที่เพิ่งกระทำเสร็จ เป็นต้น

รูปแบบคำสั่งสำหรับการสร้างปุ่ม คือ b = Button(root, option=value, ...)

พารามิเตอร์ คือ

- **root** คือ วินโดวส์หลัก (root window)

- **option** คือ คุณสมบัติต่างๆ ของปุ่ม แสดงในตารางด้านล่าง

Option	คำอธิบาย
activebackground	กำหนดสีบนพื้นหลังของปุ่มเมื่อผู้ใช้คลิกบนปุ่ม เช่น Button(root, text = "Hello", activebackground = "yellow")
activeforeground	กำหนดสีข้อความบนปุ่มเมื่อผู้ใช้คลิกบนปุ่ม เช่น Button(root, text = "Hello", activebackground = "yellow", activeforeground = "red")
bd	กำหนดขนาดความกว้างขอบปุ่มมีหน่วยเป็นพิกเซล ค่าเริ่มต้นเท่ากับ 2 พิกเซล เช่น Button(root, text = "Hello", bd = 5)
bg	กำหนดสีพื้นหลังของปุ่ม เช่น Button(root, text = "Hello", bg = "red")
command	ฟังชันหรือเมธอดที่จะถูกเรียกใช้เมื่อปุ่มถูกกดหรือคลิก เช่น Button(root, text = "Hello", bg = "red", command = myFunc)
fg	กำหนดสีของฟอนต์ เช่น Button(root, text = "Hello", bg = "red", fg = "blue")
font	กำหนดรูปแบบฟอนต์ของปุ่ม เช่น Button(root, text = "Hello", bg = "red", font = "Times 10 bold") หรือ font = ("Helvetica", 16)
height	กำหนดความสูงของปุ่ม (กรณีปุ่มที่ถูกสร้างด้วยรูปภาพ จะมีหน่วยเป็นพิกเซล) เช่น Button(root, text = "Hello", height = 5)
highlightcolor	กำหนดแถบสีเมื่อเฟรมได้รับความสนใจ เช่น Button(root, text = "Hello", highlightcolor = "green")
image	กำหนดรูปภาพให้กับปุ่มแทนการใช้ข้อความ เช่น image = PhotoImage(file = 'printer.png') B = Button(root, text ="Hello", image=image)
justify	กำหนดตำแหน่งการแสดงผลข้อความบนปุ่ม โดย LEFT = วางข้อความชิดด้านซ้ายของปุ่ม, RIGHT = ชิดด้านขวา, CENTER = วางตรงกลางปุ่ม เช่น Button(root, text = "Hello\nPython\nLanguage", justify=LEFT)
padx	เติมข้อความว่าง (Padding) ด้านซ้ายและขวาของข้อความในปุ่ม เช่น Button(root, text = "Hello", padx = 5)
pady	เติมข้อความว่าง (Padding) ด้านบนและล่างของข้อความในปุ่ม เช่น Button(root, text = "Hello", pady = 5)
relief	กำหนดลักษณะขอบของปุ่มในแบบ 3D เช่น Button(root, text = "Hello", relief = GROOVE)

state	กำหนดให้ปุ่มทำงานหรือไม่ทำงาน ถ้ากำหนดเป็น DISABLED ปุ่มจะไม่สามารถกดได้ แต่ถ้ากำหนดเป็น ACTIVE จะสามารถกดปุ่มได้ เช่น Button(root, text = "Hello", state = DISABLED)
underline	ตัวอักษรของปุ่มจะถูกขีดเส้นใต้ โดยค่า -1 คือปุ่มจะไม่ถูกขีดเส้น แต่ตัวเลขอื่นๆ ที่เป็นค่าบวกจะทำให้ตัวอักษรบนปุ่มถูกขีดเส้น (0 คืออักษรตัวแรก) เช่น Button(root, text = "Hello", underline = 1)
width	กำหนดความกว้างของปุ่ม (กรณีปุ่มที่ถูกสร้างด้วยรูปภาพ จะมีหน่วยเป็นพิกเซล) เช่น Button(root, text = "Hello", width = 5)
wraplength	เมื่อค่าดังกล่าวถูกกำหนดเป็นจำนวนเต็มบวก ข้อความบนปุ่มจะถูกจำกัดพื้นที่โดยจะแสดงผลอยู่ในขอบเขตที่กำหนดใน wraplength เท่านั้น เช่น ข้อความ "Hello" จะใช้พื้นที่ในการแสดงผลเท่ากับ 28 พิกเซล เมื่อกำหนด wraplength = 10 จะทำให้ข้อความแสดงในแนวตั้ง ตัวอย่างเช่น Button(root, text = "Hello", wraplength = 10)

Widget ชนิดปุ่มมีเมธอดที่ช่วยเสริมในการทำงานของปุ่มคือ

- เมธอด flash() ทำหน้าที่วาดปุ่มใหม่ เช่น

```
B = Button(root, text = "Hello")
B.flash()
```

- เมธอด invoke() บังคับให้คำสั่งออฟชัน command ที่กำหนดไว้ในปุ่มทำงานทันที เช่น

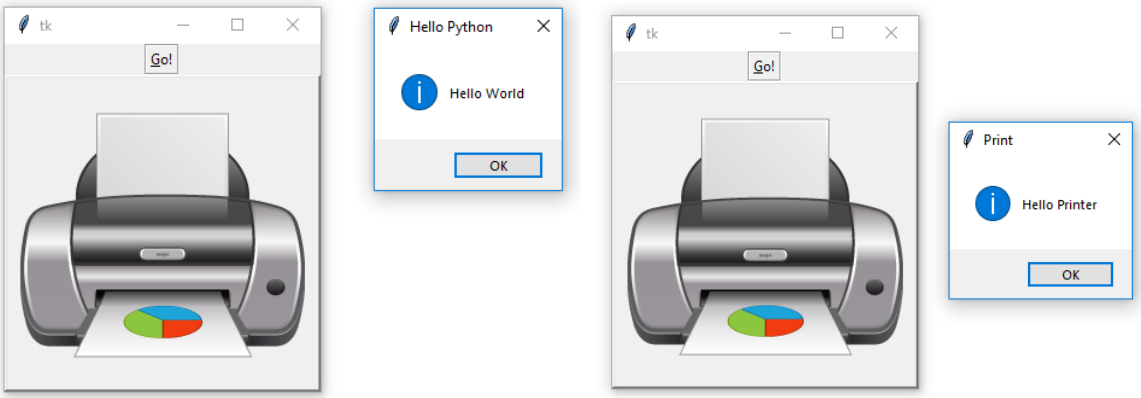
```
B = Button(root, text = "Hello", command = callBack)
B.invoke()
```

เมื่อสั่งรันโปรแกรมจะส่งผลให้ไพธอนเรียกเมธอด callback() มาทำงานทันทีโดยไม่ต้องคลิกที่ปุ่ม

ตัวอย่างการสร้างและใช้งานปุ่ม

```
from tkinter import *
from PIL import ImageTk, Image   #เรียกใช้ไลบรารี่ PIL
from tkinter import messagebox
root = Tk()

def helloCallBack():
   messagebox.showinfo( "Hello Python", "Hello World")

def printCallBack():
   messagebox.showinfo( "Print", "Hello Printer")

image = ImageTk.PhotoImage(file = 'printer.png')
B1 = Button(root, text ="Go!", relief = GROOVE, underline = 0, activebackground = "yellow",
        activeforeground = "red", command = helloCallBack)
B2 = Button(root, image = image, padx = 30, pady = 20, command = printCallBack)
B1.pack(expand=True)
B2.pack()
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม ให้ทดลองคลิกที่ปุ่ม Go! และปุ่มเครื่องพิมพ์



จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Button (ปุ่ม) โดยบรรทัดที่ 7 โปรแกรมสร้างเมธอดชื่อ helloCallBack() มีหน้าที่สร้างกล่องข้อความโดยพิมพ์ข้อความว่า "Hello World" ออกจอภาพ บรรทัดที่ 10 โปรแกรมสร้างเมธอดชื่อ printCallBack() มีหน้าที่สร้างกล่องข้อความ โดยพิมพ์ข้อความว่า "Hello Printer" ออกจอภาพ บรรทัดที่ 12 สร้างอ๊อปเจ็กต์ image ที่เชื่อมโยงไปยังรูปภาพชื่อว่า "printer.png" เพื่อใช้สำหรับแสดงบนปุ่ม บรรทัดที่ 13 โปรแกรมสร้างปุ่มโดยมีข้อความบนปุ่มคือ "Go!" (text = "Go!"), ขีดเส้นใต้ที่ตัวอักษร "G" (underline=0), ขอบของปุ่มเป็นแบบร่องลึก (relief=GROOVE), เมื่อปุ่มถูกกด ปุ่มจะเป็นสีเหลือง (activebackground = "yellow"), เมื่อปุ่มถูกกดข้อความจะเป็นสีแดง (activeforeground = "red") และเมื่อปุ่มดังกล่าวถูกคลิก โปรแกรมจะเรียกใช้เมธอด helloCallBack()

บรรทัดที่ 15 โปรแกรมสร้างปุ่มโดยมีภาพบนปุ่มคือ "printer.png" (image = image) เมื่อปุ่มดังกล่าวถูกคลิก โปรแกรมจะเรียกใช้เมธอด printCallBack() ผลลัพธ์ที่ได้แสดงดังรูปด้านบน

3. Canvas คือ Widget ที่มีลักษณะเป็นรูปสี่เหลี่ยม มีเป้าหมายเพื่อใช้สำหรับจัดวางรูปภาพ เฟรม ข้อความ หรือวาดรูปภาพที่มีความซับซ้อนได้ รูปร่างของ Canvas แสดงดังรูป

รูปแบบคำสั่งสำหรับการสร้าง Canvas คือ

```
c = Canvas( root, option=value, ... )
```

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Canvas แสดงในตารางด้านล่าง

Option	คำอธิบาย
confine	กำหนดให้ Canvas สามารถเลื่อน Scroll ได้ (ดีฟอลต์เท่ากับ True) เช่น Canvas(root, bd = 5, relief = GROOVE, height = 250, width = 300, confine = False)
cursor	กำหนดรูปแบบของเคอร์เซอร์ เคอร์เซอร์จะเปลี่ยนรูปเมื่อเคลื่อนเมาส์ทับบน Canvas เช่น Canvas(root, bd = 5, relief = GROOVE, height = 250, width = 300, cursor = "hand1")
bd	กำหนดขนาดความกว้างขอบ Canvas มีหน่วยเป็นพิกเซล ค่าเริ่มต้นเท่ากับ 2 พิกเซล เช่น Canvas(root, bd = 3, relief = GROOVE, height = 250, width = 300)
bg	กำหนดสีพื้นหลังของ Canvas เช่น Canvas(root, bd = 3, relief = GROOVE, bg = "blue", height = 250, width = 300)
scrollregion	กำหนดพื้นที่ที่ Canvas สามารถขยายได้สูงสุดเท่าใด โดยขอบเขตพื้นที่กำหนดในตัวแปรชนิด Tuple ซึ่งมีรูปแบบ tuple(w, e, n, s) โดย w คือขอบด้านซ้าย, e คือขอบด้านขวา, n คือด้านบน และ s คือด้านล่าง เช่น Canvas(root, bd = 5, relief = GROOVE, scrollregion = (0, 0, 500, 500))
xscrollincrement	กำหนดขนาดการเพิ่มขึ้นของจำนวนคอลัมน์ เมื่อ Canvas ใช้ Scrollbar ในแนวนอน ใช้ในกรณีที่ Canvas ต้องการแสดงผลมากกว่าขอบเขตที่ Canvas กำหนดไว้เช่น Canvas(frame, xscrollcommand = xscrollbar.set, yscrollcommand = yscrollbar.set, xscrollincrement = 10, yscrollincrement=10)
height	กำหนดความสูงของ Canvas เช่น Canvas(root, bd = 3, relief = GROOVE, height = 250, width = 300)
highlightcolor	กำหนดแถบสีเมื่อ Canvas ได้รับความสนใจ (Focus) เช่น Canvas(root, bd = 5, relief = GROOVE, height = 250, width = 300, highlightcolor = "green")
xscrollcommand	กำหนดให้ Canvas สามารถใช้งาน Scrollbar ในแนวนอนได้ เช่น Canvas(frame, xscrollcommand = xscrollbar.set, yscrollcommand = yscrollbar.set)
yscrollincrement	เหมือนกับ xscrollincrement แต่เปลี่ยนเป็นแนวตั้งแทน เช่น Canvas(frame, xscrollcommand = xscrollbar.set, yscrollcommand = yscrollbar.set, xscrollincrement = 10, yscrollincrement = 10)
yscrollcommand	กำหนดให้ Canvas สามารถใช้งาน Scrollbar ในแนวตั้งได้ เช่น Canvas(frame, xscrollcommand = xscrollbar.set, yscrollcommand = yscrollbar.set)

relief	กำหนดลักษณะขอบของ Canvas ในแบบ 3D เช่น Canvas(root, bd = 5, relief = GROOVE, height = 250, width = 300)
width	กำหนดความกว้างของ Canvas เช่น Canvas(root, bd = 3, relief = GROOVE, height = 250, width = 300)

Canvas สามารถวาดรูปต่างๆ ลงบน Canvas ได้ ดังนั้นเมธอดต่อไปนี้ จึงใช้งานร่วมกับ Canvas ได้เป็นอย่างดี

- การวาดเส้นโค้งบน Canvas ด้วยเมธอด create_arc() ตัวอย่างเช่น

```
coord = 10, 50, 240, 210
arc = canvas.create_arc(coord, start=0, extent=150, fill="blue")
```

- การสร้างรูปภาพบน Canvas ด้วยเมธอด create_image() ตัวอย่างเช่น

```
filename = PhotoImage(file = "sunshine.gif")
image = canvas.create_image(50, 50, anchor = NE, image = filename)
```

- การวาดเส้นบน Canvas ด้วยเมธอด create_line() ตัวอย่างเช่น

```
line = canvas.create_line(x0, y0, x1, y1,..., xn, yn, options)
```

- การวาดรูปวงรีบน Canvas ด้วยเมธอด create_oval() ตัวอย่างเช่น

```
oval = canvas.create_oval(x0, y0, x1, y1, options)
```

- การวาดรูปหลายเหลี่ยมบน Canvas ด้วยเมธอด create_ polygon() ตัวอย่างเช่น

```
polygon = canvas.create_polygon(x0, y0, x1, y1,...xn, yn, options)
```

- การวาดรูปสี่เหลี่ยมบน Canvas ด้วยเมธอด create_ rectangle() ตัวอย่างเช่น

```
rect = canvas.create_rectangle(50, 25, 150, 75,fill = "blue")
```

- การลบภาพวาดทั้งหมดออกจาก Canvas ด้วยเมธอด create_ delete() ตัวอย่างเช่น

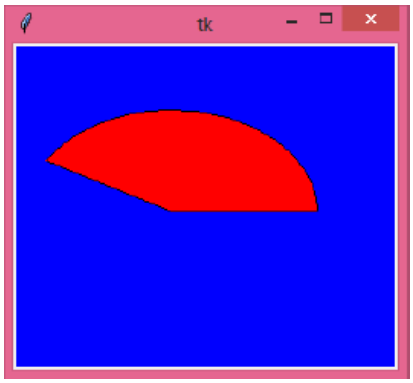
```
rect = canvas.delect(oval) #remove oval from Canvas
```

rect = canvas.**delect**(ALL) #remove ALL from Canvas

ตัวอย่างการสร้างและใช้งาน Canvas

```
from tkinter import *
root = Tk()

C = Canvas(root, bg="blue", height=250, width=300)
coord = 10, 50, 240, 210
arc = C.create_arc(coord, start=0, extent=150, fill="red")
C.pack()
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Canvas โดยบรรทัดที่ 4 โปรแกรมสร้าง Canvas ที่มีสีพื้นหลังเป็นสีน้ำเงิน (bg = "blue") มีความกว้างเท่ากับ 300 และสูงเท่ากับ 250 บรรทัดที่ 5 กำหนดตำแหน่งของ coord ซึ่งเป็นตัวแปรชนิด Tuple เท่ากับ 10, 50, 240, 210 ตามลำดับ บรรทัดที่ 6 โปรแกรมทำการวาดวงกลมเสี้ยวสีแดงลงบน Canvas ดังรูปด้านบน

4. Checkbutton คือ Widget ที่มีลักษณะเป็นรูปสี่เหลี่ยมเล็กๆ เพื่อให้ผู้ใช้สามารถเลือกได้โดยการคลิกบน Checkbutton ดังกล่าว ผู้ใช้สามารถเลือกได้มากกว่า 1 ตัวเลือก

รูปแบบคำสั่งสำหรับการสร้าง Checkbutton คือ

cb = Checkbutton(root, option=value, ...)

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Checkbutton แสดงในตารางด้านล่าง

Option	คำอธิบาย
activebackground	กำหนดสีบนพื้นหลังของ Checkbutton เมื่อผู้ใช้คลิกบนปุ่ม เช่น Checkbutton(root, text = "Music", activebackground = "gray")
activeforeground	กำหนดสีข้อความบน Checkbutton เมื่อผู้ใช้คลิกบนปุ่ม เช่น Checkbutton(root, text = "Music", activebackground = "gray", activeforeground = "white")
bd	กำหนดขนาดความกว้างขอบ Checkbutton มีหน่วยเป็นพิกเซล ค่าเริ่มต้นเท่ากับ 2 พิกเซล เช่น Checkbutton(root, text = "Music", bg = "yellow", bd = 5, relief = GROOVE)
bg	กำหนดสีพื้นหลังของ Checkbutton เช่น Checkbutton(root, text = "Music", bg = "yellow")
bitmap	แสดงภาพแบบ monochrome บน Checkbutton
command	ฟังชันหรือเมธอดที่จะถูกเรียกใช้เมื่อ Checkbutton ถูกกดหรือคลิก เช่น Checkbutton(root, text = "Music", command = myFunc)
fg	กำหนดสีของฟอนต์ เช่น Checkbutton(root, text = "Music", fg = "red")
cursor	กำหนดรูปแบบของเคอร์เซอร์ เคอร์เซอร์จะเปลี่ยนรูปเมื่อเคลื่อนเมาส์ทับบน Checkbutton เช่น Checkbutton(root, text = "Music", fg = "red", cursor = "hand1")
font	กำหนดรูปแบบของฟอนต์ของ Checkbutton เช่น Checkbutton(root, text = "Music", fg = "red", font = "Times 10 bold")
height	กำหนดความสูงของ Checkbutton เช่น Checkbutton(root, text = "Music", height = 5, width = 10, bg = "pink")
width	กำหนดความกว้างของ Checkbutton เช่น Checkbutton(root, text = "Music", height = 5, width = 10, bg = "pink")
highlightcolor	กำหนดแถบสีเมื่อ Checkbutton ได้รับความสนใจ เช่น Checkbutton(root, text = "Music", height = 5, width = 10, highlightcolor = "green")
image	กำหนดรูปภาพให้กับปุ่มแทนการใช้ข้อความ เช่น image = PhotoImage(file = 'printer.png') Checkbutton(root, text = "Music", height = 5, width = 10, image = image)

justify	กำหนดตำแหน่งการแสดงผลข้อความบนปุ่ม โดย LEFT = วางข้อความชิดด้านซ้ายของปุ่ม, RIGHT = ชิดด้านขวา, CENTER = วางตรงกลางปุ่ม เช่น Checkbutton(root, text = "Music\nVideo", justify = LEFT)
padx	เติมข้อความว่าง (Padding) ด้านซ้ายและขวาของข้อความใน Checkbutton เช่น Checkbutton(root, text = "Music", padx = 5)
pady	เติมข้อความว่าง (Padding) ด้านบนและล่างของข้อความใน Checkbutton เช่น Checkbutton(root, text = "Music", pady = 5)
relief	กำหนดลักษณะขอบของ Checkbutton ในแบบ 3D เช่น Checkbutton(root, text = "Music", relief = GROOVE)
state	กำหนดให้ Checkbutton ทำงานหรือไม่ทำงาน ถ้ากำหนดเป็น DISABLED Checkbutton จะไม่ทำงาน เช่น Checkbutton(root, text = "Music", state = DISABLED)
underline	ตัวอักษรของ Checkbutton จะถูกขีดเส้นใต้ (0 คืออักษรตัวแรก, -1 = ไม่ขีดเส้นใต้) เช่น Checkbutton(root, text = "Music", underline = 1)
wraplength	ข้อความบนปุ่มจะถูกจำกัดพื้นที่ โดยจะแสดงผลอยู่ในขอบเขตที่กำหนดใน wraplength เท่านั้น เช่น Checkbutton(root, text = "Music", wraplength = 20)
onvalue	กำหนดค่าเริ่มต้นให้กับ Checkbutton เมื่อ Checkbutton ถูกคลิกเลือกโปรแกรมจะดึงค่าใน onvalue ไปใช้งาน เช่น Checkbutton(root, text = "Music", onvalue = 1), หน้าที่ของ onvalue คือ จัดเตรียมค่าข้อมูลเพื่อให้เมธอดอื่นๆ นำไปใช้งานนั่นเอง (onvalue ในรูปแบบสตริงคือ "on")
offvalue	กำหนดค่าเริ่มต้นให้กับ Checkbutton เมื่อ Checkbutton ถูกคลิกยกเลิกโปรแกรมจะดึงค่าใน offvalue ไปใช้งาน เช่น Checkbutton(root, text = "Music", offvalue = 0), หน้าที่ของ offvalue คือ จัดเตรียมค่าข้อมูลเพื่อให้เมธอดอื่นๆ นำไปใช้งานเช่นเดียวกับ onvalue (offvalue ในรูปแบบสตริงคือ "off")
selectcolor	กำหนดสีของช่องว่างใน Checkbutton เช่น Checkbutton(root, text = "Music", selectcolor = "red")
selectimage	กำหนดรูปภาพของช่องว่างใน checkbutton เช่น image = PhotoImage(file = 'printer.png') Checkbutton(root, text = "Music", selectimage = image)
text	กำหนดข้อความให้กับ Checkbutton ถ้าต้องการกำหนดข้อความมากกว่า 1 บรรทัดให้ใช้ \n เช่น "Music \n Audio \n Guitar"

variable	ใช้สำหรับดึงข้อมูลจาก Widgets หรือดึงข้อมูลจาก onvalue และ offvalue นั่นเอง โดยจะทำงานร่วมกับเมธอด IntVar() เมื่อข้อมูลใน onvalue/offvalue เป็นตัวเลข และทำงานร่วมกับเมธอด StringVar() เมื่อข้อมูลใน onvalue/offvalue เป็นสตริง เช่น Checkbutton(root, text = "Music", variable = IntVar(), onvalue = 1, offvalue = 0)

Widget ชนิด Checkbutton มีเมธอดที่ช่วยสนับสนุนการทำงานคือ

- เมธอด deselect() ทำหน้าที่เคลียร์ค่า Checkbutton ที่เลือกไว้ (turn-off) เช่น

```
C = Checkbutton(root, text = "Music")
C.deselect()
```

- เมธอด flash() ทำหน้าที่วาด Checkbutton ใหม่ เช่น

```
C = Checkbutton(root, text = "Music")
C.flash()
```

- เมธอด invoke() จะบังคับให้ทำคำสั่งหลัง command ทันที เช่น

```
C = Checkbutton(root, text = "Music", command=callBack)
C.invoke()
```

- เมธอด select() เซ็ต (turn-on) ค่าให้กับ Checkbutton เมื่อสถานะเดิมของ Checkbutton ไม่ถูกเลือก จะทำให้สถานะกลายเป็นถูกเลือกแทน เช่น

```
C = Checkbutton(root, text = "Music")
C.select()
```

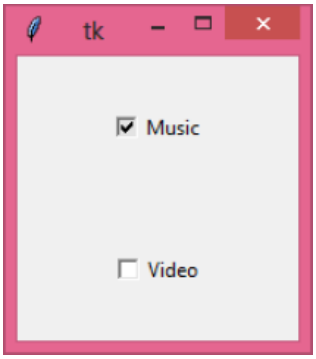
- เมธอด toggle() สลับระหว่าง turn-on และ turn-off เมื่อสถานะเดิมของ Checkbutton ถูกกำหนดเป็น on เมื่อเรียกเมธอด toggle() จะทำให้ Checkbutton กลายเป็น off เช่น

```
C = Checkbutton(root, text = "Music")
C.toggle()
```

ตัวอย่างการสร้างและใช้งาน Checkbutton

```
from tkinter import *
root = Tk()

CheckVar1 = StringVar()
CheckVar2 = IntVar()

def checkCallBack():
   C1.select()
   C2.toggle()
   print(CheckVar1.get())
   print(CheckVar2.get())

C1 = Checkbutton(root, text = "Music", variable = CheckVar1, onvalue = "on", offvalue = "off",
            height = 5, width = 20, command = checkCallBack)
C2 = Checkbutton(root, text = "Video", variable = CheckVar2, onvalue = 1, offvalue = 0,
            height = 5, width = 20, command = checkCallBack)
C1.pack()
C2.pack()
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูป ให้ทดลองคลิกที่ Checkbutton และสังเกตการเปลี่ยนแปลง



จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Checkbutton บรรทัดที่ 4 เป็นการสร้างตัวแปรชื่อ CheckVar1 เพื่อใช้สำหรับเก็บค่าข้อมูลชนิดสตริง ในที่นี้คือ 'on' หรือ 'off' ที่เกิดขึ้นจากการคลิกที่ Checkbutton (C1) โดยตัวแปรดังกล่าวสร้างขึ้นจากคลาสชื่อ StringVar() บรรทัดที่ 5 เป็นการสร้างตัวแปรชื่อ CheckVar2 ซึ่งเป็นชนิดจำนวนเต็ม ในที่นี้คือ 0 หรือ 1 ที่เกิดจากการคลิก Checkbutton (C2) โดยสร้างมาจากคลาสชื่อ IntVar() บรรทัดที่ 7 โปรแกรมสร้างฟังชันชื่อ checkCallBack() เพื่อทดสอบการทำงานของ Checkbutton C1 และ C2 โดยฟังชันดังกล่าวเรียกใช้งานเมธอด select() และ toggle() พร้อมกับพิมพ์ค่า CheckVar1 และ CheckVar2 ออกทางจอภาพเมื่อผู้ใช้มีการคลิกที่ Checkbutton

บรรทัดที่ 13 สร้าง Widget Checkbutton ชื่อ C1 ที่มีข้อความว่า 'Music' มีความสูงเท่ากับ 5 ความกว้างเท่ากับ 20 เมื่อ Checkbutton C1 ถูกคลิกเลือกจะนาค่าใน onvalue ('on') เก็บไว้ในตัวแปร CheckVar1 ทันที แต่ถ้า C1 ไม่ถูกเลือก โปรแกรมจะนาค่า offvalue ('off') เก็บไว้ในตัวแปร CheckVar1 แทน ใน Checkbutton C1 โปรแกรมฝังคำสั่งเอาไว้ ถ้ามีการคลิกที่ Checkbutton C1 โปรแกรมจะเรียกใช้เมธอด checkCallBack() ทันที (command = checkCallBack)

บรรทัดที่ 15 สร้าง Widget Checkbutton ชื่อ C2 ที่มีข้อความว่า 'Video' มีความสูงเท่ากับ 5 ความกว้างเท่ากับ 20 เมื่อ Checkbutton C2 ถูกคลิกเลือกจะนาค่าใน onvalue (1) เก็บไว้ในตัวแปร CheckVar2 ทันที แต่ถ้า C2 ไม่ถูกเลือก โปรแกรมจะนาค่า offvalue (0) เก็บไว้ในตัวแปร CheckVar2 แทน ใน Checkbutton C2 โปรแกรมฝังคำสั่งเอาไว้ ถ้ามีการคลิกที่ Checkbutton C2 โปรแกรมจะเรียกใช้เมธอด checkCallBack() ผลลัพธ์ที่ได้แสดงดังในรูปด้านบน

5. Entry (นำเข้าข้อมูล) คือ Widget ที่มีลักษณะเป็นกล่องข้อความ เพื่อใช้รับข้อมูลจากผู้ใช้เข้ามาประมวลผลในโปรแกรม เช่น การป้อนชื่อ-สกุล รหัสผ่าน เป็นต้น

รูปแบบคำสั่งสำหรับการสร้าง Entry คือ

```
e = Entry( root, option=value, ... )
```

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Entry

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ bg, bd, cursor, font, fg, highlightcolor, justify, relief และ state สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
exportselection	โดยปกติเมื่อผู้ใช้เลือกข้อความภายใน Entry ข้อมูลที่ถูกเลือกจะถูกส่งไปเก็บไว้ในคลิปบอร์ด (Clipboard) โดยอัตโนมัติ เมื่อไม่ต้องการให้ข้อมูลดังกล่าวถูกส่งไปยังคลิปบอร์ด ให้กำหนด exportselection = 0 เช่น Entry(root, bd = 5, exportselection = 0)
selectbackground	กำหนดสีพื้นหลังของข้อความเมื่อข้อความดังกล่าวถูกเลือก (highlight) เช่น Entry(root, bd = 5, selectbackground = "red")
selectborderwidth	กำหนดขนาดความกว้างของขอบรอบๆ ข้อความที่ถูกเลือก (ค่าดีฟอลต์คือ 1 พิกเซล) เช่น Entry(root, bd = 3, width = 10, selectborderwidth = 10)
selectforeground	กำหนดสีของข้อความใน Entry เมื่อข้อความดังกล่าวถูกเลือก เช่น Entry(root, bd = 3, width = 10, selectforeground = "red")
show	ข้อความที่ป้อนเข้าไปใน Entry จะมีลักษณะเป็น Clear text (ไม่มีการเข้ารหัส) แต่บางครั้ง ผู้ใช้งานจำเป็นต้องซ่อนข้อความดังกล่าว เช่น รหัสผ่าน เป็นต้น สามารถกำหนดโดย show="*" เช่น Entry(root, bd = 3,width = 10, show = "*")
textvariable	กำหนดตัวแปรสำหรับใช้เก็บค่าที่เกิดขึ้นจากการป้อนข้อมูลลงใน Entry โดยทำงานร่วมกับคลาส StringVar() เช่น entryVar = StringVar() Entry(root, bd = 3, width = 10, show = "*", textvariable = entryVar)
width	กำหนดขนาดความกว้างของ Entry ที่แสดงผล เช่น Entry(root, bd = 3, width = 5)
xscrollcommand	เมื่อคาดว่าผู้ใช้งานจะป้อนข้อมูลเกินความกว้าง (Width) ของ Entry ที่กำหนดไว้ สามารถใช้ xscrollcommand (แท็บสไลด์ในแนวนอน) ทำงานร่วมกับ Entry ได้ เช่น scrollbar = Scrollbar(root, orient = HORIZONTAL) Entry(root, bd = 3, width = 5, show = "*", xscrollcommand = scrollbar.set)

Widget ชนิด Entry มีเมธอดที่ช่วยสนับสนุนการทางาน คือ

- เมธอด delete(first, last = None) ทำหน้าที่ลบตัวอักษรออกจาก Entry โดยกำหนดตำแหน่งตัวอักษรเริ่มต้นที่พารามิเตอร์ first และตำแหน่งตัวอักษรตัวสุดท้ายในพารามิเตอร์ last ถ้าไม่ได้กำหนดค่าให้กับพารามิเตอร์ last ตัวอักษรตัวแรกจะถูกลบเพียงตัวเดียวเท่านั้น เช่น

```
E = Entry(root, bd = 3, width = 5, show = "*")
E.delete(0, END)
```

- เมธอด get() ทำหน้าดึงข้อมูลจาก Entry เป็นสตริง เช่น

```
E = Entry(root, bd = 3, width = 5, show = "*")
E.get()
```

- เมธอด icursor(index) กำหนดตำแหน่งของเคอร์เซอร์ในข้อความของ Entry ผ่านพารามิเตอร์ index เช่น

```
E = Entry(root, bd = 3, width = 5)
E.icursor(3)
```

- เมธอด index(index) เลื่อนตัวชี้ไปยังตำแหน่งที่ต้องการของข้อความใน Entry เช่น

```
E = Entry(root, bd = 3, width = 5)
E.index(3)
```

- เมธอด insert(index, s) เพิ่มสตริงในตำแหน่ง (index) ที่กำหนด เช่น

```
E = Entry(root, bd = 3, width = 5)
E.insert(3, "Hello")
```

- เมธอด select_adjust(index) เมธอดนี้ถูกใช้เพื่อยืนยันว่าตัวอักษรที่เลือกตรงตามที่ระบุไว้จริง เช่น

```
E = Entry(root, bd = 3, width = 5)
E.select_adjust(3)
```

- เมธอด select_clear() เคลียร์แทบ highlight บนข้อความที่ถูกเลือกใน Entry เช่น

E = Entry(root, bd=3, width=5)

E**.select_clear**()

- เมธอด select_from(index) กำหนดตำแหน่ง ANCHOR ด้วย index เช่น

E = Entry(root, bd = 3, width = 5)

E**.select_from**(3)

- เมธอด select_present() เมื่อข้อความถูกเลือก (highlight) จะคืนค่าเป็น True แต่ไม่ถูกเลือกจะคืนค่าเป็น False เช่น

E = Entry(root, bd = 3, width = 5)

E**.select_present**()

- เมธอด select_range(start, end) เลือกช่วงของข้อความที่ต้องการ โดยกำหนดตำแหน่งเริ่มต้น (start) และตำแหน่งสิ้นสุด (end) เช่น

E = Entry(root, bd = 3, width = 5)

E**.select_range**(2, 4)

- เมธอด select_to(index) เลือกช่วงของข้อความตั้งแต่ ANCHOR ไปถึงตัวอักษรตัวสุดท้าย เช่น

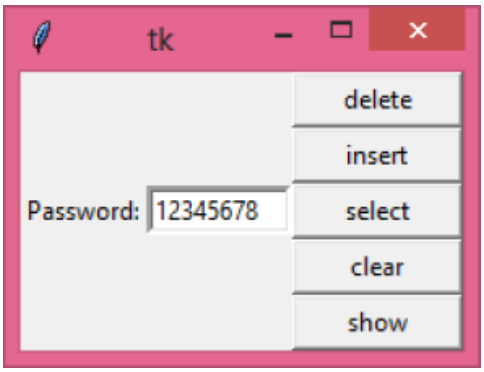
E = Entry(root, bd = 3, width = 5)

E**.select_to**(5)

ตัวอย่างการสร้างและใช้งาน Entry

```
from tkinter import *
root = Tk()
entryVar = StringVar()
def deleteCallBack():
   E1.delete(2, 4)
def insertCallBack():
   E1.insert(3, "Hello")
def selectCallBack():
   E1.select_range(2, 4)
def clearCallBack():
   E1.select_clear()
def showCallBack():
   print(entryVar.get())
L1 = Label(root, text="Password:")
L1.pack(side = LEFT)
E1 = Entry(root, bd=3, width=10, textvariable=entryVar)
E1.pack(side = LEFT)
B1 = Button(root, text="delete", width=10, command=deleteCallBack)
B1.pack()
B2 = Button(root, text="insert", width=10, command=insertCallBack)
B2.pack()
B3 = Button(root, text="select", width=10, command=selectCallBack)
B3.pack()
B4 = Button(root, text="clear", width=10, command=clearCallBack)
B4.pack()
B5 = Button(root, text="show", width=10, command=showCallBack)
B5.pack()

root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูป ให้ลองป้อนข้อมูลและคลิกเลือกปุ่มต่างๆ พร้อมกับสังเกตการเปลี่ยนแปลง



จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Entry บรรทัดที่ 3 สร้างตัวแปรชื่อ entryVar เพื่อใช้สำหรับเก็บค่าข้อมูลชนิดสตริงที่ป้อนลงใน Entry โดยตัวแปรดังกล่าวสร้างขึ้นจากคลาสชื่อ StringVar() บรรทัดที่ 4 สร้างฟังชันชื่อ deleteCallBack() ทำหน้าที่ลบข้อความใน Entry โดยการระบุตำแหน่งเริ่มต้น (first) และสิ้นสุด (last) ของข้อความด้วยเมธอด delete(first, last) บรรทัดที่ 6 สร้างฟังชันชื่อ insertCallBack() ทำหน้าที่แทรกหรือเพิ่มข้อความลงใน Entry ด้วยเมธอด insert(index, s) โดย index คือตำแหน่งที่ต้องการเพิ่มข้อความลงใน Entry และ s คือข้อความ บรรทัดที่ 8 สร้างฟังชัน selectCallBack() ทำหน้าที่เลือกช่วงของข้อความ (highlight) ใน Entry โดยใช้เมธอด selectRange(start, end) บรรทัดที่ 12 สร้างฟังชันชื่อ showCallBack() ทำหน้าที่แสดงผลข้อมูลที่อยู่ใน Entry ผ่านตัวแปรชนิดสตริงชื่อ entryVar โดยใช้เมธอด get()

บรรทัดที่ 14 โปรแกรมสร้าง Label เพื่อพิมพ์คำว่า 'Password' ออกจอภาพ บรรทัดที่ 16 สร้าง Entry ที่มีความกว้างของเท่ากับ 10 เมื่อผู้ใช้ป้อนข้อมูลใดๆ ลงใน Entry ข้อความเหล่าจะถูกเก็บไว้ในตัวแปร entryVar (textvariable=entryVar) บรรทัดที่ 18 สร้างปุ่มมีข้อความบนปุ่มคือ 'delete' เมื่อปุ่มดังกล่าวถูกกด โปรแกรมจะเรียกฟังชัน deleteCallBack() เข้ามาทำงานทันที บรรทัดที่ 20 สร้างปุ่มมีข้อความคือ 'insert' เมื่อปุ่มดังกล่าวถูกกด โปรแกรมจะเรียกฟังก์ชัน insertCallBack() บรรทัดที่ 22 สร้างปุ่มมีข้อความคือ 'select' เมื่อปุ่มดังกล่าวถูกกด โปรแกรมจะเรียกฟังก์ชัน selectCallBack() บรรทัดที่ 24 สร้างปุ่มมีข้อความคือ 'clear' เมื่อปุ่มดังกล่าวถูกกด โปรแกรมจะเรียกฟังก์ชัน clearCallBack() บรรทัดที่ 26 สร้างปุ่มมีข้อความคือ 'show' เมื่อปุ่มดังกล่าวถูกกด โปรแกรมจะเรียกฟังก์ชัน showCallBack() ผลลัพธ์แสดงดังรูปด้านบน

6. Label (เลเบลหรือป้ายชื่อ) คือ Widget ที่มีลักษณะเป็นป้ายของข้อความ เพื่อใช้แสดงข้อความต่างๆ ผู้ใช้งานทราบ เช่น ป้ายชื่อผู้ใช้งาน (User label) ป้ายชื่อรหัสผ่าน (Password label) เป็นต้น

รูปแบบคำสั่งสำหรับการสร้าง Label คือ

```
l = Label( root, option = value, ... )
```

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Label

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
anchor	กำหนดตำแหน่งการวาง Label โดยการกำหนดทิศคือ N, W, S, E เป็นต้น (อ่านค่าดีฟอลต์คือวางกลางวินโดวส์ เช่น Label(root, anchor = NW, text = "User Name:")
text	กำหนดข้อความให้แสดงบน Label เช่น Label(root, text = 'User Name:', font = ("Helvetica", 9), fg = "red")
bitmap	กำหนดรูปภาพบิตแม็พที่ต้องการแสดงบน Label เช่น Label(root, bitmap = "error", fg = "red")
textvariable	กำหนดตัวแปรสำหรับใช้เก็บข้อความของ Label โดยทำงานร่วมกับคลาส StringVar() เช่น labelVar = StringVar() Label(root, textvariable = labelVar, text = 'User Name:')

คลาส StringVar() และคลาส IntVar() มีเมธอดสำคัญที่ช่วยสนับสนุนการทำงานของ Widgets คือ

- เมธอด get() ทำหน้าที่ดึงข้อมูลจากตัวแปรคลาส StringVar() และ IntVar() มาแสดงผล เช่น

```
stringVariable = StringVar()

Label(root, text="User Name:", textvariable = stringVariable)

print(stringVariable.get())
```

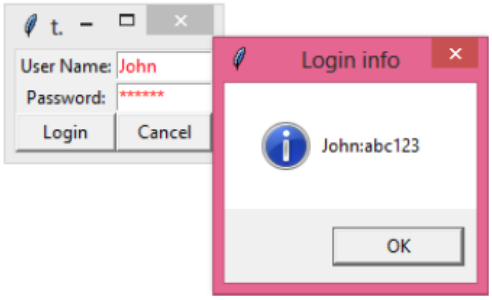
- เมธอด set() ทำหน้าที่กำหนดข้อความใหม่ให้กับ Label เช่น

```
stringVariable = StringVar()

Label(root, text="User Name:", textvariable = stringVariable)

print(stringVariable.set("User:"))
```

ตัวอย่างการสร้างและใช้งาน Label

```
from tkinter import *
from tkinter import messagebox
root = Tk()

userVar = StringVar()
passwdVar = StringVar()

def showLoginInfo():
   msg = userVar.get() + ":" + passwdVar.get()
   print(msg)
   messagebox.showinfo("Login info", msg)
def clearLogin():
   userVar.set("")
   passwdVar.set("")

L1 = Label(root, text='User Name:').grid(column=0, row=0, sticky=(W, E))
E1 = Entry(root, width=10, fg="red", textvariable=userVar).grid(column=1, row=0, sticky=(W, E))
L2 = Label(root, text='Password:').grid(column=0, row=1, sticky=(W, E))
E2 = Entry(root, width=10, show="*", fg="red", textvariable=passwdVar).grid(column=1, row=1, sticky=(W, E))
B1 = Button(root, text='Login', command=showLoginInfo).grid(column=0, row=2, sticky=(W, E))
B2 = Button(root, text='Cancel', command=clearLogin).grid(column=1, row=2, sticky=(W, E))
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรมดังรูป



จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Label บรรทัดที่ 5 และ 6 สร้างตัวแปรชื่อ userVar และ passwordVar เพื่อใช้สำหรับเก็บค่าข้อมูลชนิดสตริงที่ป้อนลงใน Entry ของ User Name และ Password ตามลำดับ บรรทัดที่ 8 สร้างฟังชันชื่อ showLoginInfo() ทำหน้าที่แสดงชื่อผู้ใช้งานและรหัสผ่านด้วยกล่องข้อความชื่อ "Login info" และพิมพ์ข้อมูลออกทาง Python shell ด้วย บรรทัดที่ 12 สร้างฟังชันชื่อ clearLogin() ทำหน้าที่ลบชื่อผู้ใช้และรหัสผ่านออกจาก Entry

บรรทัดที่ 16 โปรแกรมสร้าง Label เพื่อพิมพ์คำว่า 'User Name:' ออกจอภาพ บรรทัดที่ 17 สร้าง Entry ที่มีความกว้างของเท่ากับ 10, ตัวอักษรที่ป้อนจะถูกกำหนดให้เป็นสีแดง และข้อมูลที่ป้อนจะเก็บไว้ในตัวแปรชื่อ userVar (textvariable=userVar) บรรทัดที่ 18 สร้าง Label เพื่อพิมพ์คำว่า 'Password:' ออกจอภาพ บรรทัดที่ 19 สร้าง Entry ที่มีความกว้างของเท่ากับ 10 ข้อมูลที่ป้อนจะถูกเข้ารหัสด้วยตัวอักษร '*' และเก็บข้อมูลไว้ในตัวแปรชื่อ passwdVar บรรทัดที่ 20 สร้างปุ่มที่มีข้อความบนปุ่มคือ 'Login' เมื่อปุ่มดังกล่าวถูกกด โปรแกรมจะเรียกฟังชัน showLoginInfor() เข้ามาทำงานทันที บรรทัดที่ 21 สร้างปุ่มมีข้อความคือ 'Cancel' เมื่อปุ่มดังกล่าวถูกกด โปรแกรมจะเรียกฟังชัน clearLogin() ผลลัพธ์แสดงดังรูปด้านบน

7. Listbox คือ Widget ที่มีลักษณะเป็นรายการของสมาชิกที่ส่วนใหญ่มีตัวเลือกมากกว่า 1 ตัวเลือก โดยผู้ใช้งานสามารถเลือกสมาชิกจากรายการดังกล่าวได้เพียงตัวเดียวเท่านั้น เช่น ประเทศ คำนำหน้าชื่อ เป็นต้น

รูปแบบคำสั่งสำหรับการสร้าง Listbox คือ

```
l = Listbox( root, option=value, ... )
```

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Listbox

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
selectbackground	กำหนดสีข้อความใน Listbox เมื่อสมาชิกถูกเลือก เช่น Listbox(root, selectbackground = "red")
selectmode	กำหนดคุณสมบัติว่าจะเลือกสมาชิกจาก Listbox อย่างไร โดยมีโหมดให้เลือกดังนี้คือ **- BROWSE** เลือกสมาชิกได้เพียงตัวเดียว เมื่อลากเมาส์ (drag) แท็บสี (highlight) จะวิ่งไปพร้อมๆ กับเมาส์ **- SINGLE** เลือกสมาชิกได้เพียงตัวเดียว และไม่สามารถลากเมาส์ได้ **- MULTIPLE** เลือกสมาชิกได้มากกว่า 1 ตัว โดยการคลิกเลือกสมาชิกแต่ละตัว ไม่สามารถลากเมาส์เพื่อเลือกสมาชิกหลายๆ ตัว พร้อมกันได้ **- EXTENDED** เลือกสมาชิกได้มากกว่า 1 ตัว โดยการลากเมาส์เพื่อเลือกสมาชิกหลายๆ ตัว พร้อมกันได้ เช่น Listbox(root, selectmode = EXTENDED)
xscrollcommand	สร้างแท็บสไลด์แนวนอน (HORIZONTAL) ให้กับ Listbox เช่น xscrollbar = Scrollbar(root, orient = VERTICAL) xscrollbar.pack(side = BOTTOM, fill = X) Listbox(root, selectmode = EXTENDED, xscrollcommand = xscrollbar.set)
yscrollcommand	สร้างแท็บสไลด์แนวตั้งให้กับ Listbox (VERTICAL) เช่น yscrollbar = Scrollbar(root, orient = VERTICAL) yscrollbar.pack(side = RIGHT, fill = Y) Listbox(root, selectmode = EXTENDED, yscrollcommand = yscrollbar.set)

- เมธอด selection_set(first, last) ทำหน้าที่เลือกช่วงสมาชิกที่ต้องการ โดยใช้กำหนดตำแหน่งสมาชิกตัวแรกในพารามิเตอร์ first และสมาชิกตัวสุดท้ายคือ last ถ้าระบุเฉพาะ first แสดงว่าเลือกสมาชิกจาก Listbox เพียงตัวเดียวเท่านั้น เช่น

```
Lb1 = Listbox(root)

Lb1.insert(1, "Python")

Lb1.insert(2, "Perl")
```

```
Lb1.pack()
Lb1.selection_set(1)
Lb1.selection_set(0, END)
```

- เมธอด curselection() ทำหน้าที่ดึงข้อมูลสมาชิกที่ถูกเลือกใน Listbox ค่าที่ส่งกลับจะเป็นข้อมูลชนิด Tuple แต่ถ้าสมาชิกใน Listbox ไม่ได้ถูกเลือกไว้จะส่งค่ากลับเป็นทัพเพิลที่ว่างเปล่า เช่น

```
Lb1 = Listbox(root)
Lb1.insert(1, "Python")
Lb1.insert(2, "Perl")
Lb1.insert(3, "C")
Lb1.pack()
Lb1.selection_set(0, 1)
print(Lb1.curselection())
```

ผลลัพธ์ที่ได้คือ (0, 1)

- เมธอด delete(first, last = None) ลบสมาชิกออกจาก Listbox ถ้ากำหนดตำแหน่ง first และ last แสดงว่าเป็นการลบสมาชิกแบบช่วง แต่ถ้ากำหนดเฉพาะ first จะเป็นการลบสมาชิกจาก Listbox เพียงตัวเดียวเท่านั้น เช่น

```
Lb1 = Listbox(root)
Lb1.insert(1, "Python")
Lb1.insert(2, "Perl")
Lb1.insert(3, "C")
Lb1.pack()
Lb1.delete(1, 2)
```

- เมธอด get(first, last = None) ดึงสมาชิกจาก Listbox ถ้ากำหนดตำแหน่ง first และ last แสดงว่าเป็นการดึงสมาชิกแบบช่วง แต่ถ้ากำหนดเฉพาะ first จะเป็นการดึงสมาชิกจาก Listbox เพียงตัวเดียวเท่านั้น ค่าที่ส่งคืนจากเมธอดดังกล่าวเป็นชนิด Tuple เช่น

```
Lb1 = Listbox(root)
Lb1.insert(1, "Python")
Lb1.insert(2, "Perl")
Lb1.insert(3, "C")
Lb1.pack()
Lb1.get(1, 2)
```

ผลลัพธ์ที่ได้คือ ('Perl', 'C')

- เมธอด index(i) กำหนดตำแหน่งตัวชี้ด้วย index เช่น

```
Lb1.index(1)
```

- เมธอด insert(index, *elements) เพิ่มสมาชิกตั้งแต่ 1 – n ตัว ลงใน Listbox โดยสามารถกำหนดตำแหน่งที่จะเพิ่มด้วย index ถ้าต้องการเพิ่มสมาชิกในตำแหน่งท้ายของ Listbox ให้ทำการกำหนด index เป็น END เช่น

```
Lb1 = Listbox(root)
Lb1.insert(1, "Python")
Lb1.insert(2, "Perl")
Lb1.insert(3, "C")
Lb1.pack()
elements = ("C++", "Prolog")
for item in elements:
    Lb1.insert(END, item)
```

- เมธอด nearest(y) ดึงค่าตำแหน่งบนสุดของ Listbox เช่น

Lb1.**nearest(1)**

- เมธอด see(index) เปลี่ยนตำแหน่งของ Listbox โดยอ้างอิงจาก index เช่น

Lb1.**see(2)**

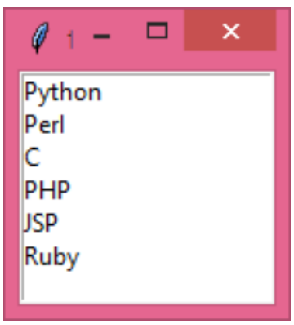
- เมธอด size() คำนวณจำนวนสมาชิกที่มีทั้งหมดใน Listbox เช่น

Lb1.**size()**

ตัวอย่างการสร้างและใช้งาน Listbox

```
from tkinter import *
root = Tk()


Lb1 = Listbox(root)
Lb1.insert(1, "Python")
Lb1.insert(2, "Perl")
Lb1.insert(3, "C")
Lb1.insert(4, "PHP")
Lb1.insert(5, "JSP")
Lb1.insert(6, "Ruby")
Lb1.pack()
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



ตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Listbox บรรทัดที่ 4 สร้าง Listbox ในหน้าต่างหลัก บรรทัดที่ 5 – 10 เป็นการเพิ่มสมาชิกให้กับ Listbox คือ 'Python', 'Perl', 'C', 'PHP', 'JSP' และ 'Ruby' ผลลัพธ์ที่ได้แสดงดังรูปด้านบน

8. Menubutton คือ Widget ที่มีลักษณะเป็นเมนูแบบเลื่อนลง เมื่อผู้ใช้คลิกเลือกเมนูดังกล่าวจะคงอยู่ตลอดไปจนกว่าจะปิดโปรแกรม ผู้ใช้สามารถเลือกรายการใดรายการหนึ่งใน Menubutton โดยการคลิกที่รายการที่ต้องการ

รูปแบบคำสั่งสำหรับการสร้าง Menubutton คือ

```
me = Menubutton( root, option=value, ... )
```

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Menubutton

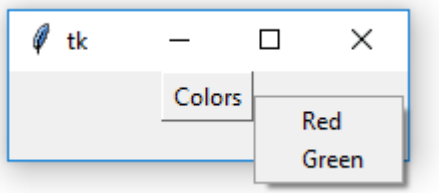
Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ activebackground, activeforeground, anchor, bg, bitmap, bd, cursor, font, fg, heigh, width, image, justify, padx, pady, relief, state, text, textvariable, underline และ wraplength สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
direction	กำหนดตำแหน่งของเมนูที่ต้องการแสดงผล ถ้ากำหนดเป็น LEFT เมนูจะปรากฏทางด้านซ้ายของปุ่มที่กำลังแสดงผล ถ้ากำหนดเป็น RIGHT เมนูจะปรากฏทางด้านขวาของปุ่ม ถ้ากำหนดเป็น "above" เมนูจะแสดงด้านบนของปุ่ม สำหรับค่าดีฟอลต์คือ "below" เช่น Menubutton(root, text = "Colors", relief = RAISED, direction = RIGHT)
disabledforeground	สีของตัวอักษรใน Menubutton จะไม่ถูกใช้งาน เช่น Menubutton(root, text = "Colors", disabledforeground = "black")
menu	กำหนดเมนูย่อยที่สัมพันธ์กับ Menubutton เช่น mb = Menubutton(root, text = "Colors", relief = RAISED) mb.menu.add_checkbutton(label = "Red") mb.menu.add_checkbutton(label = "Green")

ตัวอย่างการสร้างและใช้งาน Menubutton

```
from tkinter import *
root = Tk()

mb = Menubutton(root, text="Colors", relief=RAISED, direction=RIGHT)
mb.grid()
mb.menu = Menu(mb, tearoff=0)
mb["menu"] = mb.menu

mb.menu.add_checkbutton(label="Red")
mb.menu.add_checkbutton(label="Green")
mb.pack()
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Menubutton บรรทัดที่ 4 สร้าง Menubutton ชื่อ mb โดยเมนูดังกล่าวมีข้อความคือ "Colors" และเมนูย่อยจะปรากฏทางด้านขวาของเมนู "Colors" บรรทัดที่ 6 สร้างเมนูย่อยภายใน Menubutton บรรทัดที่ 9 และ 10 เพิ่มรายการในเมนูย่อยชื่อ "Red" และ "Green" ตามลำดับ

9. Message (ข้อความ) คือ Widget ที่มีลักษณะเป็นข้อความเพื่ออธิบายบางสิ่งบางอย่างในโปรแกรม (โดยแก้ไขไม่ได้) คล้ายกับ Label แต่แตกต่างกันคือ Message จะถูกปรับขนาดการแสดงผลให้เหมาะสมโดยอัตโนมัติ

รูปแบบคำสั่งสำหรับการสร้าง Message คือ

```
me = Message( root, option = value, ... )
```

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

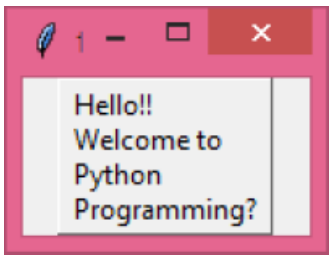
- option คือ คุณสมบัติต่างๆ ของ Message

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ anchor, bg, bitmap, bd, cursor, font, fg, height, image, justify, padx, pady, relief, text, textvariable, underline, width และ wraplength

ตัวอย่างการสร้างและใช้งาน Message

```
from tkinter import *
root = Tk()

var = StringVar()
me = Message(root, textvariable=var, relief=RAISED )

var.set("Hello!! Welcome to Python Programming?")
me.pack()
root.mainloop()
```

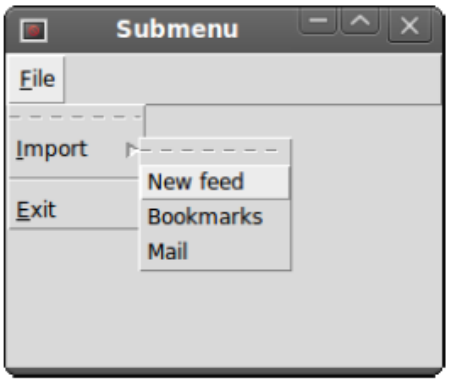
ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Message บรรทัดที่ 4 สร้างตัวแปรชื่อ var เป็นชนิดสตริง ใช้สำหรับเก็บข้อมูลของ Message บรรทัดที่ 5 สร้าง Message ชื่อ me โดยไม่มีข้อความใดๆ แสดงออกจอภาพ การควบคุมการแสดงผลจะขึ้นอยู่กับตัวแปรที่ถูกกำหนดใน textvariable นั่นคือ ตัวแปร

ชื่อ var นั่นเอง บรรทัดที่ 7 กำหนดข้อความใหม่ว่า " Hello!! Welcome to Python Programming?" โดยใช้เมธอด set() ให้กับตัวแปร var ส่งผลให้ Message เปลี่ยนเป็นข้อความใหม่ที่กำหนดขึ้นทันที

10. Menu คือ Widget ที่มีลักษณะเป็นเมนูย่อย แบ่งออกเป็น 3 ประเภทคือ เมนูแบบ popup, toplevel และ pull-down ตัวอย่างเช่น เมนู File, Edit, Option, Windows และ Help เป็นต้น



รูปแบบคำสั่งสำหรับการสร้าง Menu คือ

me = Menu(root, option=value, ...)

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Menu

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ activebackground, activeforeground, bg, bd, cursor, disabledforeground, font, fg, relief, image สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
activeborderwidth	กำหนดขนาดความกว้างของกรอบเมนู เมื่อผู้ใช้คลิกเลือกเมนู (ดีฟอลต์เท่ากับ 1 พิกเซล) เช่น Menu(root, activeborderwidth = 5)
postcommand	กำหนดให้เมนูเรียกใช้เมธอดหรือฟังชัน เมื่อมีผู้ใช้คลิกเลือกเมนูดังกล่าว เช่น Menu(root, postcommand = donothing)
selectcolor	กำหนดสีของปุ่ม checkbutton หรือ radiobutton เมื่อปุ่มเหล่านี้ถูกเลือก เช่น Menu(root, selectcolor = "red")

tearoff	โดยปกติเมื่อเพิ่มรายการของเมนูย่อยเข้าไปในเมนูหลักจะเพิ่มในตำแหน่งที่ 1 แต่เมื่อกำหนดให้ tearoff = 0 จะสามารถเพิ่มเมนูย่อยในตำแหน่งที่ 0 ได้ และเมนูย่อยนั่น ๆ จะสามารถแสดงผลเป็นอิสระจากเมนูหลักได้ เช่น Menu(menubar, tearoff = 1)
title	กำหนดข้อความ title ให้กับ Menu Widget

Widget ชนิด Menu มีเมธอดที่ช่วยสนับสนุนการทางาน คือ

- เมธอด add_command(options) สร้างเมนูย่อยในเมนูหลัก เช่น

```
menubar = Menu(root)
filemenu = Menu(menubar, tearoff = 0)
filemenu.add_command(label = "New", command = donothing)
```

- เมธอด add_radiobutton(options) สร้างเมนูชนิด checkbutton ในเมนูหลัก เช่น

```
menubar = Menu(root)
filemenu = Menu(menubar, tearoff = 0)
filemenu.add_radiobutton(label = "Exit", selectcolor = "red")
```

- เมธอด add_checkbutton(options) สร้างเมนูชนิด checkbutton ในเมนูหลัก เช่น

```
menubar = Menu(root)
filemenu = Menu(menubar, tearoff = 0)
filemenu.add_checkbutton(label = "Exit", selectcolor = "red")
```

- เมธอด add_cascade(options) เพิ่มชุดของเมนูย่อยที่เรียงต่อเนื่องกันในเมนูหลัก เช่น

```
menubar = Menu(root)
filemenu = Menu(menubar, tearoff = 0)
filemenu.add_command(label = "New", command = donothing)
```

```
filemenu.add_command(label = "Open", command = donothing)

filemenu.add_command(label = "Save", command = donothing)

menubar.add_cascade(label = "File", menu = filemenu)
```

- **เมธอด add_separator()** สร้างเส้นเพื่อแยกเมนูย่อยออกจากกัน เช่น

```
filemenu.add_separator()
```

- **เมธอด add(type, options)** เพิ่มเมนูย่อยเข้าไปยังเมนูหลักแบบต่อท้าย (append) โดย type คือ ชนิดของเมนูต่างๆ เช่น cascade (submenu), checkbutton, radiobutton, หรือ separator และ options เช่น font, foreground หรือ image เป็นต้น

- **เมธอด delete(startindex [,endindex])** ลบช่วงรายการในเมนูย่อย โดยระบุรายการแรกใน startindex และรายการตัวสุดท้ายใน endindex เช่น

```
editmenu.delete(2, 4)
```

- **เมธอด entryconfig(index, options)** เมื่อเมนูย่อยถูกสร้างขึ้นแล้ว สามารถแก้ไขคุณสมบัติของเมนูย่อยเหล่านั้น ผ่านเมธอด entryconfig() โดยอ้างด้วย index เช่น

```
filemenu.entryconfig(1, label="Test")
```

- **เมธอด index(item)** ส่งค่ากลับเป็นตำแหน่งของเมนูย่อยที่เลือก เช่น

```
filemenu.index(2)
```

- **เมธอด insert_separator(index)** เพิ่มเส้นสำหรับแบ่งเมนูย่อย โดยการระบุตำแหน่งที่ต้องการแทรกเส้นดังกล่าว เช่น

```
filemenu.insert_separator(2)
```

- **เมธอด invoke(index)** เรียกใช้คำสั่งที่มีหน้าที่เกี่ยวข้องกับเมนูที่เรียกใช้งาน ถ้าเป็นเมนูแบบ checkbutton เมธอดดังกล่าวจะทำหน้าที่สลับระหว่างปุ่มถูกเลือก (set) หรือถูกยกเลิก (cleared) ถ้าเมนูเป็นแบบ radiobutton จะถูกเซ็ตหรือยกเลิก เช่น

```
filemenu.invoke(2)
```

- **เมธอด type(index)** ส่งค่ากลับเป็นชนิดของเมนูย่อย เช่น ascade", "checkbutton", "command", "radiobutton", "separator", or "tearoff" เป็นต้น เช่น

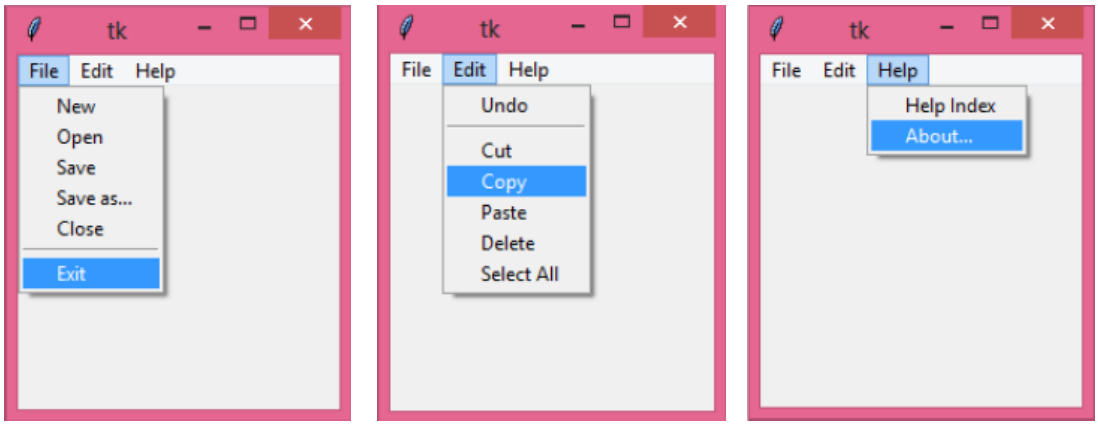
filemenu.**type**(2)

ตัวอย่างการสร้างและใช้งาน Menu

```
from tkinter import *

def donothing():
   filewin = Toplevel(root)
   button = Button(filewin, text="Do nothing button")
   button.pack()

root = Tk()
menubar = Menu(root)
#To create File menu
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="New", command=donothing)
filemenu.add_command(label="Open", command=donothing)
filemenu.add_command(label="Save", command=donothing)
filemenu.add_command(label="Save as...", command=donothing)
filemenu.add_command(label="Close", command=donothing)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=root.destroy)

menubar.add_cascade(label="File", menu=filemenu)
```

```
#To create Edit menu
editmenu = Menu(menubar, tearoff=0)
editmenu.add_command(label="Undo", command=donothing)
editmenu.add_separator()
editmenu.add_command(label="Cut", command=donothing)
editmenu.add_command(label="Copy", command=donothing)
editmenu.add_command(label="Paste", command=donothing)
editmenu.add_command(label="Delete", command=donothing)
editmenu.add_command(label="Select All", command=donothing)

menubar.add_cascade(label="Edit", menu=editmenu)
#To create Help menu
helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="Help Index", command=donothing)
helpmenu.add_command(label="About...", command=donothing)

menubar.add_cascade(label="Help", menu=helpmenu)

root.config(menu = menubar)
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งานเมนู บรรทัดที่ 3 สร้างฟังชันชื่อว่า donothing() ทำหน้าที่สร้างหน้าต่างวินโดวส์ใหม่ที่เป็นอิสระจากวินโดวส์หลัก โดยฟังชันดังกล่าวสร้างปุ่มและพิมพ์

ข้อความบนปุ่มว่า "Do nothing button" บรรทัดที่ 8 สร้างวินโดวส์หลักพร้อมกับเมนูหลักชื่อว่า menubar เพื่อใช้สำหรับรองรับเมนูย่อยที่จะสร้างขึ้นในคำสั่งลำดับถัดไป

บรรทัดที่ 11 สร้างเมนูย่อยชุดแรกมีชื่อว่า filemenu บนเมนูหลัก (Menubar) โดยเมนูย่อยดังกล่าวจะถูกเพิ่มในตำแหน่งแรกของเมนูหลักได้ (tearoff = 0) บรรทัดที่ 12 – 16 สร้างรายการของเมนูย่อยโดยเริ่มจาก "New", "Open", "New", "Save", …, "Close" ตามลำดับ บรรทัดที่ 17 สร้างเส้นสำหรับแบ่งหมวดหมู่ของเมนูย่อยออกจากกัน บรรทัดที่ 18 สร้างเมนูย่อยชื่อว่า "Exit" เมื่อผู้ใช้งานกดปุ่มดังกล่าว โปรแกรมจะยุติการทำงานทันที (command = root.destroy) บรรทัดที่ 20 เพิ่มเมนูย่อยที่สร้างขึ้นชื่อว่า filemenu เข้าไปยังเมนูหลัก ในลักษณะแบบลำดับชั้นคล้ายน้ำตก (Cascade) เมนูหลักดังกล่าวมีชื่อว่า "File"

บรรทัดที่ 23 สร้างเมนูหลักชื่อว่า "Edit" เพื่อรองรับเมนูย่อยๆ ที่ทำหน้าที่เกี่ยวกับการแก้ไขแฟ้ม เช่น "Cut", "Copy", "Paste" และเส้นแยก (separator) ตามลำดับ (บรรทัดที่ 24 - 30) ในบรรทัดที่ 32 เพิ่มรายการเมนูย่อยที่สร้างขึ้นในเมนูหลักชื่อ "Edit" ด้วยเมธอด add_cascade(label = "Edit", menu = editmenu) บรรทัดที่ 35 – 39 สร้างเมนูย่อยที่ทำหน้าที่เกี่ยวกับการช่วยเหลือ (Help) คำสั่งที่ใช้งานจะคล้ายกับการสร้างเมนู File และ Edit บรรทัดที่ 41 เป็นการกระตุ้นให้วินโดวส์หลักอัพเดทเมนูหลัก เพื่อให้ทุกอย่างทำงานได้อย่างถูกต้องด้วยเมธอด root.config(menu = menubar) ซึ่งจะต้องกระทำทุกครั้งเมื่อสร้างเมนูต่างๆ เสร็จเรียบร้อยแล้ว

11. Radiobutton คือ Widget ที่มีลักษณะเป็นปุ่มกลมมีช่องว่างอยู่ภายใน เมื่อถูกเลือกจะเปลี่ยนสถานะเป็นสีที่ทึบขึ้น Radiobutton ส่วนใหญ่จะถูกสร้างเป็นกลุ่มของตัวเลือก เพื่อให้ผู้ใช้งานสามารถเลือกรายการใดรายการหนึ่งเพียงรายการเดียวเท่านั้น เช่น เลือกคำนำหน้าชื่อ ชาย, นางสาว, นาง เป็นต้น

รูปแบบคำสั่งสำหรับการสร้าง Radiobutton คือ

```
r = Radiobutton( root, option = value, ... )
```

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Radiobutton

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ activebackground, activeforeground, anchor, bg, bitmap, borderwidth, command, cursor, font, fg, height, highlightbackground, highlightcolor, image, justify, padx, pady, relief, selectcolor, selectimage, state, text,

textvariable, underline, width, wraplength สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
value	กำหนดค่าให้กับ Radiobutton และจะสัมพันธ์กับ variable เช่น ถ้ากำหนดค่าใน value เท่ากับจำนวนเต็ม (int) ตัวแปร variable จะต้องเป็นจำนวนเต็มด้วย ในกรณีของสตริงก็มีลักษณะเช่นเดียวกัน เช่น var = IntVar(), str = StringVar() Radiobutton(root, text = "Mr.", variable = var, value = 1) Radiobutton(root, text = "Mr.", variable = str, value = "Mr")
variable	กำหนดตัวแปรที่ใช้สาหรับเก็บข้อมูลที่เกิดขึ้นจากการทำงานของ Radiobutton (ใช้ได้ทั้งสตริงและจำนวนเต็ม) สำหรับตัวอย่างเหมือนกับ value

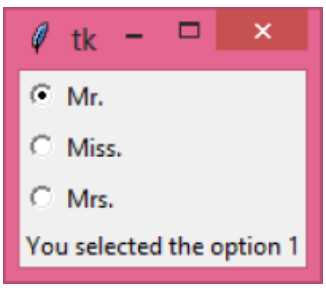
Widget ชนิด Radiobutton มีเมธอดที่ช่วยสนับสนุนการทา งาน คือ

- **เมธอด deselect()** เคลียร์ค่ารายการที่เลือกไว้ใน Radiobutton
- **เมธอด flash()** วาด Radiobutton ใหม่
- **เมธอด select()** กำหนดค่าให้กับ Radiobutton

ตัวอย่างการสร้างและใช้งาน Radiobutton

```
from tkinter import *

def sel():
   selection = "You selected the option " + str(var.get())
   label.config(text = selection)

root = Tk()
var = IntVar()
R1 = Radiobutton(root, text="Mr.", variable=var, value=1, command=sel)
R1.pack(anchor = W)

R2 = Radiobutton(root, text="Miss.", variable=var, value=2, command=sel)
R2.pack(anchor = W)

R3 = Radiobutton(root, text="Mrs.", variable=var, value=3, command=sel)
R3.pack(anchor = W)

label = Label(root)
label.pack()
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม

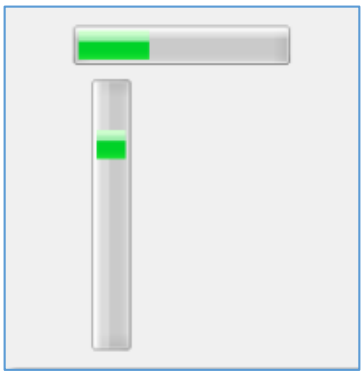


จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Radiobutton บรรทัดที่ 3 สร้างฟังชันชื่อว่า sel() ทำหน้าที่กำหนดข้อความใหม่ให้กับ Label เป็น "You selected the option" ตามด้วยค่าที่เก็บอยู่ในตัวแปร

var (var.get()) โดยใช้เมธอด Label.config() บรรทัดที่ 8 สร้างตัวแปรชื่อ var เป็นชนิดจำนวนเต็ม สำหรับเก็บข้อมูลที่เกิดขึ้นจากการดาเนินการใดๆ บน Radiobutton บรรทัดที่ 9 สร้าง Radiobutton ชื่อ R1 มีข้อความว่า "Mr." มีค่าเท่ากับ 1 (value = 1) เมื่อมีการคลิกเลือกปุ่ม Radiobuttion ดังกล่าว ผลลัพธ์จาก value จะถูกนามาเก็บไว้ในตัวแปรชื่อ var (variable = var) เมื่อปุ่มดังกล่าวถูกคลิกเลือก โปรแกรมจะเรียกฟังชัน ชื่อว่า sel() มาทำงานทันที (command = sel)

บรรทัดที่ 12 และ 16 สร้าง Radiobutton ชื่อ R1 และ R2 มีข้อความว่า "Miss." และ "Mrs." โดยมีค่าเท่ากับ 2 และ 3 ตามลำดับ เมื่อคลิกเลือกปุ่มทั้งสอง โปรแกรมจะเรียกใช้งานฟังก์ชัน sel() เช่นเดียวกับเมนู R1 บรรทัดที่ 18 สร้าง Label ชื่อ label เพื่อแสดงผลลัพธ์จากการคลิกเลือกปุ่มใน Radiobutton ออกจอภาพ ผลการทำงานของโปรแกรมแสดงดังรูปด้านบน

12. Scale คือ Widget ที่มีลักษณะเลื่อนสไลด์ขึ้นลงหรือซ้ายขวาได้ เพื่อทำหน้าที่แสดงขอบเขตของข้อมูลที่ผู้ใช้ต้องการ เช่น ปรับขนาดความเข้มของสี ความสว่าง ความคมชัด เป็นต้น



รูปแบบคำสั่งสำหรับการสร้าง Scale คือ

s = Scale(root, option = value, ...)

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Scale

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ activebackground, bg, bd, command, cursor, font, fg, highlightbackground, highlightcolor, length, relief, state, variable, width สา หรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
digits	เป็น Option ที่ใช้สำหรับแสดงรูปแบบของสเกล (scale) มี 3 รูปแบบ คือ จำนวนเต็ม (IntVar), จำนวนจริง (DoubleVar(float)), และสตริง (StringVar) เช่น Scale(root, digits = 4, orient = HORIZONTAL)
from_	เป็นเลขจำนวนเต็มหรือจำนวนจริงที่ใช้กำหนดขอบเขตเริ่มต้นของสเกล เช่น Scale(root, from_ = 0, to = 200, orient = HORIZONTAL)
label	แสดงข้อความกำกับสเกล ข้อความจะปรากฏที่มุมด้านซ้ายบนเมื่อสเกลเป็นชนิดแนวนอน, ข้อความจะปรากฏมุมด้านขวาบนเมื่อสเกลเป็นชนิดแนวตั้ง ค่าดีฟอลต์จะไม่แสดงข้อความ เช่น Scale(root, label = "Scale", orient = HORIZONTAL)
orient	กำหนด orient = HORIZONTAL เมื่อต้องการให้สเกลวางอยู่ในแนวนอน (แนวแกน x) และ orient = VERTICAL เมื่อต้องการสร้างสเกลในแนวตั้ง (แกน y) ค่าดีฟอลต์เป็นสเกลในแนวนอน (HORIZONTAL) เช่น Scale(root, orient = HORIZONTAL)
repeatdelay	ใช้กำหนดเวลาเพื่อหน่วงการเคลื่อนที่ของปุ่มในสเกล (เคลื่อนที่ขึ้น-ลง) ในกรณีที่ผู้ใช้คลิกในช่องของสเกลค้างไว้ (ดีฟอลต์ = 300) เช่น Scale(root, repeatdelay = 5, orient = HORIZONTAL)
resolution	กำหนดช่วงของสเกลเมื่อเพิ่มขึ้นหรือลดลง เช่น เมื่อกำหนดช่วงของสเกลเท่ากับ from_ = -1.0 ถึง to = 1.0 และกำหนด resolution = 0.5 สเกลจะเพิ่มขึ้นและลดลงดังนี้คือ -1.0, -0.5, 0.0, +0.5, และ +1.0 เช่น Scale(root, from_ = -1.0, to = 1.0, resolution = 0.5, orient = HORIZONTAL)
showvalue	โดยปกติค่าของสเกลจะแสดงผลร่วมกับแท็บสเกลเสมอ เมื่อไม่ต้องการแสดงค่าของสเกลให้กำหนด showvalue = 0 เช่น Scale(root, from_ = -1.0, to = 1.0, resolution = 0.5, showvalue = 0, orient = HORIZONTAL)
sliderlength	กำหนดขนาดของแท็บสไลด์ของสเกล (โดยปกติแท็บจะมีขนาดเท่ากับ 30 พิกเซล) เช่น Scale(root, sliderlength = 10, orient = HORIZONTAL)
takefocus	โดยปกติ สเกลจะโฟกัสเป็นแบบวงรอบ เมื่อไม่ต้องการพฤติกรรมดังกล่าวให้กำหนด takefocus = 0 เช่น Scale(root, from_ =1, to = 10, resolution = 1, takefocus = 0)
tickinterval	กำหนดการแสดงตัวเลขช่วงของสเกล เมื่อเป็นสเกลแนวนอนจะแสดงช่วงสเกลด้านล่าง แต่ถ้าเป็นสเกลแนวตั้งจะแสดงด้านซ้าย เช่น Scale(root, from_ = 1, to = 10, tickinterval = 1, orient = HORIZONTAL)

to	เป็นเลขจำนวนเต็มหรือจำนวนจริงที่ใช้กำหนดขอบเขตสิ้นสุดของสเกล เช่น Scale(root, from_ = 0, to = 200, orient = HORIZONTAL)
troughcolor	กำหนดสีของร่องหรือรางของสเกล เช่น Scale(root, from_ = 1, to = 10, troughcolor = "red", orient = HORIZONTAL)

Widget ชนิด Scale มีเมธอดที่ช่วยสนับสนุนการทำงาน คือ

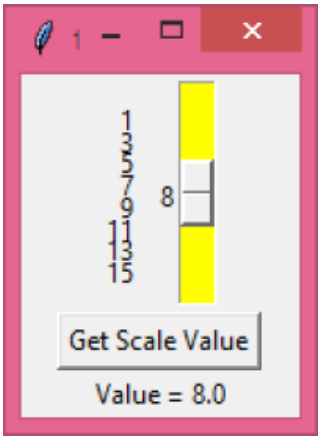
- **เมธอด get()** คืนค่าของสเกลปัจจุบันที่กำลังทำงานอยู่

- **เมธอด set(value)** กำหนดค่าสเกลใหม่

ตัวอย่างการสร้างและใช้งาน Scale

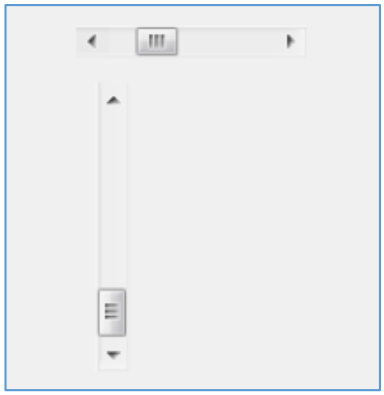
```
from tkinter import *
def sel():
   selection = "Value = " + str(var.get())
   label.config(text = selection)

root = Tk()
var = DoubleVar()
scale = Scale(root, from_=1, to=15, resolution=1, tickinterval=1,
          troughcolor="yellow", variable=var, orient=VERTICAL)
scale.pack(anchor=CENTER)

button = Button(root, text="Get Scale Value", command=sel)
button.pack(anchor=CENTER)

label = Label(root)
label.pack()
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Scale บรรทัดที่ 2 สร้างฟังชันชื่อว่า sel() ทำหน้าที่แสดงข้อความให้กับ Label มีค่าเท่ากับ "Value =" ตามด้วยค่าที่เก็บอยู่ในตัวแปร var (var.get()) โดยใช้เมธอด Label.config() บรรทัดที่ 7 สร้างตัวแปรชื่อ var เป็นชนิดจำนวนจริงขนาดใหญ่ (Double) สำหรับเก็บข้อมูลที่เกิดขึ้นจากการเลื่อนสเกล บรรทัดที่ 8 สร้าง Scale ชื่อ scale มีขอบเขตของสเกลตั้งแต่ 1 ถึง 15 (from_ = 1, to = 15), ช่วงของสเกลเท่ากับ 1 (resolution = 1), แสดงค่าของสเกลอยู่ด้านซ้ายตั้งแต่ 1 ถึง 15 (tickinterval = 1), รางของสเกลเป็นสีเหลือง (troughcolor = "yellow"), เป็นสเกลในแนวตั้ง (orient = VERTICAL), เมื่อผู้ใช้เลื่อนแท็บของสเกล ผลลัพธ์จะเก็บไว้ในตัวแปรชื่อ var (variable = var) บรรทัดที่ 12 สร้างปุ่มชื่อ button มีข้อความว่า "Get Scale Value" เมื่อคลิกปุ่มดังกล่าวโปรแกรมจะเรียกฟังชัน sel() มาทำงาน ผลการทำงานของโปรแกรมแสดงดังรูปด้านบน

13. Scrollbar คือ Widget ที่มีลักษณะเป็นแท็บสไลด์เลื่อนขึ้น-ลง หรือซ้าย-ขวาได้ เพื่อเพิ่มขนาดพื้นที่สำหรับแสดงผลหรือให้ผู้ใช้ป้อนข้อมูลเพิ่มขึ้น นิยมใช้งานร่วมกับ Listbox, Text, Canvas และ Entry เป็นต้น



รูปแบบคำสั่งสำหรับการสร้าง Scrollbar คือ

s = Scrollbar(root, option = value, ...)

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Scrolbar

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets (Scale) ที่ได้กล่าวมาแล้วข้างต้น คือ activebackground, bg, bd, command, cursor, orient, repeatdelay, takefocus, troughcolor, width สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
borderwidth	กำหนดขนาดของหัวลูกศรและแท็บสไลด์ของ Scrollbar เช่น Scrollbar(root, borderwidth = 5)
jump	กำหนดพฤติกรรมเมื่อเกิดการเคลื่อนที่ของ Scrollbar (แท็บ) เมื่อกำหนด jump = 0 จะทำให้สามารถเรียกใช้ command เพื่อเรียกใช้ฟังชันที่กำหนดไว้เข้ามาทำงาน, เมื่อกำหนด jump = 1 จะปิดการใช้งาน command เช่น Scrollbar(root, jump = 0, command = jumpCall)
repeatinterval	กำหนดระยะเวลาเมื่อผู้ใช้กดค้างที่รางของ Scrollbar ก่อนที่แท็บของ Scrollbar จะเคลื่อนที่ไปยังทิศทางที่ผู้ใช้ต้องการ ค่าดีฟอลต์เท่ากับ 300 มิลลิวินาที เช่น Scrollbar(root, repeatdelay = 100)

Widget ชนิด Scale มีเมธอดที่ช่วยสนับสนุนการทำงาน คือ

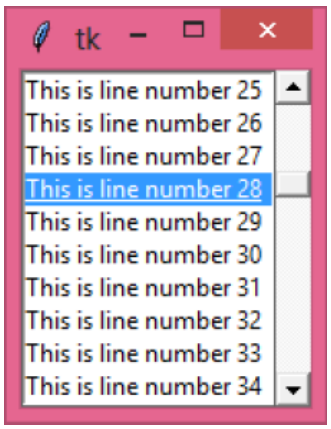
- เมธอด get() คืนค่าตำแหน่งปัจจุบันของแท็บที่กำลังทำงานอยู่ซึ่งมี 2 ค่าคือ (a, b) โดย a คือ ตำแหน่งที่อยู่ด้านซ้าย (Scrollbar เป็นชนิดแนวนอน) และด้านบน (Scrollbar เป็นชนิดแนวตั้ง) ของแท็บ และ b คือ คือตำแหน่งที่อยู่ด้านขวาและล่างของแท็บ เช่น print(scrollbar.get()) ผลลัพธ์คือ (0.41, 0.51) เมื่อ Scrollbar เป็นแนวตั้ง ค่า 0.41 คือตำแหน่งของแท็บด้านบน และ 0.51 คือตำแหน่งของแท็บด้านล่าง

- เมธอด set(first, last) ใช้สำหรับกรณีที่นำ Scrollbar ไปใช้กับ Widget ชนิดอื่นๆ โดย Widgets ที่ต้องการเรียกใช้เมธอดดังกล่าวจะเรียกผ่าน xscrollcommand หรือ yscrollcommand แทน ผลลัพธ์ที่ได้เหมือนกับการเรียกเมธอด get() นั่นเอง

ตัวอย่างการสร้างและใช้งาน Scrollbar

```
from tkinter import *

root = Tk()

scrollbar = Scrollbar(root)
scrollbar.pack(side = RIGHT, fill=Y)

mylist = Listbox(root, yscrollcommand = scrollbar.set)

for line in range(100):
   mylist.insert(END, "This is line number " + str(line))

mylist.pack(side = LEFT, fill = BOTH)
scrollbar.config(command = mylist.yview)

mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Scrollbar บรรทัดที่ 5 สร้าง Scrollbar ชื่อ scrollbar บรรทัดที่ 8 สร้าง Listbox ชื่อว่า mylist จากนั้นทำการเพิ่ม Scrollbar เข้าไปใน Listbox ในแนวตั้งหรือแนวแกน y ด้วยคำสั่ง yscrollcommand (yscrollcommand = scrollbar.set) บรรทัดที่ 10 โปรแกรมใช้ลูป for สั่งพิมพ์ "This is line number" และตามด้วยตัวเลขที่เริ่มตั้งแต่ 0 – 99 ลงใน Listbox โดยใช้เมธอด insert() ส่งผลให้ Scrollbar สร้างแท็บสไลด์มีขนาดที่ครอบคลุมรายการทั้งหมด (ถ้าสั่งพิมพ์รายการน้อยๆ เช่น 5 รายการ Scrollbar จะไม่สร้างแท็บสำหรับเลื่อนสไลด์ให้) บรรทัดที่ 14 โปรแกรมสั่งกระตุ้นให้ Scrollbar ทำงานด้วยเมธอด config() ผ่านอ๊อปชัน command ผลการทำงานของโปรแกรมแสดงดังรูปด้านบน

14. Text คือ Widget ที่อนุญาตให้ผู้เขียนโปรแกรมสามารถสร้างข้อความเพื่ออธิบายบางสิ่งบางอย่างในโปรแกรม โดย Text มีความสามารถหลายอย่าง เช่น เปลี่ยนสีพื้นข้อความ สีตัวอักษร รูปแบบฟอนต์ ขนาด และอื่นๆ

รูปแบบคำสั่งสำหรับการสร้าง Text คือ

```
t = Text( root, option = value, ... )
```

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Text

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ bg, bd, cursor, font, fg, height, highlightbackground, highlightcolor, padx, pady, relief, state, width, xscrollcommand, yscrollcommand สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
exportselection	โดยปกติเมื่อผู้ใช้เลือกข้อความภายใน Text ข้อมูลที่ถูกเลือกจะถูกส่งไปเก็บไว้ในคลิปบอร์ด (Clipboard) โดยอัตโนมัติ เมื่อไม่ต้องการให้ข้อมูลดังกล่าวถูกส่งไปยังคลิปบอร์ด ให้กำหนด exportselection = 0 เช่น Text(root, exportselection = 0)
highlightthickness	กำหนดขนาดของ Highlight Focus ของ Text ค่าดีฟอลต์เท่ากับ 1 แต่ถ้าไม่ต้องการให้ออฟชันดังกล่าวทำงานให้กำหนดเป็น 0 เช่น Text(root, highlightthickness = 0)
insertbackground	กำหนดสีของของเคอร์เซอร์ ณ ตำแหน่งปัจจุบัน เช่น Text(root, insertbackground = "red")

insertborderwidth	ขนาดของกรอบแบบ 3D รอบๆ เคอร์เซอร์ ค่าดีฟอลต์คือ 0
insertofftime	กำหนดเวลาให้เคอร์เซอร์หยุดทำงาน มีหน่วยเป็นมิลลิวินาที ค่าดีฟอลต์เท่ากับ 300 มิลลิวินาที ถ้าไม่ต้องการให้เคอร์เซอร์กระพริบ กำหนดให้ insertofftime = 0 เช่น Text(root, insertbackground = "red", insertofftime = 100)
insertontime	กำหนดเวลาให้เคอร์เซอร์ทำงาน (กระพริบ) มีหน่วยเป็นมิลลิวินาที ค่าดีฟอลต์เท่ากับ 600 มิลลิวินาที ถ้าไม่ต้องการให้เคอร์เซอร์ปรากฏ กำหนดให้ insertofftime = 0 เช่น Text(root, insertbackground = "red", insertofftime = 0)
insertwidth	กำหนดขนาดของเคอร์เซอร์ ค่าดีฟอลต์เท่ากับ 2 พิกเซล เช่น Text(root, insertbackground = "red", insertwidth = 10)
selectbackground	กำหนดสีพื้นหลังเมื่อผู้ใช้เลือกข้อความ เช่น Text(root, insertbackground = "red", selectbackground = "black")
selectborderwidth	กำหนดความกว้างของกรอบรอบๆ Text
spacing1	กำหนดขนาดความกว้างด้านบนของข้อความในแต่ละบรรทัด ค่าดีฟอลต์เท่ากับ 0 เช่น Text(root, spacing1 = 10)
spacing2	กำหนดระยะห่างระหว่างบรรทัดของข้อความใน Text ค่าดีฟอลต์เท่ากับ 0 เช่น Text(root, spacing2 = 10)
spacing3	กำหนดขนาดความกว้างด้านล่างของข้อความในแต่ละบรรทัด ค่าดีฟอลต์เท่ากับ 0 เช่น Text(root, spacing3 = 10)
tabs	กำหนดขนาดของแท็บในข้อความ หน่วยเป็นพิกเซล เช่น Text(root, tabs=100) (ให้ผู้ใช้คลิกที่ข้อความแล้วทดสอบกดที่ปุ่ม TAB)
wrap	กำหนดขนาดของข้อความที่จะถูกครอบ (Focus) ถ้ากำหนด wrap = WORD โปรแกรมครอบทีละคำ (แยกด้วยข้อความว่าง) แต่ถ้ากำหนดเป็น wrap = CHAR จะเป็นการครอบทีละตัวอักษรแทน เช่น Text(root, wrap = WORD)
xscrollcommand	กำหนด Scrollbar ให้กับ Text ในแนวนอน ใช้ในกรณีที่ข้อความมีขนาดความยาวมากๆ เช่น Text(root, xscrollcommand = xscrollbar.set, yscrollcommand = yscrollbar.set)
yscrollcommand	กำหนด Scrollbar ให้กับ Text ในแนวตั้ง เช่น Text(root, xscrollcommand = xscrollbar.set, yscrollcommand = yscrollbar.set)

Widget ชนิด Text มีเมธอดที่ช่วยสนับสนุนการทำงาน คือ

- เมธอด delete(startindex [,endindex]) ลบตัวอักษรหรือข้อความใน Text โดยระบุช่วงของข้อความ ถ้ากำหนดเฉพาะ startindex ข้อความจะถูกลบตั้งแต่ startindex ถึงตำแหน่งสิ้นสุดของข้อความ เช่น text.delete(1.0) โดย 1 หมายถึงบรรทัดที่ 1, 0 หมายถึงอักษรตัวที่ 0 (linenumber.character number)

- เมธอด get(startindex [,endindex]) คืนค่าเป็นตัวอักษรหรือข้อความจาก Text โดยใช้ startindex กำหนดตำแหน่งเริ่มต้นและตำแหน่งสุดท้ายของข้อความด้วย endindex เช่น print(text**.get**(1.0, END))

- เมธอด index(index) คืนค่าสมบูรณ์ (absolute value) จาก Text โดยใช้ index กำหนดตำแหน่ง เช่น print(text**.index**(4.0))

Text ยังมีเมธอดที่ช่วยสำหรับการทำ Marks, Tabs และ Indexes ดังนี้

- เมธอด index(mark) คืนค่าแถวและคอลัมน์ที่ผู้ใช้ทำเครื่องหมายไว้ (Marks) เช่น

```
text.mark_set("Python", INSERT)
print(text.index(INSERT))
```

ผลลัพธ์ที่ได้คือ 1.26 (แถวที่ 1, คอลัมน์ที่ 26)

- เมธอด mark_gravity(mark [,gravity]) กำหนดลักษณะทิศทางของการ Marks ซึ่งเป็นไปได้ 2 ทิศทาง คือ ทิศทางที่เป็นจุดอ้างอิงทางด้านขวา (RIGHT) หรือด้านซ้าย (LEFT) เช่น ข้อความว่า "This is Python Programming" เมื่อทำการ Marks ไว้กับข้อความเป็นคำว่า "Python" และทำการกำหนดทิศทางเป็น LEFT แสดงรูปแบบคำสั่ง ดังนี้

```
text.insert(INSERT, "This is Python Programming")
text.mark_set("Python", INSERT)
text.mark_gravity("Python", LEFT)
```

- เมธอด mark_names() คืนค่าข้อความทั้งหมดที่ Marks ไว้ เช่น print(text**.mark_names**())

- เมธอด mark_set(mark, index) กำหนดตำแหน่ง Marks ใหม่ เช่น

```
text.insert(INSERT, "This is Python Programming")
```

text**.mark_set**("Python", CURRENT)

- เมธอด mark_unset(mark) เคลียร์ค่าที่ Marks ไว้ เช่น text**.mark_unset**(CURRENT)

Text ยังมีเมธอดที่ช่วยสำหรับการทำงานเกี่ยวกับ Tag ดังนี้

- เมธอด tag_add(tagname, startindex[,endindex] ...) ทำหน้าที่กำหนดป้ายชื่อลงในข้อความใน Text โดย tagname คือชื่อของ tag, startindex คือตำแหน่งเริ่มต้นที่ต้องการเพิ่ม tag, endindex (option = ไม่ใส่ก็ได้) ตำแหน่งสุดท้ายที่ต้องการเพิ่ม tag เช่น text**.tag_add**("here", "1.0", "1.4") จากคำสั่งด้านบนเป็นการกำหนด tag ชื่อ "here" โดยขนาดของ tage มีความยาวตั้งแต่ตัวอักษรที่ 0 (.0) ถึง 4 (.4) และอยู่ในบรรทัดที่ 1 (1.) ของข้อความใน Text

- เมธอด tag_config ทำหน้าที่ปรับแต่งคุณสมบัติต่างๆ หลังจากที่สร้าง Text ขึ้นมาใช้งานแล้ว เช่น การกำหนดสีพื้นหลัง สีตัวอักษร ขัดเส้นใต้ เป็นต้น เช่น

text**.tag_config**("here", background="yellow", foreground="blue")

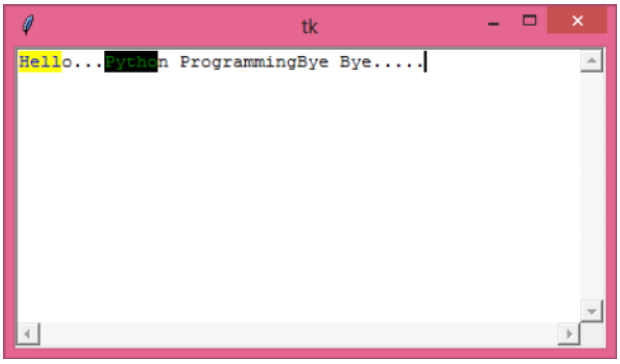
- เมธอด tag_delete(tagname) ทำหน้าที่ลบ tag จากที่เคยกำหนดไว้

- เมธอด tag_remove(tagname [,startindex[.endindex]] ...) ทำหน้าที่ลบ tag ออกจากพื้นที่ที่เคยกำหนดไว้ แต่ไม่ลบ Tag ที่นิยามเอาไว้ในโปรแกรม

ตัวอย่างการสร้างและใช้งาน Text

```
from tkinter import *
root = Tk()
frame = Frame(root, bd=2, relief=SUNKEN)
frame.grid_rowconfigure(0, weight=1)
frame.grid_columnconfigure(0, weight=1)
xscrollbar = Scrollbar(frame, orient=HORIZONTAL)
xscrollbar.grid(row=1, column=0, sticky=E+W)
yscrollbar = Scrollbar(frame)
yscrollbar.grid(row=0, column=1, sticky=N+S)

text = Text(frame, bd=0, xscrollcommand=xscrollbar.set, yscrollcommand=yscrollbar.set)
text.grid(row=0, column=0, sticky=N+S+E+W)

xscrollbar.config(command=text.xview)
yscrollbar.config(command=text.yview)

text.insert(INSERT, "Hello...Python Programming")
text.insert(END, "Bye Bye.....")
text.tag_add("here", "1.0", "1.4")
text.tag_add("start", "1.8", "1.13")
text.tag_config("here", background="yellow", foreground="blue")
text.tag_config("start", background="black", foreground="green")
frame.pack()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



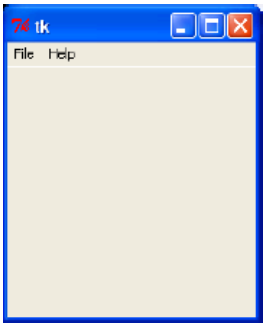
จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Text บรรทัดที่ 3 สร้างเฟรมชื่อ frame มีความหนาของกรอบเท่ากับ 2 พิกเซล เป็นชนิดกรอบแบบร่องลึก บรรทัดที่ 4 และ 5 สร้างกริดบนเฟรมมีขนาดเท่ากับ 1 แถว 1 คอลัมน์ บรรทัดที่ 6 และ 7 สร้าง Scrollbar ในแนวนอนลงบนเฟรม วางในตำแหน่งแถวที่ 1 และคอลัมน์ที่ 0 ในกริด และวาง Scrollbar จากด้านทิศตะวันออกไปทิศตะวันตก (sticky=E+W) บรรทัดที่ 8 และ 9 สร้าง Scrollbar ในแนวตั้งลงบนเฟรม วางในตำแหน่งแถวที่ 0 และคอลัมน์ที่ 1 ในกริด และวาง Scrollbar จากด้านทิศเหนือไปทิศใต้ (sticky=N+S)

บรรทัดที่ 11 สร้าง Text ลงบนเฟรม โดยมี Scrollbar อยู่ด้านขวา (yscrollcommand) และด้านล่าง (xscrollcommand) ของ Text ในบรรทัดที่ 12 Text จะถูกวางลงในแถวที่ 0 คอลัมน์ 0 ของกริดใน 4 ทิศทาง (row=0, column=0, sticky=N+S+E+W) บรรทัดที่ 14 และ 15 ออกคำสั่งให้ scrollbar แสดงผลใน Text

บรรทัดที่ 17 เพิ่มข้อความว่า "Hello...Python Programming" ลงใน Text โดยใช้ตำแหน่งที่ Marks ไว้คือ INSERT (โดยปกติ INSERT Mark จะอยู่ในตำแหน่งด้านท้ายของข้อความใน Text สามารถเปลี่ยนจุด Marks ของ INSERT ใหม่ได้โดยใช้เมธอด mark_set() ที่อธิบายไว้ในหัวข้อที่ผ่านมา) บรรทัดที่ 18 เพิ่มข้อความว่า " Bye Bye....." ลงใน Text โดยใช้ตำแหน่งที่ Marks ไว้คือ END ซึ่งเป็นตำแหน่งสุดท้ายในข้อความ ผลลัพธ์จากการทำงานของบรรทัดที่ 17 และ 18 คือข้อความว่า "Hello...Python ProgrammingBye Bye....." บน Text

บรรทัดที่ 19 สร้าง tag ชื่อ "here" โดยครอบคลุมตัวอักษรตั้งแต่ตัวที่ 0 – 3 ของบรรทัดที่ 1 เอาไว้ (" here", "1.0", "1.4") คือข้อความว่า "Hell" บรรทัดที่ 20 สร้าง tag ชื่อ "start" โดยครอบคลุมตัวอักษรตั้งแต่ตัวที่ 8 – 13 เอาไว้ คือข้อความว่า "Python" บรรทัดที่ 21 สั่งให้ระบายสี tage ชื่อ "here" โดยมีสีของพื้นหลังเป็นสีเหลือง และตัวอักษรเป็นสีน้ำเงิน บรรทัดที่ 22 สั่งให้ระบายสี tage ชื่อ "start" โดยมีสีของพื้นหลังเป็นสีดำ และตัวอักษรเป็นสีเขียวตามลำดับ ผลการทำงานของโปรแกรมแสดงดังรูปด้านบน

15. Toplevel เป็น Widget ที่อยู่บนสุดของวินโดวส์ ไม่จำเป็นต้องมีวินโดวส์อื่นๆ คอยควบคุมหรืออยู่ภายใต้วินโดวส์ใดๆ หรือพูดง่ายๆ คือ Toplevel จะถูกดูแลจาก Window Manager โดยตรงนั่นเอง ดังแสดงดังรูป



รูปแบบคำสั่งสำหรับการสร้าง Toplevel คือ

t = Toplevel(root, option=value, ...)

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Toplevel

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ bg, bd, cursor, font, fg, height, relief, width เป็นต้น

Widget ชนิด Toplevel มีเมธอดที่ช่วยสนับสนุนการทำงาน คือ

- **เมธอด deiconify()** แสดงวินโดวส์ทันที เมื่อเรียกใช้เมธอดดังกล่าว เช่น

top = Toplevel()

top**.deiconify()**

- **เมธอด frame()** คืนค่า system-specific window identifier เช่น

print(text**.get**(1.0, END))

ผลลัพธ์คือ 0x2403c2

- **เมธอด group(window)** จัดกลุ่มของวินโดวส์

- **เมธอด iconify()** แปลงวินโดวส์ไปเป็น icon

- **เมธอด protocol(name, function)** ลงทะเบียนฟังชันเพื่อทำหน้าที่เป็น callback ฟังชัน

- **เมธอด state()** คืนค่าสถานะปัจจุบันของวินโดวส์ ค่าที่เป็นไปได้คือ Normal, iconic, withdrawn และ icon

- **เมธอด withdraw()** ลบวินโดวส์ออกจากจอภาพ

- **เมธอด maxsize(width, height)** กำหนดขนาดวินโดวส์ที่ใหญ่ที่สุด

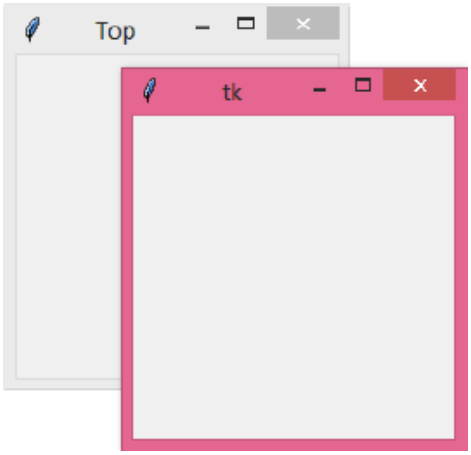
- **เมธอด minsize(width, height)** กำหนดขนาดวินโดวส์ที่เล็กที่สุด

- **เมธอด title(string)** กำหนด title ของวินโดวส์ที่สร้างขึ้น

ตัวอย่างการสร้างและใช้งาน Text (ตัวอย่างที่ 1)

```
from tkinter import *


root = Tk()
top = Toplevel()
top.title("Toplevel")
top.deiconify()


top.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม

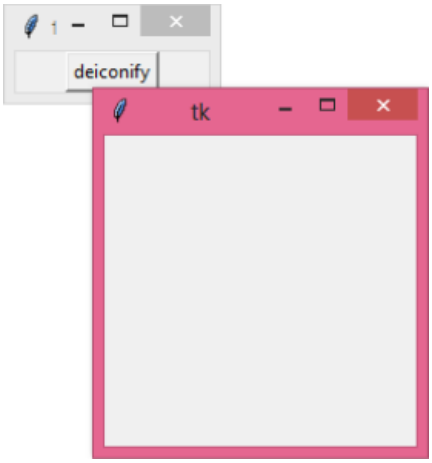


จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Toplevel บรรทัดที่ 3 สร้างวินโดวส์หลักหรือ root window ชื่อว่า root บรรทัดที่ 4 สร้างวินโดวส์หลักที่เป็นอิสระจาก root ชื่อว่า top บรรทัดที่ 5 กำหนด title ของ top เท่ากับ "Toplevel" บรรทัดที่ 6 สั่งวาดวินโดวส์ใหม่ด้วยเมธอด deiconify() ผลลัพธ์แสดงดังรูปด้านบน

ตัวอย่างการสร้างและใช้งาน Text (ตัวอย่างที่ 2)

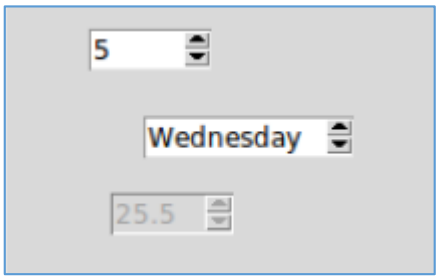
```
from tkinter import *

root = Tk()
root.withdraw()

top = Toplevel(root)
top.protocol("WM_DELETE_WINDOW", root.destroy)

but = Button(top, text='deiconify')
but['command'] = root.deiconify
but.pack()

root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



ตัวอย่างโปรแกรม บรรทัดที่ 3 สร้างหน้าต่างหลักชื่อ root จากนั้นบรรทัดที่ 4 โปรแกรมลบวินโดวส์ด้วยเมธอด withdraw() บรรทัดที่ 6 โปรแกรมสร้างวินโดวส์ใหม่ชื่อ top โดยสืบทอดคุณสมบัติทั้งหมดมาจากวินโดวส์ root บรรทัดที่ 7 โปรแกรมทำการลงทะเบียนเมธอด root.destroy กับ "WM_DELETE_WINDOW" เพื่อใช้สำหรับลบหรือทาลายวินโดวส์ออกจากจอภาพ บรรทัดที่ 9 สร้างปุ่มชื่อ but มีข้อความบนปุ่มคือ 'deiconify' บรรทัดที่ 10 เป็นการกำหนดว่าเมื่อคลิกปุ่มดังกล่าว โปรแกรมจะเรียกเมธอด root.deiconify เพื่อยุติการทำงานของวินโดวส์ ผลลัพธ์แสดงดังรูปด้านบน

16. Spinbox คือ Widget ที่เหมือนกับ Entry แต่สามารถกำหนดขอบเขตของข้อมูลได้ ดังแสดงในรูป



รูปแบบคำสั่งสำหรับการสร้าง Spinbox คือ

```
s = Spinbox( root, option=value, ... )
```

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ Spinbox

Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ activebackground, bg, bd, command, cursor, disabledbackground, disabledforeground, font, fg, justify, relief, repeatdelay, repeatinterval, state, textvariable, width, wrap, xscrollcommand สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
format	กำหนดรูปแบบของสตริงในการแสดงผล เช่น Spinbox(root, from_= 0, to = 10, format = '%10.4f')
from_	กำหนดจำนวนบรรทัดเริ่มต้นของ spinbox เช่น Spinbox(root, from_ = 0, to = 10, format = '%10.4f')
to	กำหนดจำนวนบรรทัดสุดท้ายของ spinbox เช่น Spinbox(root, from_ = 0, to = 10, format = '%10.4f')

Widget ชนิด Spinbox มีเมธอดที่ช่วยสนับสนุนการทำงาน คือ

- เมธอด delete(startindex [,endindex]) ลบตัวอักษรหรือข้อความใน Spinbox โดยระบุช่วงของข้อความ ถ้ากำหนดเฉพาะ startindex ข้อความจะถูกลบตั้งแต่ startindex ถึงตำแหน่งสิ้นสุดของข้อความ เช่น spin.delete(1.0) โดย 1 หมายถึงบรรทัดที่ 1, 0 หมายถึงอักษรตัวที่ 0 (linenumber.character number)

- เมธอด get(startindex [,endindex]) คืนค่าเป็นตัวอักษรหรือข้อความจาก Spinbox โดยใช้ startindex กำหนดตำแหน่งเริ่มต้น และตำแหน่งสุดท้ายของข้อความด้วย endindex เช่น print(spin.get(1.0, END))

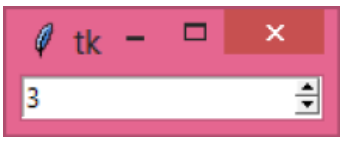
- เมธอด index(index) คืนค่าสมบูรณ์ (absolute value) จาก Spinbox โดยใช้ index กำหนดตำแหน่ง เช่น print(spin.index(4.0))

- เมธอด insert(index [,string]...) แทรกข้อความในตำแหน่งที่ระบุใน index เช่น spin.insert(1.0, "Python")

ตัวอย่างการสร้างและใช้งาน Spinbox

```
from tkinter import *
root = Tk()


spin = Spinbox(root, from_=0, to=10)
spin.pack()


root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน Spinbox ในบรรทัดที่ 4 สร้าง Spinbox ชื่อ spin มีจำนวนบรรทัดเท่ากับ 10 บรรทัด เพื่อให้ผู้ใช้งานสามารถป้อนข้อมูลได้เพิ่มขึ้น โดยการคลิกเลือกที่ลูกศรขึ้นและลงที่อยู่ทางด้านขวาของ Spinbox

17. LabelFrame คือ Widget ที่ผสมผสานกันระหว่าง Frame กับ Label นั่นคือ มีความสามารถในการรองรับ Widgets ต่างๆ เหมือนเฟรม และจัดการกับข้อความได้เหมือนกับ Label นั่นเอง

รูปแบบคำสั่งสำหรับการสร้าง LabelFrame คือ

lf1 = LabelFrame(root, option = value, ...)

พารามิเตอร์ คือ

- root คือ วินโดวส์หลัก (root window)

- option คือ คุณสมบัติต่างๆ ของ LabelFrame

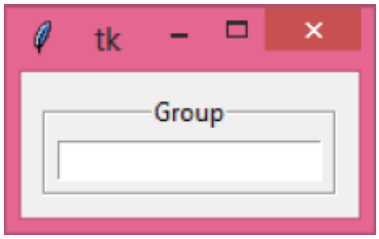
Option ดังต่อไปนี้มีคุณสมบัติการทำงานที่เหมือนกับ Widgets ที่ได้กล่าวมาแล้วข้างต้น คือ bg, bd, cursor, font, height, highlightbackground, highlightcolor, highlightthickness, relief, text, width สำหรับ option ที่แตกต่างจาก Widgets ตัวอื่นๆ แสดงในตารางด้านล่าง

Option	คำอธิบาย
labelAnchor	กำหนดตำแหน่งที่จะวางเลเบล ค่าดีฟอลต์คือ nw ซึ่งมีรูปแบบดังรูป เช่น LabelFrame(root, text = "Group", labelanchor="n", padx = 5, pady = 5)

ตัวอย่างการสร้างและใช้งาน LabelFrame

```
from tkinter import *
root = Tk()

lf1 = LabelFrame(root, text = "Group", labelanchor="n", padx = 5, pady = 5)
lf1.pack(padx=10, pady=10)

e = Entry(lf1)
e.pack()

root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน LabelFrame ในบรรทัดที่ 4 สร้าง LabelFrame ชื่อ lf1 โดยมีข้อความว่า "Group" วางอยู่ในทิศเหนือ (labelanchor = "n") ของวินโดวส์ บรรทัดที่ 7 สร้าง Entry และเพิ่มลงใน LabelFrame

18. MessageBox คือ Widget ที่ใช้สำหรับแสดงข้อความที่เหมาะสม หรือตามที่ผู้เขียนโปรแกรมต้องการ เช่น ข้อความเกี่ยวกับการกระทำที่ผิดพลาดของผู้ใช้งาน ข้อความแจ้งเตือนต่างๆ เป็นต้น

รูปแบบคำสั่งสำหรับการสร้าง MessageBox คือ

```
mb = messagebox.FunctionName(title, message [, options])
```

พารามิเตอร์ คือ

- title คือ title ของ messagebox

- message คือ ข้อความที่ต้องการแสดงใน messagebox

- options คือ คุณสมบัติต่างๆ ของ MessageBox เช่น ABORT, RETRY หรือ IGNORE

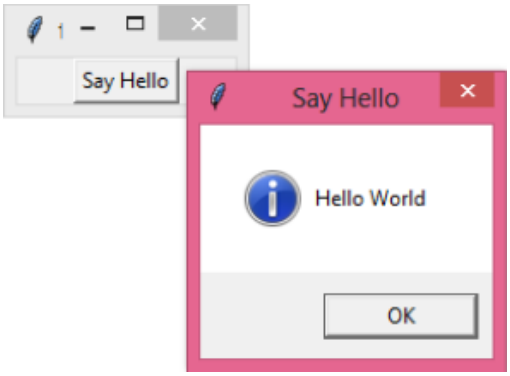
สำหรับฟังชัน (FunctionName) ที่สามารถใช้งานได้ดังนี้

- showinfo()
- showwarning()
- showerror()
- askquestion()
- askokcancel()
- askyesno()
- askretrycancel()

ตัวอย่างการสร้างและใช้งาน messagebox

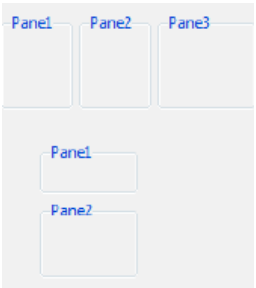
```
from tkinter import *
from tkinter import messagebox
root = Tk()
def hello():
   messagebox.showinfo("Say Hello", "Hello World")

B1 = Button(root, text = "Say Hello", command = hello)
B1.pack()

root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม

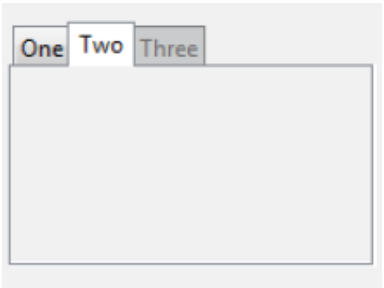


จากตัวอย่างโปรแกรม แสดงการสร้างและใช้งาน messagebox ในบรรทัดที่ 4 สร้างฟังชันชื่อ hello() โดยพิมพ์ข้อความว่า "Hello World" ผ่าน messagebox บรรทัดที่ 7 สร้างปุ่มชื่อ B1 มีข้อความว่า "Say Hello" เมื่อกดปุ่มดังกล่าว โปรแกรมจะเรียกฟังชัน hello() มาทำงาน

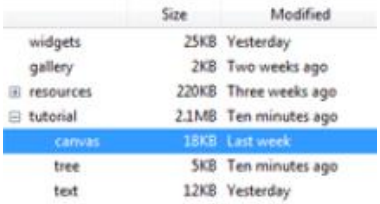
19. Widgets อื่นๆ ที่น่าสนใจ ไพธอนยังมี Widgets ที่น่าสนใจอื่นๆ เช่น Paned Windows, Notebook, Tree, Combobox, SizeGrip, Progressbar ซึ่งมีรูปแบบคือ



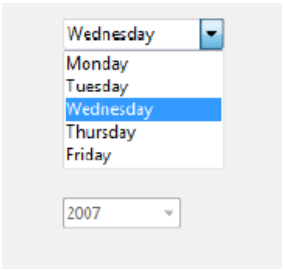
Paned Windows



Notebook

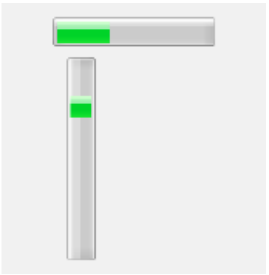


Tree



Combobox

SizeGrip



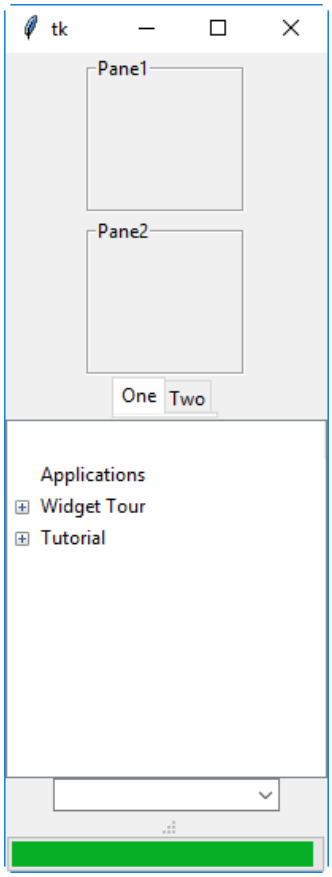
Progressbar

ตัวอย่างโปรแกรมแสดงการสร้างและใช้งาน Widgets ต่างๆ ที่แสดงไว้ในรูปด้านบน

```
from tkinter import *
from tkinter import ttk
root = Tk()

# PanedWindow
p = PanedWindow(root, orient=VERTICAL)
f1 = LabelFrame(p, text = 'Pane1', width = 100, height = 100)
f2 = LabelFrame(p, text = 'Pane2', width = 100, height = 100)
p.add(f1)
p.add(f2)
p.pack()

# Notebook
n = ttk.Notebook(root)
f1 = Frame(n)
f2 = Frame(n)
n.add(f1, text = 'One')
n.add(f2, text = 'Two')
n.pack()
```

```
#Treeview
tree = ttk.Treeview(root)
tree.insert('', 'end', 'widgets', text = 'Widget Tour')
tree.insert('', 0, 'gallery', text = 'Applications')
id = tree.insert('', 'end', text = 'Tutorial')
tree.insert('widgets', 'end', text = 'Canvas')
tree.insert(id, 'end', text = 'Tree')
tree.pack()

# Combobox
countryvar = StringVar()
country = ttk.Combobox(root, textvariable = countryvar)
country['values'] = ('USA', 'Canada', 'Australia')
country.pack()

# Sizegrip
sg = ttk.Sizegrip(root)
sg.pack()

# Progressbar
p = ttk.Progressbar(root, orient = HORIZONTAL, length = 200, mode = 'determinate')
p.start()
p.pack()
root.mainloop()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม

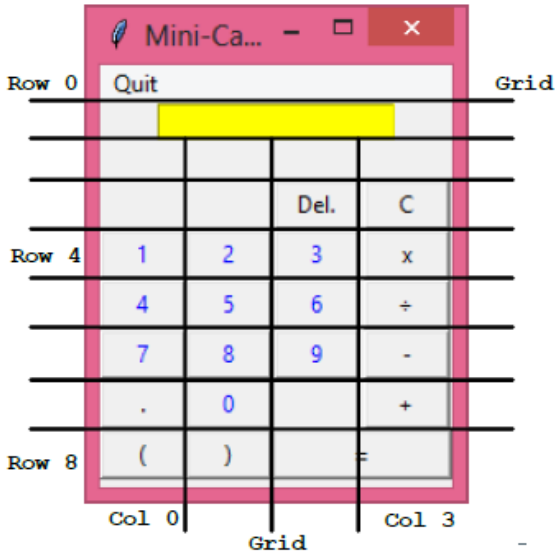


6. ตัวอย่างการประยุกต์ใช้งานไพธอน GUI

ในหัวข้อนี้ผู้เขียนจะนาเสนอรูปแบบการใช้งานไพธอน GUI ประยุกต์เข้ากับการใช้งานจริง ซึ่งมีเป้าหมายเพื่อต้องการให้ผู้อ่านเข้าใจวิธีการพัฒนาไพธอน GUI ในภาพรวม โดยการนำเอา Widgets ต่างๆ ที่อธิบายมาแล้วในตอนต้น ประกอบเข้าเป็นแอพพลิเคชันที่มีความซับซ้อนมากขึ้น เพื่อเป็นแนวทางในการพัฒนาโปรแกรมต่อไปในอนาคตได้ ซึ่งจะแสดงไว้ 1 ตัวอย่างคือ โปรแกรมเครื่องคิดเลข และเกม Tic-Tac-Toe ดังนี้

โปรแกรมเครื่องคิดเลขขนาดเล็ก (Mini-Calculator) มีขั้นตอนในการออกแบบและพัฒนาโปรแกรมดังนี้

1) สเก็ตช์ (Sketch) เครื่องคิดเลขลงบนกระดาษแบบคร่าวๆ ดังรูป



2) นำเข้าโมดูลในการสร้าง GUI (tkinter)

```
from tkinter import *
```

3) ประกาศตัวแปรชนิดโกลบอล (global) เพื่อให้ฟังชันต่างๆ สามารถเรียกใช้งานได้ทุกๆ ที่ในโปรแกรม

```
expression = ""
```

4) สร้างหน้าต่างหลัก (วินโดวส์หลัก) หรือ root window

```
root = drawMainWindow()
```

5) สร้างเมนู (Menubar)

```
menu = drawMenuBar()
```

6) สร้างหน้าจอแสดงผลลัพธ์ของเครื่องคิดเลข (Entry Widget)

```
display = drawDisplay()
```

7) สร้างปุ่มของเครื่องคิดเลขทั้งหมด

```
drawCalButton()
```

8) เขียนโปรแกรมดักจับและควบคุมเหตุการณ์ต่างๆ ที่เกิดขึ้นบนเครื่องคิดเลข

```
handleEvents()
```

9) กำหนดให้โปรแกรมวนลูบรับคำสั่งไปเรื่อยๆ จนกว่าจะยุติโปรแกรม

mainloop()

ตัวอย่างโปรแกรมชื่อ calculator.py

```
from tkinter import *


expression = ""


def drawMainWindow():
    root = Tk()
    root.title("Mini-Calculator")
    return root


def drawMenuBar(root):
    menu = Menu(root)
    menu.add_command(label="Quit", command=root.destroy)
    root.config(menu=menu)
    return menu


def drawDisplay(root):
    display = Entry(root, bg="yellow", fg="red")
    display.grid(row=1, column=0, columnspan=5)
    Label(root).grid(row=2, column=0)
    return display
```

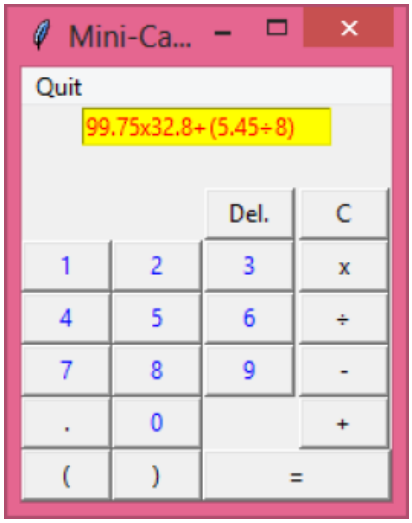
```
def collectExpression(keypress):
    global expression
    expression += keypress
    print(expression)
    return expression


def insertExpression(display, keypress):
    display.insert(END, keypress)


def clearDisplay(display, mode):
    global expression
    if mode == "DEL":
        expression = expression[0:len(display.get())-1]
        display.delete(len(display.get())-1, END)
        print(expression)
    elif mode == "DELALL":
        expression = ""
        display.delete(0, END)
        print(expression)
```

```
def handleEvents(display, keypress):
    if keypress == "C":
        clearDisplay(display, "DELALL")
    elif keypress == "DEL":
        clearDisplay(display, "DEL")
    elif keypress == ".":
        insertExpression(display, keypress)
        collectExpression(keypress)
    elif keypress == "(":
        insertExpression(display, keypress)
        collectExpression(keypress)
    elif keypress == ")":
        insertExpression(display, keypress)
        collectExpression(keypress)
    elif keypress == "+":
        insertExpression(display, keypress)
        collectExpression(keypress)
    elif keypress == "-":
        insertExpression(display, keypress)
        collectExpression(keypress)
    elif keypress == "x":
        insertExpression(display, keypress)
        collectExpression("*")
```

```
elif keypress == "÷":
    insertExpression(display, keypress)
    collectExpression("/")
elif keypress == "=":
    global expression
    try:
        result = eval(expression)
        print(result)
        clearDisplay(display, "DELALL")
        insertExpression(display, str(result))
    except:
        clearDisplay(display, "DELALL")
        insertExpression(display, "Error: Can't execute")
else:
    insertExpression(display, keypress)
    collectExpression(keypress)
```

```
def drawCalButton(root, display):
    Button(root, text="1", width=5, foreground="blue", command=lambda: handleEvents(display, "1")).grid(row=4, column=0)
    Button(root, text="2", width=5, foreground="blue", command=lambda: handleEvents(display, "2")).grid(row=4, column=1)
    Button(root, text="3", width=5, foreground="blue", command=lambda: handleEvents(display, "3")).grid(row=4, column=2)
    Button(root, text="4", width=5, foreground="blue", command=lambda: handleEvents(display, "4")).grid(row=5, column=0)
    Button(root, text="5", width=5, foreground="blue", command=lambda: handleEvents(display, "5")).grid(row=5, column=1)
    Button(root, text="6", width=5, foreground="blue", command=lambda: handleEvents(display, "6")).grid(row=5, column=2)
    Button(root, text="7", width=5, foreground="blue", command=lambda: handleEvents(display, "7")).grid(row=6, column=0)
    Button(root, text="8", width=5, foreground="blue", command=lambda: handleEvents(display, "8")).grid(row=6, column=1)
    Button(root, text="9", width=5, foreground="blue", command=lambda: handleEvents(display, "9")).grid(row=6, column=2)
    Button(root, text="0", width=5, foreground="blue", command=lambda: handleEvents(display, "0")).grid(row=7, column=1)
    Button(root, text=".", width=5, command=lambda: handleEvents(display, ".")).grid(row=7, column=0)
    Button(root, text="=", width=12, command=lambda: handleEvents(display, "=")).grid(row=8, column=2, columnspan=2)
    Button(root, text="(", width=5, command=lambda: handleEvents(display, "(")).grid(row=8, column=0)
    Button(root, text=")", width=5, command=lambda: handleEvents(display, ")")).grid(row=8, column=1)
    Button(root, text="x", width=5, command=lambda: handleEvents(display, "x")).grid(row=4, column=3)
    Button(root, text="÷", width=5, command=lambda: handleEvents(display, "÷")).grid(row=5, column=3)
    Button(root, text="-", width=5, command=lambda: handleEvents(display, "-")).grid(row=6, column=3)
    Button(root, text="+", width=5, command=lambda: handleEvents(display, "+")).grid(row=7, column=3)
    Button(root, text="C", width=5, command=lambda: handleEvents(display, "C")).grid(row=3, column=3)
    Button(root, text="Del.", width=5, command=lambda: handleEvents(display, "DEL")).grid(row=3, column=2)

def calculator():
    root = drawMainWindow()
    menu = drawMenuBar(root)
    display = drawDisplay(root)
    drawCalButton(root, display)
    mainloop()


if __name__ == '__main__':
    calculator()
```

ผลลัพธ์ที่ได้เมื่อสั่งรันโปรแกรม



จากตัวอย่างโปรแกรม บรรทัดที่ 1 เป็นการนำเข้าโมดูล tkinter เพื่อสร้าง GUI สำหรับเครื่องคิดเลข บรรทัดที่ 3 ประกาศตัวแปรชนิดโกลบอลชื่อ expression เพื่อใช้สำหรับเก็บนิพจน์คณิตศาสตร์ที่ผู้ใช้ป้อนให้กับเครื่องคิดเลข บรรทัดที่ 5 สร้างหน้าต่างหลัก โดยมี title คือ "Mini-Calculator" บรรทัดที่ 10 สร้างเมนูที่มีเมนูย่อยเพียงเมนูเดียวคือ Quit เพื่อใช้สำหรับออกจากโปรแกรม บรรทัดที่ 16 สร้าง Entry เพื่อรับข้อมูลจากผู้ใช้งานเพื่อนำมาประมวลผลภายในโปรแกรมเครื่องคิดเลข โดยวางลงบนกริดในตำแหน่งแถวที่ 1 คอลัมน์ที่ 0 และวางเลเบลเพื่อคั่นระหว่างปุ่มเครื่องคิดเลขกับ Entry โดยวางอยู่บนกริดในตำแหน่งแถวที่ 2 คอลัมน์ที่ 0

บรรทัดที่ 22 สร้างฟังชันชื่อ collectExpression() ทำหน้าที่เชื่อมต่อนิพจน์ของคณิตศาสตร์ที่ผู้ใช้งานป้อนเข้ามาด้วยปุ่ม บรรทัดที่ 28 สร้างฟังชันชื่อ insertExpression() ทำหน้าที่แสดงนิพจน์คณิตศาสตร์ออกทางจอภาพ บรรทัดที่ 31 สร้างฟังชันชื่อ clearDisplay() ทำหน้าที่ลบนิพจน์คณิตศาสตร์แบบครั้งเดียวทั้งหมด และลบแบบทีละตัว บรรทัดที่ 42 สร้างฟังชันชื่อ handleEvents() ทำหน้าที่ตอบสนองเมื่อผู้ใช้งานกดปุ่มต่างๆ บนเครื่องคิดเลข บรรทัดที่ 82 สร้างฟังชันชื่อ drawCalButton() ทำหน้าที่วาด Widgets ต่างๆ ลงบนหน้าต่างวินโดวส์ บรรทัดที่ 104 สร้างฟังชัน calculator() ทำหน้าที่ควบคุมการทำงานทั้งหมดของโปรแกรม โดยรวบรวมฟังชันหลักๆ เข้าไว้ด้วยกัน บรรทัดที่ 111 โปรแกรมตรวจสอบว่ามีการสร้างตัวแปร __main__ ไว้หรือไม่ (โดยปกติไพธอนจะสร้างตัวแปร __main__ ไว้แล้วอัตโนมัติ) ถ้าประกาศตัวแปรดังกล่าวไว้ในโปรแกรม จะส่งผลให้สามารถเรียกฟังชัน calculator() เข้ามาทำงานได้ เพราะเงื่อนไขใน if เป็นจริงนั่นเอง