

# สารบัญ

<b>1 แนะนำภาษาโปรแกรมไพธอน</b>	
1.1 ประวัติภาษาโปรแกรมไพธอน . . . . .	
1.2 การทำงานของภาษาโปรแกรมไพธอน . . . . .	
1.3 ข้อดีของภาษาโปรแกรมไพธอน . . . . .	
1.4 ความสามารถของภาษาโปรแกรมไพธอน . . . . .	
1.5 หน่วยงานที่เลือกใช้ภาษาโปรแกรมไพธอน . . . . .	
<b>2 แนะนำเครื่องมือพัฒนาโปรแกรมด้วยภาษาไพธอน</b>	
2.1 การติดตั้งตัวแปลงภาษาไพธอน . . . . .	
2.1.1 การติดตั้งบนระบบปฏิบัติการ Microsoft Windows 11 . . . . .	
2.1.2 การติดตั้งบนระบบปฏิบัติการ Ubuntu Linux 22.04 LTS . . . . .	
2.1.3 การติดตั้งบนระบบปฏิบัติการ macOS Ventura . . . . .	
2.1.4 การติดตั้งผ่าน pyenv . . . . .	
2.2 การใช้งานไพธอนสคริปต์ . . . . .	
2.3 รู้จักกับเครื่องมือ Jupyter Notebook . . . . .	
2.3.1 การติดตั้งและการเปิดใช้งาน Jupyter Notebook . . . . .	
2.4 เริ่มต้นใช้งาน Jupyter Notebook . . . . .	
2.4.1 แนะนำเมนูต่าง ๆ ก่อนเขียนคำสั่งโปรแกรม . . . . .	
2.4.2 การใช้เครื่องมือ Jupyter Notebook เขียนภาษาไพธอน . . . . .	
2.4.3 การขอความช่วยเหลือ . . . . .	
2.4.4 การ Download และ Upload ไฟล์ . . . . .	
2.4.5 ประมวลผลคำสั่งผ่านคีย์ลัด . . . . .	
2.5 การแสดงผลและการรับข้อมูล . . . . .	
2.5.1 การแสดงผลออกทางจอภาพ . . . . .	
2.5.2 การรับข้อมูล . . . . .	
2.6 การเขียนคำอธิบายโปรแกรม . . . . .	
<b>3 ตัวแปรและชนิดข้อมูลในภาษาโปรแกรมไพธอน</b>	
3.1 การตั้งชื่อตัวแปรในภาษาโปรแกรมไพธอน . . . . .	
3.2 การประกาศตัวแปร . . . . .	
3.3 ตัวแปรชนิดค่าคงที่ . . . . .	
3.4 ชนิดข้อมูลจำนวนเต็ม . . . . .	

3.5	ชนิดข้อมูลจำนวนทศนิยม . . . . .
3.6	ชนิดข้อมูลจำนวนเชิงซ้อน . . . . .
3.7	ชนิดข้อมูลตรรกะ . . . . .
3.8	ชนิดข้อมูลสายอักขระหรือสตริง . . . . .
3.9	การแปลงชนิดข้อมูล . . . . .
3.10	การกำหนดค่าให้กับตัวแปร . . . . .
3.11	การตรวจสอบชนิดข้อมูล . . . . .
3.12	นิพจน์ . . . . .
3.13	ตัวดำเนินการทางคณิตศาสตร์ . . . . .
3.14	ตัวดำเนินการเปรียบเทียบ . . . . .
3.15	ตัวดำเนินการกำหนดค่า . . . . .
3.16	ตัวดำเนินการตรรกศาสตร์ . . . . .
3.17	ตัวดำเนินการระดับบิต . . . . .
3.18	ตัวดำเนินการตรวจสอบสมาชิก . . . . .
3.19	ตัวดำเนินการเอกลักษณ์ . . . . .
3.20	ลำดับความสำคัญตัวดำเนินการ . . . . .
<b>4</b>	<b>ชนิดข้อมูลสตริง</b>
4.1	รู้จักกับชนิดข้อมูลสตริง . . . . .
4.2	การเขียนต่อสตริง . . . . .
4.3	การเข้าถึงตำแหน่งตัวอักขระในชนิดข้อมูลสตริง . . . . .
4.4	การควบคุมการแสดงผลออกทางจอภาพ . . . . .
4.4.1	การจัดรูปแบบแสดงผล . . . . .
4.4.2	การจัดรูปแบบแสดงผลด้วย f-string . . . . .
4.4.3	การจัดตำแหน่งแสดงผลข้อความ . . . . .
4.5	เมธอดที่ใช้กับชนิดข้อมูลลุ่มอักขระหรือสตริง . . . . .
4.5.1	เมธอดการเปลี่ยนลักษณะสตริง . . . . .
4.5.2	เมธอดการจัดตำแหน่งสตริง . . . . .
4.5.3	เมธอดสำหรับตรวจสอบสตริง . . . . .
4.5.4	เมธอดสำหรับการนับค่า ต้นหา และแก้ไขสตริง . . . . .
4.5.5	เมธอดสำหรับรวม แยก และตัดสตริง . . . . .
4.6	การดำเนินการกับชนิดข้อมูลอักขระหรือสตริง . . . . .
<b>5</b>	<b>โครงสร้างข้อมูลภาษาโปรแกรมไพธอน</b>
5.1	รู้จักกับข้อมูลชนิดลิสต์ . . . . .
5.1.1	การเข้าถึงตำแหน่งข้อมูลของข้อมูลชนิดลิสต์ . . . . .

5.1.2	เมอรอดและฟังก์ชันที่ใช้กับข้อมูลชนิดลิสต์ . . . . .
5.1.3	การดำเนินการกับข้อมูลชนิดลิสต์ . . . . .
5.2	รู้จักกับข้อมูลชนิดทูเพิล . . . . .
5.2.1	เมอรอดและฟังก์ชันที่ใช้กับข้อมูลชนิดทูเพิล . . . . .
5.2.2	การดำเนินการกับข้อมูลชนิดทูเพิล . . . . .
5.3	รู้จักกับข้อมูลชนิดดิกชันนารี . . . . .
5.3.1	เมอรอดและฟังก์ชันที่ใช้กับข้อมูลชนิดดิกชันนารี . . . . .
5.3.2	การดำเนินการกับข้อมูลชนิดดิกชันนารี . . . . .
5.4	รู้จักกับข้อมูลชนิดเซต . . . . .
5.4.1	เมอรอดและฟังก์ชันที่ใช้กับข้อมูลชนิดเซต . . . . .
5.4.2	การดำเนินการกับข้อมูลชนิดเซต . . . . .
5.5	การแปลงข้อมูลชนิดลิสต์ ทูเพิล ดิกชันนารี และเซต . . . . .
6	คำสั่งควบคุมทิศทางการทำงานโปรแกรม
6.1	รู้จักกับผังงาน . . . . .
6.2	การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบลำดับ . . . . .
6.3	การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบมีเงื่อนไข . . . . .
6.3.1	คำสั่ง <b>if</b> . . . . .
6.3.2	คำสั่ง <b>if ... else</b> . . . . .
6.3.3	คำสั่ง <b>if ... elif ... else</b> . . . . .
6.3.4	คำสั่ง <b>if</b> ซ้อนกัน . . . . .
6.4	คำสั่งโปรแกรมควบคุมการทำงานแบบทำซ้ำ . . . . .
6.4.1	คำสั่งวนซ้ำด้วย <b>while</b> . . . . .
6.4.2	การใช้คำสั่ง <b>else</b> ร่วมกับคำสั่งวนซ้ำ <b>while</b> . . . . .
6.4.3	คำสั่งวนซ้ำด้วย <b>for</b> . . . . .
6.4.4	การใช้คำสั่ง <b>else</b> ร่วมกับคำสั่งทำซ้ำ <b>for</b> . . . . .
6.5	การใช้คำสั่งทำซ้ำ <b>while</b> หรือ <b>for</b> ซ้อนกัน . . . . .
6.6	การควบคุมการทำซ้ำด้วยคำสั่ง <b>break</b> , <b>continue</b> และ <b>pass</b> . . . . .
7	ฟังก์ชัน
7.1	การสร้างฟังก์ชันขึ้นมาใช้งาน . . . . .
7.1.1	การสร้างฟังก์ชันที่ไม่มีการส่งค่าและรับค่า . . . . .
7.1.2	การสร้างฟังก์ชันที่มีการรับค่า แต่ไม่มีการส่งค่ากลับ . . . . .
7.1.3	การสร้างฟังก์ชันที่ไม่มีการรับค่า แต่มีการส่งค่ากลับ . . . . .
7.1.4	การสร้างฟังก์ชันที่มีการรับค่า พร้อมทั้งมีการส่งค่ากลับ . . . . .
7.2	อาร์กิวเมนต์และพารามิเตอร์ . . . . .

7.3	รูปแบบการส่งค่าอาร์กิวเมนต์ให้กับค่าพารามิเตอร์ . . . . .
7.3.1	การส่งค่าอาร์กิวเมนต์แบบ Required arguments . . . . .
7.3.2	การส่งอาร์กิวเมนต์แบบ Keyword arguments . . . . .
7.3.3	การส่งอาร์กิวเมนต์แบบ Default arguments . . . . .
7.3.4	การส่งอาร์กิวเมนต์แบบ Variable-length arguments . . . . .
7.4	การสร้างฟังก์ชันด้วยคำสั่ง <b>lambda</b> . . . . .
7.5	ขอบเขตการเรียกใช้งานตัวแปร . . . . .
7.6	ฟังก์ชัน Built-in ภาษาไพธอน . . . . .
<b>8</b>	<b>แนะนำการเขียนโปรแกรมเชิงวัตถุ</b>
8.1	ทำไมต้องเขียนโปรแกรมแบบ OOP . . . . .
8.2	เริ่มต้นเขียนโปรแกรมแบบ OOP ด้วยภาษาไพธอน . . . . .
8.2.1	การสร้างคลาส . . . . .
8.2.2	การสร้างเมธอด . . . . .
8.2.3	การสร้างจินสแตนซ์ . . . . .
8.3	การสร้างแอ็ตทริบิวต์ และคอนสตรัคเตอร์ . . . . .
8.4	การสืบทอด . . . . .
8.5	เมธอด <b>__str__()</b> และ <b>__repr__()</b> . . . . .
8.6	การไอเวอร์โหลดตัวดำเนินการ . . . . .
8.7	การไอเวอร์โหลดตึงเมธอด . . . . .
8.8	การห่อหุ้ม/ซ่อนข้อมูล . . . . .
<b>9</b>	<b>การจัดการข้อผิดพลาด</b>
9.1	ประเภทของ Exception ในภาษาไพธอน . . . . .
9.2	การตรวจจับข้อผิดพลาดประเภท Exception . . . . .
9.2.1	การใช้คำสั่ง <b>try...except</b> ตรวจจับประเภท Exception . . . . .
9.2.2	การใช้คำสั่ง <b>try...except</b> ตรวจจับ Exception หลายตัว . . . . .
9.2.3	การใช้คำสั่ง <b>try...except...else</b> ตรวจจับประเภท Exception . . . . .
9.2.4	การใช้คำสั่ง <b>try...except...finally</b> ตรวจจับประเภท Exception . . . . .
9.2.5	การใช้คำสั่ง <b>try...except</b> ซ้อนกันเพื่อตรวจจับประเภท Exception . . . . .
9.3	การสร้างข้อความแจ้งเตือนข้อผิดพลาดด้วยคำสั่ง <b>raise exception</b> . . . . .
9.4	การสร้างและเรียกใช้งาน Exception ที่สร้างขึ้นเอง . . . . .
9.5	การยืนยันความถูกต้อง . . . . .
<b>10</b>	<b>การจัดการกับไฟล์</b>
10.1	โหมดของไฟล์ . . . . .
10.1.1	โหมดไฟล์ข้อความ (Text File Mode) . . . . .

10.1.2	โหมดไฟล์ในนารี (Binary File Mode) . . . . .
10.2	เมธอดสำหรับจัดการกับไฟล์ . . . . .
10.3	การอ่านและเขียนไฟล์ในนารี (Read / Write Binary File) . . . . .
10.4	เมธอดสำหรับการจัดการไดเรกทอรี (Directory Methods) . . . . .
10.5	การตรวจสอบการจัดการไฟล์และไดเรกทอรี . . . . .
10.6	การอ่านและเขียนไฟล์ด้วยคำสั่ง with . . . . .
<b>11</b>	<b>การใช้งานโมดูลและแพ็คเกจ</b>
11.1	การตรวจสอบโมดูลในภาษาไพธอน . . . . .
11.2	รูปแบบการเรียกใช้งานโมดูลและฟังก์ชัน . . . . .
11.3	การสร้างโมดูลขึ้นใช้งาน . . . . .
11.4	การสร้างแพ็คเกจ . . . . .
11.5	โมดูลสำเร็จสู่ปั้นภาษาไพธอน . . . . .
11.5.1	โมดูลคณิตศาสตร์ . . . . .
11.5.2	โมดูลแสดงผลวันที่และเวลา . . . . .



# บทที่ 1

## แนะนำภาษาโปรแกรมไพธอน

ในปัจจุบันภาษาโปรแกรมไพธอนเป็นภาษาโปรแกรมที่ได้รับความนิยมเป็นอย่างมากโดยเฉพาะในด้านวิทยาศาสตร์ วิศวกรรมศาสตร์ วิทยาการข้อมูล และปัญญาประดิษฐ์ ในบทนี้เราจะแนะนำภาษาโปรแกรมไพธอนให้ผู้อ่านได้รู้ตั้งแต่ประวัติ ขั้นตอนการทำงาน ตลอดจนข้อดีและความสามารถของภาษาโปรแกรมไพธอน

### 1.1 ประวัติภาษาโปรแกรมไพธอน



รูป 1.1: Guido van Rossum

ภาษาโปรแกรมไพธอน (Python Programming Language) ถูกพัฒนาขึ้นโดย Guido van Rossum ชาวเนเธอร์แลนด์ ซึ่งสำเร็จการศึกษาระดับปริญญาโทสาขาวิศวกรรมศาสตร์และวิทยาการคอมพิวเตอร์จาก University of Amsterdam ประเทศเนเธอร์แลนด์ เขาเคยได้เข้าร่วมทำงานกับ Centrum Wiskunde & Informatica (CWI) ประเทศเนเธอร์แลนด์, National Institute of Standards and Technology (NIST) และ Corporation for National Research Initiatives (CNRI) ประเทศสหรัฐอเมริกา และได้ร่วมงานกับบริษัท Google ตั้งแต่ปี 2005 จนถึงปี 2012 และย้ายไปทำงานที่ Dropbox ในปี 2013 จนกระทั่งเกษียณตัวเองในเดือนตุลาคมปี 2019 และได้กลับเข้ามาเริ่มงานอีกครั้งกับ Microsoft เมื่อปลายปี 2020 โดยมีเป้าหมายหลักคือการพัฒนาให้ Python

ทำงานได้เร็วขึ้นอย่างน้อย 5 เท่าภายในระยะเวลา 4 ปี ทั้งนี้ที่มาของชื่อภาษาโปรแกรมไพธอน มาจากตอนที่ Van Rossum กำลังเริ่มพัฒนาภาษาโปรแกรมไพธอน เขายังได้อ่านสคริปต์ของการแสดง “Monty Python’s Flying Circus” ซึ่งเป็นซีรีส์ตลกของช่อง BBC ในยุค 1970s ทำให้เขาได้คิดว่า เขายังต้องการชื่อที่สั้น มีเอกลักษณ์ และค่อนข้างลึกลับน่าค้นหา เขายังคงได้ตัดสินใจใช้ชื่อ Python สำหรับภาษาโปรแกรมที่เขา正在กำลังพัฒนา

ภาษาโปรแกรมไพธอนเป็นภาษาระดับสูงที่ได้รับการพัฒนาจากการผสมผสานความหลากหลายของภาษาอื่น ๆ เช่น ABC, Modula-3, Icon, Perl, Lisp, Smalltalk เป็นต้น นอกจากนี้ยังมีความสามารถในการจัดการหน่วยความจำแบบอัตโนมัติ รวมไปถึงการจัดการในเรื่องของตัวแปรที่สร้างขึ้นมาใช้งานโดยไม่ต้องกำหนดชนิดข้อมูล

ภาษาโปรแกรมไพธอนได้รับการเผยแพร่สู่สาธารณะในปี 1991 ด้วยเวอร์ชัน 0.9.0 ต่อมาในปี 1994 ได้ออกเวอร์ชัน 1.0 ในปี 2000 ได้ออกเวอร์ชัน 2.0 ในปี 2008 ได้ออกเวอร์ชัน 3.0 ในปัจจุบันเวอร์ชันภาษาโปรแกรมไพธอนคือ 3.11.0 เราสามารถดาวน์โหลดตัวซอฟต์แวร์ติดตั้งภาษาโปรแกรมไพธอนได้ที่ <http://www.python.org> โดยไม่มีค่าใช้จ่าย (เป็นซอฟต์แวร์ประเภท Open Source) สำหรับภาษาโปรแกรมไพธอนเวอร์ชัน 2 ได้มีการประกาศหยุดการพัฒนาแล้ว โดยตั้งแต่ 1 มกราคม 2020 จะใช้ Python 3 เป็นมาตรฐานเพียงอย่างเดียวเท่านั้น ทั้งนี้เวอร์ชันของภาษาโปรแกรมไพธอนและวันเดือนปีที่ถูกปล่อยออกมาให้ใช้งานแสดงดังตาราง 1.1

จุดเด่นอีกอย่างหนึ่งของภาษาโปรแกรมไพธอนคือมีชุมชน (Community) ที่มีนักพัฒนาเข้าร่วมพัฒนาภาษาโปรแกรมไพธอนอย่างมากมาย และมีไลบรารี (Library) ให้ดาวน์โหลดเลือกใช้งานโดยไม่เสียค่าใช้จ่ายจำนวนมากไม่ใช่จะเป็นไลบรารีที่เกี่ยวข้องการประมวลผลข้อมูลทางด้านคณิตศาสตร์และวิทยาศาสตร์, ไลบรารีประมวลผลข้อมูลภาพ (Image Processing), ไลบรารีประมวลผลภาพสามมิติ (3D Graphics Rendering) นอกจากนี้ยังมีผู้สนับสนุนหลักรายใหญ่อย่างเช่น Microsoft, Google, eBay ซึ่งจะทำให่อนาคตของภาษาโปรแกรมไพธอนมีความสดใสและมีแนวโน้มที่จะได้รับความนิยมเพิ่มมากขึ้น

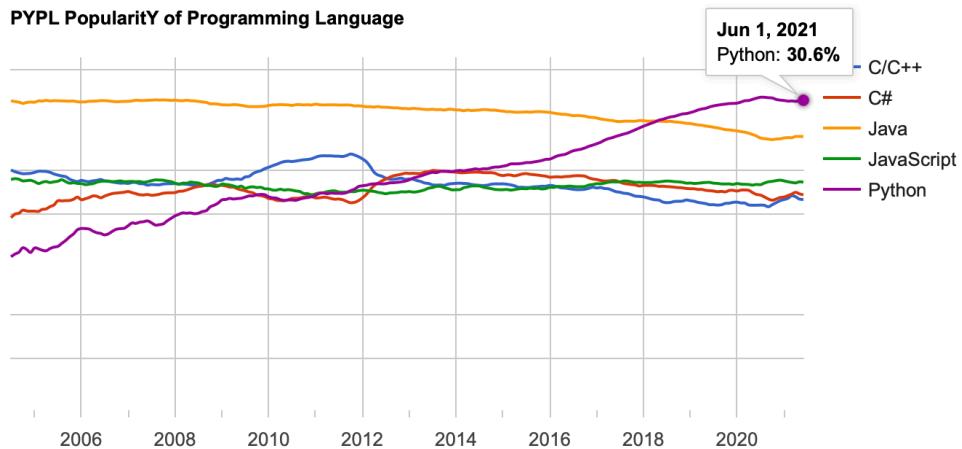
ในปี 2021 ภาษาโปรแกรมไพธอนได้ก้าวขึ้นมาเป็นภาษาโปรแกรมยอดนิยมอันดับต้น ๆ เนื่องจากมีโครงสร้างไม่ซับซ้อน ทำให้พัฒนาโปรแกรมได้อย่างรวดเร็ว อีกทั้งยังมีอาชีพใหม่เกิดขึ้นที่เป็นรุ่นจักกันคือนักวิเคราะห์ข้อมูล (Data Analyst) และนักวิทยาศาสตร์ข้อมูล (Data Scientist) ซึ่งส่วนใหญ่เลือกที่จะใช้ภาษาโปรแกรมไพธอนประมวลผลข้อมูล เพราะมีไลบรารี Open Source เป็นจำนวนมาก เช่น ไลบรารี Tensorflow ที่พัฒนาโดย Google, ไลบรารี Caffe ที่พัฒนาโดย Berkeley Artificial Intelligence Research, ไลบรารี Theano ที่พัฒนา University of Montreal เป็นต้น

จากดัชนีความนิยมของภาษาโปรแกรม<sup>1</sup> ที่คำนวณมาจากจำนวนบทความสอนการใช้งาน (Tutorial) ที่ถูกค้นหาผ่าน Google จากผู้ใช้งานทั่วโลก ในเดือนมิถุนายน 2021 พ布ว่า ภาษาโปรแกรมไพธอน ได้ก้าวขึ้นมาเป็นอันดับ 1 ของภาษาโปรแกรมที่ได้รับความนิยมมากที่สุด มีส่วนแบ่งมากถึง 30.6% ซึ่งแซงหน้าภาษาโปรแกรมอย่างเช่น Java, Javascript, C# และ C/C++ แสดงดังรูป 1.2

<sup>1</sup>The PYPL PopularitY of Programming Language Index, ดูเพิ่มเติม <https://pypl.github.io/PYPL.html>

เวอร์ชัน		วันเปิดตัว
Python 1.0	Python 1.0	January 26, 1994
	Python 1.5	December 31, 1997
	Python 1.6	September 4, 2000
Python 2.0	Python 2.0	October 16, 2000
	Python 2.1	April 17, 2001
	Python 2.2	December 21, 2001
	Python 2.3	July 29, 2003
	Python 2.4	November 30, 2004
	Python 2.5	September 19, 2006
	Python 2.6	October 1, 2008
	Python 2.7	July 3, 2010
Python 3.0	Python 3.0	December 3, 2008
	Python 3.1	June 27, 2009
	Python 3.2	February 20, 2011
	Python 3.3	September 29, 2012
	Python 3.4	March 16, 2014
	Python 3.5	September 13, 2015
	Python 3.6	December 23, 2015
	Python 3.7	June 27, 2018
	Python 3.8	October 14, 2019
	Python 3.9	October 5, 2020
	Python 3.10	October 4, 2021
	Python 3.11	October 24, 2022

ตาราง 1.1: เวอร์ชันของภาษาโปรแกรม Python



รูป 1.2: กราฟเปรียบเทียบดัชนีความนิยมของภาษาโปรแกรม (PYPL Index) เมื่อเดือนมิถุนายน 2021 (Logarithmic Scale), ที่มา <https://pypl.github.io/PYPL.html>

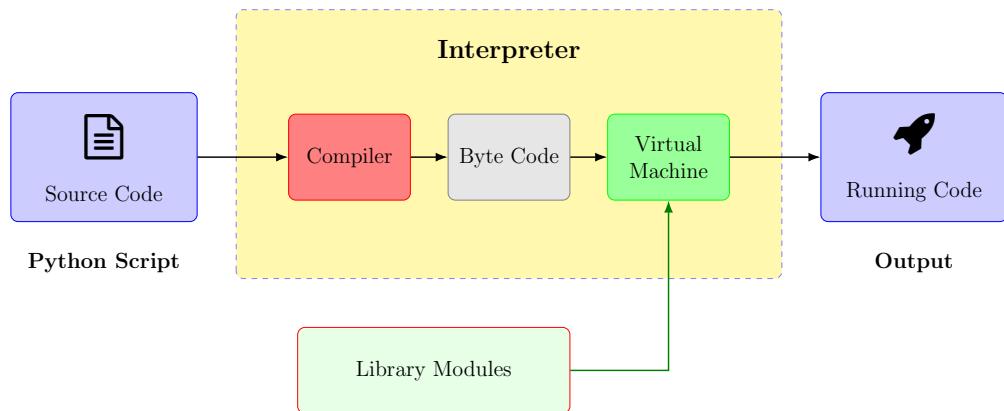
## 1.2 การทำงานของภาษาโปรแกรมไพธอน

ก่อนที่เราจะเริ่มลงมือเขียนโปรแกรมด้วยภาษาโปรแกรมไพธอน เราควรรู้ว่าขั้นตอนการทำงานในภาษาโปรแกรมไพธอนมีอะไรบ้างเพื่อเป็นพื้นฐานและความเข้าใจที่ดีในการพัฒนาโปรแกรม โดยในรูป 1.3 แสดงขั้นตอนการทำงานของภาษาโปรแกรมไพธอนโดยสรุป

โดยหลักการพื้นฐาน การพัฒนาโปรแกรมจะเริ่มต้นจากการเขียนไฟล์รหัสต้น (Source Code) ที่บรรจุคำสั่งภาษาโปรแกรมไพธอนเป็น Plain Text ซึ่งมีนามสกุลเป็น .py โดยที่ต่อไปนี้เราจะเรียกไฟล์พกนี้ว่า “ไพธอนสคริปต์” (Python Script) และเราจะใช้ตัวแปลงภาษา (Interpreter) ในการประมวล ไพธอนสคริปต์ให้ทำงานบนระบบปฏิบัติการ

ในขั้นตอนการแปลงภาษาเนี้ยจะเริ่มจากการประมวล (Compilation) โดยที่คอมไพล์เลอร์ (Compiler) ทำการประมวลและแปล (Translator) คำสั่งในไพธอนสคริปต์เป็นคำสั่งในรูป Byte Code ที่มีนามสกุลเป็น .pyc ซึ่งสามารถทำงานได้โดยไม่ขึ้นอยู่กับระบบปฏิบัติการผ่านเครื่องจักรเสมือนจริง (Virtual Machine) ที่เรียกว่า Python Virtual Machine: PVM ทั้งนี้ไฟล์ Byte Code ที่ได้จากการแปลจะมีคำสั่งใกล้เคียงกับภาษาเครื่องมากขึ้น แต่ไม่ใช่ภาษาเครื่องเหมือนการประมวลผ่าน C/C+ + Compilers

ภาษาโปรแกรมไพธอนใช้ตัวแปลงภาษาแบบ Interpreter ซึ่งเกือบจะการแปลงคำสั่ง ใน Source Code ที่ละเอียดจากบรรทัดลงมาบรรทัดล่าง และจะมีการแจ้งเตือนเมื่อตรวจสอบบรรทัดที่เขียนผิดหลักไวยากรณ์ภาษาโปรแกรมไพธอน ซึ่งผู้พัฒนาจำเป็นต้องแก้ไขให้ถูกต้องจนกว่าตัวแปลงภาษาจะตรวจไม่พบข้อผิดพลาดใน Source Code



รูป 1.3: ขั้นตอนการทำงานภาษาโปรแกรมไพธอน

ด้วยการออกแบบเช่นนี้ทำให้โปรแกรมที่ถูกพัฒนาด้วยภาษาโปรแกรมไพธอนสามารถทำงานได้ทั้งบนระบบปฏิบัติการไมโครซอฟท์วินโดวส์ (Microsoft Windows), ลีนุกซ์ (Linux), แมคโออีส (macOS) และยังรวมไปถึงระบบปฏิบัติการสมาร์ทโฟนอย่างเช่น Android และ iOS อีกด้วย

### 1.3 ข้อดีของภาษาโปรแกรมไพธอน

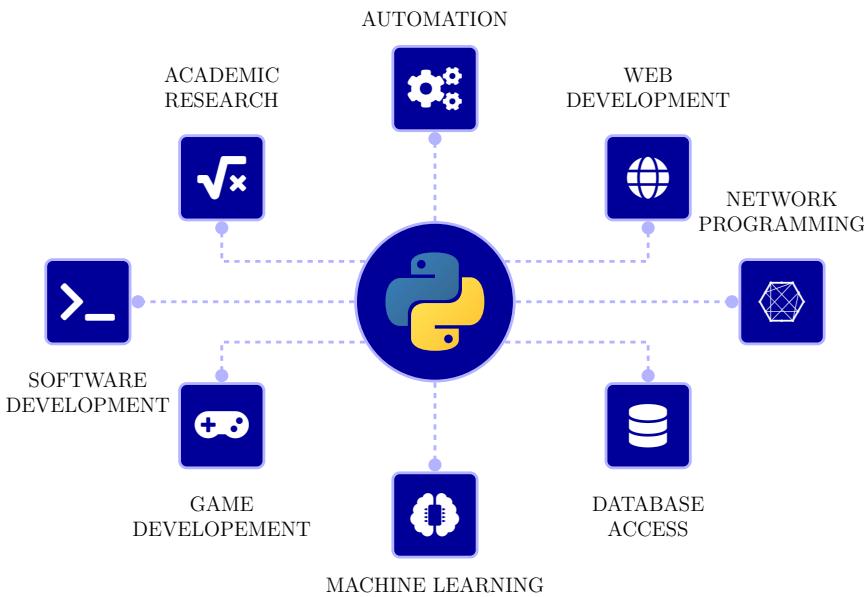
ภาษาโปรแกรมไพธอนได้ถูกนำมาพัฒนาโปรแกรมในหลาย ๆ ด้าน เนื่องจากมีข้อดีดังนี้

1. มีโครงสร้างการเขียนคำสั่งโปรแกรมที่ง่าย ไม่ซับซ้อน เหมาะสำหรับผู้ที่ต้องการเริ่มเขียนโปรแกรม
2. ภาษาโปรแกรมไพธอนไม่จำเป็นต้องกำหนดชนิดข้อมูลล่วงหน้าให้กับตัวแปรที่สร้างขึ้นมาเก็บค่า ข้อมูลโดยจะรู้ได้เองจากค่าที่กำหนดให้กับตัวแปรนั้น
3. คำสั่งโปรแกรมของภาษาโปรแกรมไพธอนสามารถประมวลผลบนระบบปฏิบัติการอื่นได้ เช่น พัฒนาโปรแกรมบน Microsoft Windows และนำไปประมวลผลบนระบบปฏิบัติการ Linux เป็นต้น
4. ภาษาโปรแกรมไพธอนมีเครื่องมือ IDE (Integrated Development Environment tool) รองรับการพัฒนาโปรแกรมเป็นจำนวนมาก เช่น IDLE, PyCharm, Spyder, Kite และ Thonny เป็นต้น นอกจากนี้ยังมี Jupyter Notebook (ทำงานผ่านเว็บбраузอร์) เป็นเครื่องมือที่ได้รับความนิยมอย่างมากในการทำงานด้าน Data Science
5. ภาษาโปรแกรมไพธอนมีไลบรารีพร้อมให้เรียกใช้งานจำนวนมากทั้งทางด้านคณิตศาสตร์และวิทยาศาสตร์ รวมไปถึงวิศวกรรมศาสตร์ และยังมีไลบรารีอื่น ๆ ที่มีกลุ่มนักพัฒนาร่วมกันสร้างขึ้นมาไว้ให้ติดตั้งเพิ่มเติมในภายหลังโดยไม่เสียค่าใช้จ่ายมาก เช่น ไลบรารีการประมวลผลภาพ (Image Processing), ไลบรารีทางด้านปัญญาประดิษฐ์ (Artificial Intelligent: AI), ไลบรารีการเรียนรู้ด้วยเครื่อง (Machine Learning) เป็นต้น
6. รองรับการพัฒนาแอพพลิเคชั่นแบบโครงสร้าง (Structure Programming) และสนับสนุนการพัฒนาแอพพลิเคชั่นแบบเชิงวัตถุ (Object-Oriented Programming; OOP)
7. ภาษาโปรแกรมไพธอนสามารถพัฒนาโปรแกรมร่วมกับภาษาอื่น ๆ ได้ เช่น พัฒนาร่วมกับภาษา Java ผ่านไลบรารี Jython พัฒนาร่วมกับภาษา C ผ่านไลบรารี CPython เป็นต้น

## 1.4 ความสามารถของภาษาโปรแกรมไฟรอน

ภาษาโปรแกรมไฟรอนมาพัฒนาเป็นโปรแกรมให้ทำงานด้านต่าง ๆ ไม่ว่าจะเป็นงานทางด้านเครือข่าย (Network), โปรแกรมบนมือถือ (Mobile Application), IOT (Internet of Things) การพัฒนาโปรแกรมบนเว็บ (Web Application) นอกจากนี้ภาษาโปรแกรมไฟรอนมีความเป็นภาษาการ (Glue Language) คือ สามารถเรียกใช้งานภาษาอื่น ๆ เพื่อประสานประมวลผลข้อมูลร่วมกันได้ และยังมีข้อดี และความสามารถที่เด่น ๆ อาทิเช่น

1. มีความสามารถเชื่อมต่อ กับระบบฐานข้อมูลได้อย่างหลากหลาย ทั้งระบบฐานข้อมูลแบบ SQL อย่าง MySQL, MariaDB, PostgreSQL หรือระบบฐานข้อมูลแบบ NoSQL อย่างเช่น MongoDB, Apache CouchDB เป็นต้น
2. ทำงานได้บนทุกระบบปฏิบัติการทั้ง Windows, Linux, macOS, Android, iOS
3. รองรับการพัฒนาโปรแกรมแบบกราฟิกที่เรียกว่า Graphic User Interface (GUI) โดยเรียกใช้ งานผ่านไลบรารี Tkinter ซึ่งเป็นไลบรารีมาตรฐานที่มาพร้อมกับภาษาโปรแกรมไฟรอน หรือจะ ติดตั้งไลบรารีเพิ่มเติม เช่น PyQt5, WxPython เป็นต้น
4. รองรับการพัฒนาแอปพลิเคชันผ่านโทรศัพท์มือถือที่สนับสนุนการทำงานทั้งบนระบบปฏิบัติการ Android และ iOS โดยใช้ไลบรารี Kivy ซึ่งเป็นไลบรารีที่สามารถนำมายังงานได้ฟรี
5. รองรับการพัฒนาเว็บแอปพลิเคชันที่มีไลบรารีที่ได้รับความนิยมอย่างเช่น Django, Flask, TurboGears, Web2py เป็นต้น
6. มีไลบรารีที่ใช้สำหรับพัฒนา Game อย่างเช่น PyGame, Pykyra, Pyglet เป็นต้น นอกจากนี้ยัง มีไลบรารีอื่น ๆ ที่นำมายังการพัฒนาภาพ 2D, 3D และ Animation อีกจำนวนมาก เช่น Pymunk, PyOpenGL, PySoy เป็นต้น
7. รองรับการพัฒนาอุปกรณ์ที่สามารถเชื่อมต่อเข้ากับระบบอินเตอร์เน็ต (Internet of Things: IoTs) ที่นำไปประยุกต์พัฒนาในด้านต่าง ๆ เช่น Smart Home, Smart Farm เป็นต้น



รูป 1.4: การนำภาษาโปรแกรมไปประยุกต์ใช้งาน

## 1.5 หน่วยงานที่เลือกใช้ภาษาโปรแกรมไพธอน

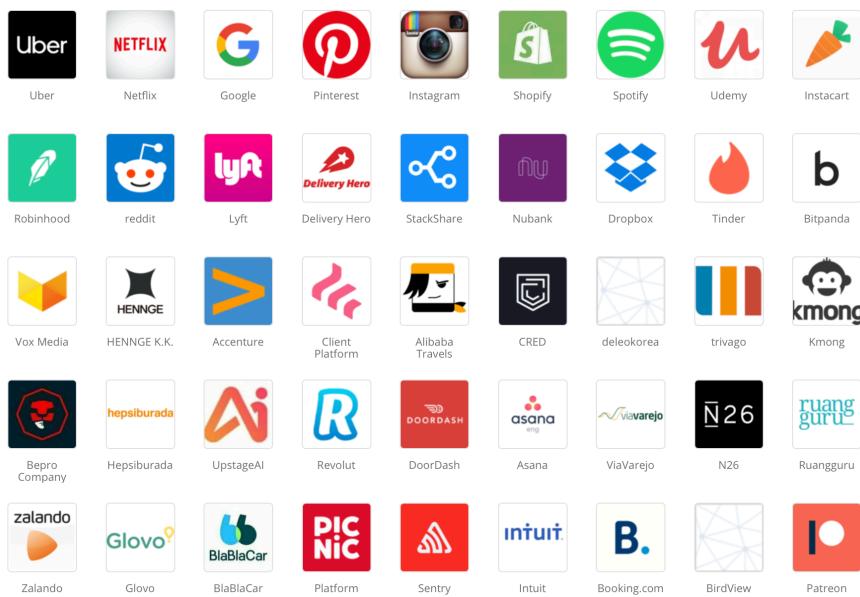
จากความสามารถของภาษาโปรแกรมไพธอนที่ได้กล่าวมาในหัวข้อก่อนหน้านี้ ทำให้หลายหน่วยงานทั่วโลกได้เลือกใช้ภาษาโปรแกรมไพธอนในการพัฒนาซอฟต์แวร์และนำเสนอผลิตภัณฑ์สู่ท้องตลาด ในหัวข้อนี้เราได้รวบรวมตัวอย่างหน่วยงาน และการนำภาษาโปรแกรมไพธอนไปใช้ ดังนี้

- Dropbox ผู้ให้บริการการซิงก์ (Syncronization) และฝาก (Hosting) ไฟล์ข้อมูลแบบออนไลน์ เคยได้ทำงานร่วมกับ Guido Van Rossum และได้ใช้ไพธอนในการพัฒนา Dropbox Desktop Client ที่สามารถรองรับผู้ใช้งานได้มากกว่า 400 ล้านคน
- Google บริษัทมหาชนสัญชาติอเมริกันผู้ให้บริการ Search Engine อีเมล แผนที่ออนไลน์ ซอฟต์แวร์จัดการด้านสำนักงาน เครื่องข่ายออนไลน์ และวีดีโอออนไลน์ (YouTube) ได้เลือกใช้ภาษาโปรแกรมไพธอนในการพัฒนาระบบภายในจำนวนมาก เช่น อัลกอริズึมหลักในการสืบค้นของ Search Engine ได้ถูกพัฒนาโดยใช้ภาษาโปรแกรมไพธอน และ C++, ระบบการแสดงผลและการจัดการวีดีโอของ YouTube ได้ถูกพัฒนาโดยใช้ภาษาโปรแกรมไพธอน, เว็บไซต์ของ Google Developers เองก็เลือกภาษาโปรแกรมไพธอนในการพัฒนา
- Zope Corporation ได้ใช้ภาษาโปรแกรมไพธอนในการพัฒนาเว็บแอพลิเคชันเซอร์เวอร์ (Web Application Server) แบบ Open Source ที่ทรงประสิทธิภาพ

- Reddit เว็บไซต์รวบรวมข่าวสาร จัดลำดับข้อมูลออนไลน์ และเว็บสนทน่า ได้เลือกใช้โปรแกรมภาษาไพธอนในการพัฒนา เนื่องจากผู้ร่วมก่อตั้งเห็นถึงข้อดีของภาษาที่สามารถติดตามแก้ไขและพัฒนาเว็บไซต์ได้อย่างราบรื่น
- National Aeronautics and Space Administration: NASA หน่วยงานรัฐบาลของสหรัฐอเมริกา ซึ่งรับผิดชอบโครงการอวกาศ การบุกเบิกอนาคตแห่งการสำรวจอวกาศ การค้นพบทางวิทยาศาสตร์ และงานวิจัยทางการบินและอวกาศ ซึ่งแม้จะไม่เคยประกาศอย่างเป็นทางการว่าทาง NASA ได้เลือกใช้ภาษาโปรแกรมใดในการพัฒนาซอฟต์แวร์ของตนเอง แต่ก็มีหลักฐานหลายชิ้นชี้ให้เห็นว่า ภาษาโปรแกรมไพธอนได้เข้ามา มีบทบาทสำคัญในหลายโครงการสำคัญ เช่น บริษัท United Space Alliance: USA ผู้รับจ้างหลักในงานดูแลภาระส่วนของ NASA ได้เลือกใช้ภาษาโปรแกรมไพธอนในการพัฒนาระบบการทำงานอัตโนมัติ (Workflow Automation System: WAS) สำหรับ NASA ด้วยเหตุผลในเรื่องความเร็ว ราคาถูก และทำงานได้อย่างสมบูรณ์ หรือจะเป็นในส่วนของโครงการอวกาศขนาดใหญ่ในอดีตของ NASA ที่ได้เปิดเผยแพร่สาธารณะกว่า 400 โครงการ ทั้งในด้านการการวิจัยสำรวจ ดวงดาว, ดาวเคราะห์, ขั้นบรรยากาศ, การบิน, เชนเชอร์ ระยะไกล และสารสนเทศภูมิศาสตร์<sup>2</sup> ที่ได้ถูกพัฒนาด้วยภาษาโปรแกรมภาษาไพธอน
- Facebook บริษัทมหาชนสัญชาติอเมริกันผู้ให้บริการเครือข่ายสังคม (Social Network) ที่มีผู้ใช้บริการมากที่สุดในปัจจุบัน ได้เลือกใช้ภาษาโปรแกรมไพธอนในการจัดการระบบ硬件ด้วยโครงสร้าง (Infrastructure Management) ช่วยให้วิศวกรดูแลรักษาระบบได้อย่างราบรื่นและมีประสิทธิภาพ และยังได้พัฒนาโครงการโอเพนซอร์สจำนวนมากด้วยภาษาโปรแกรมไพธอน เช่น Facebook Ads API และ Python Async IRCbot Framework เป็นต้น
- Netflix บริษัทมหาชนสัญชาติอเมริกันที่ทำธุรกิจการผลิตสื่อ และให้บริการสื่อที่นำเสนอโดยตรงกับผู้รับบริการผ่านทางอินเทอร์เน็ตทั่วโลก เปิดโอกาสให้นักพัฒนาของตนได้เลือกใช้ภาษาโปรแกรมได้อย่างอิสระ และด้วยความง่ายในการติดต่อและ Source Code และจำนวนไลบรารีสนับสนุนการแก้ปัญหาจำนวนมาก ทำให้ภาษาโปรแกรมไพธอนได้ถูกเหล่านักพัฒนาเลือกใช้เป็นภาษาโปรแกรมอันดับต้น ๆ ในการพัฒนาหลายโครงการของ Netflix เช่น Central Alert Gateway (CAG), Chaos Gorilla, Security Monkey, Chronos และโดยเฉพาะอย่างยิ่ง กับโครงการ System of Regional Failover ที่สามารถทำให้เวลาตอบรับ (Response Time) ในสถานการณ์ไฟฟ้าขัดข้องหรืออุปกรณ์เสียหาย ลดลงร้อยละ 85 โดยไม่ต้องเพิ่มค่าใช้จ่ายใด ๆ ก็ได้ถูกพัฒนาด้วยภาษาโปรแกรมไพธอนทั้งหมด
- Instagram ผู้ให้บริการเครือข่ายสังคมด้านการแลกเปลี่ยนรูปภาพและวีดีโอ ซึ่งระบบทั้งหมดนั้นถูกพัฒนาบน Django หนึ่งใน Web Framework เขียนโดยภาษาโปรแกรมไพธอน สามารถรองรับผู้ใช้งานได้มากกว่า 400 ล้านคนต่อวัน นับเป็นข้อพิสูจน์สำคัญที่ทำให้เห็นว่าภาษาโปรแกรม

---

<sup>2</sup>NASA Open Source Software Projects, ดูเพิ่มเติม <https://code.nasa.gov/>



รูป 1.5: ตัวอย่างหน่วยงานที่เลือกใช้ภาษาโปรแกรมไพธอน

รวมไพธอนไม่เพียงแต่จะเขี่ยมยอดในการประยุกต์ใช้งานได้หลากหลาย แต่มันยังสามารถถูกขยายให้รองรับผู้ใช้งานจำนวนมหาศาลได้อย่างยอดเยี่ยม

แม้แต่ในประเทศไทยในขณะนี้ภาษาโปรแกรมไพธอนได้ถูกบรรจุให้เป็นภาษาที่ใช้ทำการเรียนการสอน การเขียนโปรแกรมในหลักสูตรระดับมัธยมศึกษา รวมไปถึงในระดับมหาวิทยาลัยก็ดำเนินการใช้ทำการเรียนการสอนเช่นกัน

## สรุปก่อนจบบท

ในบทนี้เราได้แนะนำภาษาโปรแกรมไพธอน ซึ่งในปัจจุบันเป็นภาษาโปรแกรมที่กำลังได้รับความนิยมอย่างมาก โดยเฉพาะอย่างยิ่งในสาขาวิชาการข้อมูล ภาษาโปรแกรมไพธอนยังมีความสามารถอีกหลาย ๆ ด้านไม่ว่าจะเป็นการพัฒนาเว็บแอพพลิเคชัน การพัฒนาโปรแกรมภาษาโปรแกรมไพธอนบนอุปกรณ์ต่าง ๆ ที่มีการเชื่อมต่ออินเตอร์เน็ต นอกจากนี้ยังมีไลบรารีอีกมากมายที่ผู้พัฒนาทั่วทุกมุมโลกเตรียมไว้ให้ใช้งานโดยไม่มีค่าใช้จ่าย อีกทั้งยังมีเครื่องมือที่หลากหลายให้เลือกนำมาใช้เขียนคำสั่งโปรแกรมภาษาโปรแกรมไพธอนได้อย่างสะดวก

## แบบฝึกหัด

1. จงบอกคุณลักษณะของภาษา RATE ที่ต้องการ และยกตัวอย่างภาษาอื่น ๆ นอกจาก Python
2. จงบอกข้อแตกต่างระหว่างตัวแปรภาษาแบบอินเตอร์พรีเซอร์กับคอมไพร์เซอร์
3. จงยกตัวอย่างไลบรารีของภาษาโปรแกรม Python มา 5 อย่าง และอธิบายว่าไลบรารีนั้นสำหรับงานประเภทใด



## บทที่ 2

# แนะนำเครื่องมือพัฒนาโปรแกรมด้วยภาษา Python

ในบทนี้เราจะได้เรียนรู้ วิธีการติดตั้งตัวแปรภาษา Python และเครื่องมือที่ใช้พัฒนาโปรแกรมภาษา Python โดยพื้นฐานแล้วไฟล์ Python Script (Python Script) คือไฟล์ข้อความที่ไม่มีการเข้ารหัส (Plain Text) บรรจุคำสั่งในภาษา Python และบันทึกโดยใช้นามสกุล .py ซึ่งสามารถใช้โปรแกรมอ่านหรือแก้ไขข้อความ (Text Editor) อย่างเช่น Notepad ในการจัดการได้ นั่นหมายความว่า เราต้องการเพียง Text Editor และตัวแปลงภาษา Python ก็สามารถพัฒนาโปรแกรมภาษา Python ได้แล้ว

ทั้งนี้เครื่องมือ Text Editor อย่างเช่น Vim, Atom, Visual Studio Code<sup>1</sup>, Sublime Text<sup>2</sup> หรือชุดเครื่องมือพัฒนาโปรแกรม (Integrated Development Environment) อย่างเช่น PyCharm<sup>3</sup> ซึ่งให้เลือกด่วนโดยกดคำสั่งที่ต้องการ แบบมีค่าใช้จ่ายและแบบฟรี (ซึ่งคุณสมบัติต่างกัน) และ Spyder<sup>4</sup> ที่ได้เข้ามาเป็นทางเลือกใหม่สำหรับผู้ที่ต้องการเขียนโค้ด Python ให้ทำงานได้อย่างมีประสิทธิภาพเพิ่มขึ้น

สำหรับเครื่องมือที่นิยมนิยมพัฒนาโปรแกรมด้าน Data Science และนำมายังงานโดยไม่เสียค่าใช้จ่ายคือ Jupyter Notebook หรือ Jupyter Lab ซึ่งเป็นอีกเครื่องมือที่ได้รับการพัฒนาให้ใช้งานได้สะดวกขึ้น โดยสามารถใช้งานผ่านระบบคลาวด์ (Cloud Computer Systems) ได้

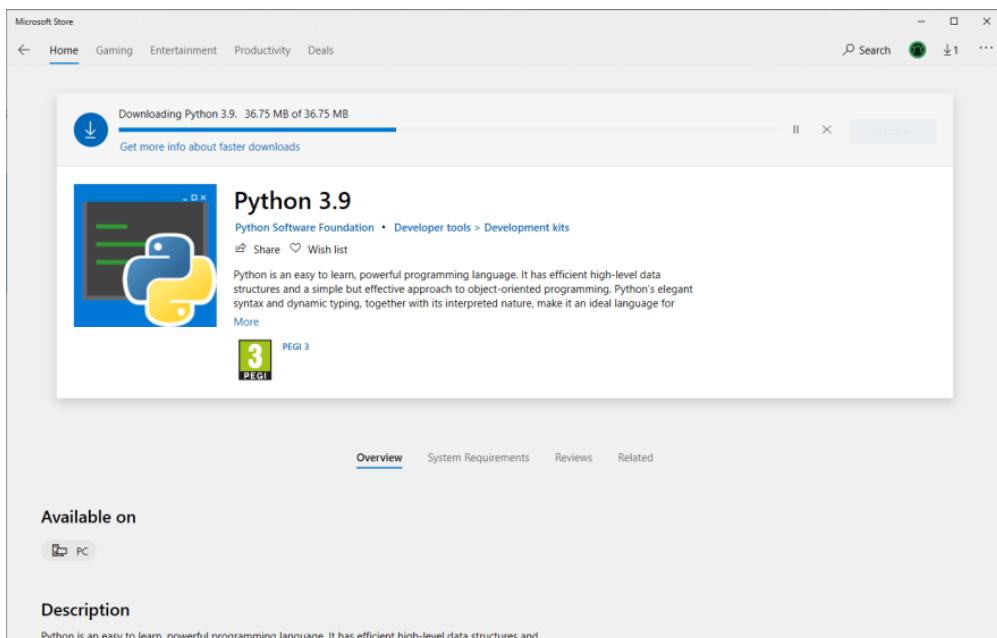
---

<sup>1</sup>Microsoft Visual Studio Code, ดูเพิ่มเติม <https://code.visualstudio.com>

<sup>2</sup>Sublime Text: Text Editing, Done Right, ดูเพิ่มเติม <https://www.sublimetext.com>

<sup>3</sup>PyCharm: The Python IDE for Professional Developers, ดูเพิ่มเติม <https://www.jetbrains.com/pycharm/>

<sup>4</sup>Spyder: The Scientific Python Development Environment, ดูเพิ่มเติม <https://www.spyder-ide.org>



รูป 2.1: การดาวน์โหลดและติดตั้งตัวแปลงภาษาไพธอนผ่าน Microsoft Store

## 2.1 การติดตั้งตัวแปลงภาษาไพธอน

ในที่นี้เราจะแนะนำวิธีการติดตั้งตัวแปลงภาษาไพธอนด้วยวิธีอ่ายจ่ายบนระบบปฏิบัติการ Microsoft Windows 11, macOS Ventura และ Ubuntu Linux 22.04 LTS

### 2.1.1 การติดตั้งบนระบบปฏิบัติการ Microsoft Windows 11

ทำการติดตั้งตัวแปลงภาษาไพธอนผ่าน Microsoft Store ดังนี้

1. เปิดโปรแกรม Microsoft Store หากผู้อ่านไม่ทราบตำแหน่งของโปรแกรมนี้ ให้ทำการคลิกที่ Start menu (ไอคอนรูป Windows บริเวณด้านล่างทางซ้าย) และพิมพ์ "Microsoft Store" (ไม่รวมเครื่องหมาย") จากนั้นจึงคลิกที่ไอคอนเพื่อเปิด Microsoft Store
2. เมื่อ Microsoft Store เปิดขึ้นมา ให้ทำการค้นหา (Search) จากแถบเมนูบริเวณด้านบนขวาโดยการพิมพ์ "Python" (ไม่รวมเครื่องหมาย")
3. ให้ทำการเลือกเวอร์ชันตัวแปลงภาษาไพธอนที่ต้องการติดตั้ง (แนะนำให้เลือกเวอร์ชันล่าสุด นอกเสียจากว่ามีความจำเป็นในการต้องเลือกใช้งานเวอร์ชันก่อนหน้า โดยข้อมูล ณ วันที่เขียนตั้งฉบับนี้คือ ไพธอนเวอร์ชัน 3.11.0) หลังจากเลือกเวอร์ชันได้แล้วให้ทำการคลิก Get เพื่อเริ่มการติดตั้ง

- หลังจากการดาวน์โหลดและติดตั้งเสร็จสิ้น ให้ทำการเปิดโปรแกรม Command Prompt หรือ PowerShell หรือ Terminal (หากไม่ทราบตำแหน่งโปรแกรมให้ทำการค้นหาผ่าน Start menu) และทำการพิมพ์คำสั่ง

```
python --version
```

เพื่อยืนยันความสมบูรณ์ของการติดตั้ง โดยผลลัพธ์จากคำสั่งควรจะเป็นไฟรอนเวอร์ชันที่เราได้ทำการเลือกไว้

- ในกระบวนการติดตั้งตัวแปลงภาษาไฟรอนผ่าน Microsoft Store นั้นจะรวมถึงโปรแกรม pip ที่ใช้ในการจัดการแพคเกจที่ไม่ได้ถูกติดตั้งมากับตัวแปลงภาษาได้ในอนาคต ให้ทำการเปิดโปรแกรม Command Prompt หรือ PowerShell หรือ Terminal และทำการพิมพ์คำสั่ง

```
pip --version
```

เพื่อยืนยันความสมบูรณ์ของการติดตั้ง โดยผลลัพธ์จากคำสั่งควรจะเป็นเวอร์ชันของ pip ที่ได้ทำการติดตั้งไป

### 2.1.2 การติดตั้งบนระบบปฏิบัติการ Ubuntu Linux 22.04 LTS

ระบบปฏิบัติการ Ubuntu Linux 22.04 LTS นั้นถูกพัฒนาอยู่บนฐานของ Debian Linux ซึ่งมาพร้อมกับตัวแปลงภาษา Python เวอร์ชัน 3+ อุปกรณ์ที่มีความสามารถใช้คำสั่งในการอัปเดตเวอร์ชันได้ดังนี้

- เปิดโปรแกรม Terminal และใช้คำสั่งต่อไปนี้

```
sudo apt update  
sudo apt -y upgrade
```

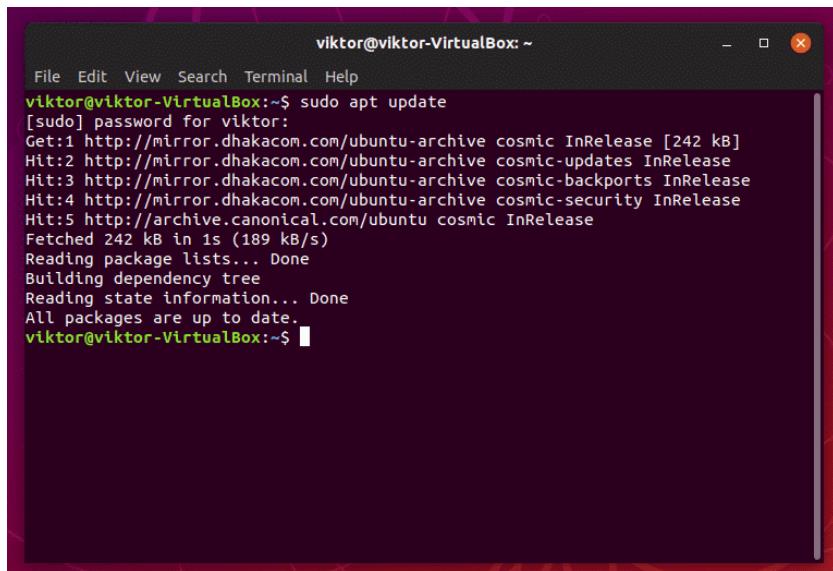
เพื่อทำการอัปเกรด (จำเป็นต้องกรอกรหัสผ่านของผู้ใช้ที่มีสิทธิในการเข้าถึง)

- หลังจากการดาวน์โหลดและติดตั้งเสร็จสิ้น ให้ทำการใช้คำสั่ง

```
python3 --version
```

เพื่อยืนยันความสมบูรณ์ของการติดตั้ง โดยผลลัพธ์จากคำสั่งควรจะเป็นไฟรอนเวอร์ชันล่าสุด

- เพื่อความสะดวกในการติดตั้งแพคเกจของไฟรอนในอนาคต เราจะทำการติดตั้งโปรแกรมจัดการแพคเกจของไฟรอน (Python Package Manager) pip โดยใช้คำสั่ง



รูป 2.2: การใช้โปรแกรม Terminal บน Ubuntu Linux 22.04 LTS

```
sudo apt install -y python3-pip
```

แล้วทำการพิมพ์คำสั่ง

```
pip3 --version
```

เพื่อยืนยันความสมบูรณ์ของการติดตั้ง โดยผลลัพธ์จากคำสั่งควรจะเป็นเวอร์ชันของ pip ที่ได้ทำการติดตั้งไป

### 2.1.3 การติดตั้งบนระบบปฏิบัติการ macOS Ventura

ระบบปฏิบัติการ macOS Ventura มาพร้อมกับตัวแปลงภาษา Python เวอร์ชัน 3+ อญ্যแລာ ทั้งนี้ เราสามารถติดตั้งเวอร์ชันที่ใหม่กว่าได้ผ่าน Homebrew<sup>5</sup> ซึ่งเป็น Package Manager ที่รองรับระบบปฏิบัติการ macOS ดังนี้

1. เปิดโปรแกรม Terminal และใช้คำสั่งต่อไปนี้

<sup>5</sup>Homebrew: The Missing Package Manager for macOS, ดูเพิ่มเติม <https://brew.sh>

```

epsilon@VEKTOR:~ >>> brew update
Already up-to-date.

epsilon@VEKTOR:~ >>> brew search python
=> Formulae
app-engine-python      ipython          python-tk@3.9
boost-python            micropython     python-yq
boost-python3           ptpython        python@3.7
bpython                 python-markdown  python@3.8
gst-python              python-tabulate  python@3.9 ✓
=> Casks
awips-python           mysql-connector-python

If you meant "python" specifically:
It was migrated from homebrew/cask to homebrew/core.

epsilon@VEKTOR:~ >>> brew install python@3.9

```

รูป 2.3: การติดตั้งตัวแปลงภาษาไฟรอนผ่าน Homebrew บน macOS Ventura

```

/bin/bash -c "$(curl -fsSL https://
← raw.githubusercontent.com/Homebrew/install/HEAD/
← install.sh)"

```

เพื่อทำการติดตั้ง Homebrew (จำเป็นต้องกรอกรหัสผ่านของผู้ใช้ที่มีสิทธิ์ในการเข้าถึง) โดยทำตามคำแนะนำที่ปรากฏ

2. หลังจากการดาวน์โหลดและติดตั้งเสร็จสิ้น (หากมีการแจ้งเตือนภัยหลังการติดตั้ง ให้ทำการติดตั้งตามคำแนะนำที่ได้กำหนดไว้) ให้ทำการใช้คำสั่ง

```

brew update
brew search python

```

เพื่อทำการอัพเดทฐานข้อมูลของ Homebrew และ ค้นหาเวอร์ชันของ Python ที่สามารถติดตั้งได้

3. เมื่อทำการเลือกเวอร์ชันที่จะติดตั้งได้แล้วให้ใช้คำสั่ง

```
brew install python@3.x
```

เพื่อทำการติดตั้ง โดยต้องแทนที่ x ด้วยเลขของ Python เวอร์ชันที่ค้นหาพบจากขั้นตอนก่อนหน้า  
นี้

4. หลังจากการดาวน์โหลดและติดตั้งเสร็จสิ้น (หากมีการแจ้งเตือนภายในหลังการติดตั้ง ให้ทำการตามคำแนะนำที่ได้กำหนดไว้) ให้ใช้คำสั่ง

```
python3 --version
pip3 --version
```

เพื่อยืนยันความสมบูรณ์ของการติดตั้ง Python และ pip (ติดตั้งมาพร้อมกัน) โดยผลลัพธ์จะแสดง  
สิ่งควรจะเป็นเวอร์ชันที่เราได้ทำการเลือกไว้

#### 2.1.4 การติดตั้งผ่าน pyenv

สำหรับระบบปฏิบัติการ macOS Ventura เราสามารถติดตั้งตัวแปลงภาษา Python ได้หลายเวอร์ชัน  
และเลือกใช้งานได้ตามความเหมาะสมผ่าน pyenv ทั้งนี้กระบวนการติดตั้งนั้นค่อนข้างซับซ้อนแต่ก็แลก  
มาด้วยความยืดหยุ่นที่มากกว่าในการพัฒนา

วิธีการต่อไปนี้ไม่แนะนำสำหรับผู้เริ่มต้น หรือผู้ที่ไม่ถนัดในการใช้งาน **Command Line**

1. อัปเดท Homebrew และติดตั้งแพคเกจที่จำเป็น

```
brew update
brew install zlib sqlite bzip2 libiconv libzip
```

2. ติดตั้ง pyenv<sup>6</sup> ผ่าน Homebrew ด้วยคำสั่ง

```
brew install pyenv
```

3. หลังจากการดาวน์โหลดและติดตั้งเสร็จสิ้น ให้ใช้คำสั่ง

---

<sup>6</sup>pyenv: Simple Python Version Management, ดูเพิ่มเติม <https://github.com/pyenv/pyenv>

```
echo -e $'if command -v pyenv 1>/dev/null 2>&1; then\n  \n    export PYENV_ROOT="$HOME/.pyenv"\n    export\n    PATH="$PYENV_ROOT/bin:$PATH"\n    eval "$(pyenv\n    init --path)"\n    eval "$(pyenv init -)"'\nfi' >>\n~/zshrc
```

เพื่อตั้งค่าการทำงานของ pyenv อัตโนมัติทุกครั้งที่เข้าใช้งาน

#### 4. ใช้คำสั่ง

```
pyenv install --list
```

เพื่อดูว่ามี Python ได้บ้างที่เราสามารถติดตั้งได้

#### 5. หลังจากเลือกเวอร์ชันที่ต้องการได้แล้วให้ทำการใช้คำสั่ง

```
pyenv install x.y.z\npyenv rehash
```

เพื่อทำการติดตั้ง โดยที่ x.y.z คือตัวเลขเวอร์ชันของ Python ที่ได้มาจากการตั้งค่า (เช่น 3.9.5 เป็นต้น)

#### 6. ใช้คำสั่ง

```
pyenv versions
```

เพื่อดูว่ามี Python ได้บ้างที่ได้ติดตั้งไปแล้ว โดยที่ผลลัพธ์ที่ได้หากมีเครื่องหมาย \* อยู่หน้าเวอร์ชันได้หมายถึงเวอร์ชันนั้นได้ถูกเลือกใช้งาน (ค่าเริ่มต้นจะเป็น systems)

#### 7. เราสามารถเลือกเวอร์ชันของ pyhon ที่ต้องการได้โดยใช้คำสั่ง

```
pyenv global x.y.z
```

#### 8. เพื่อยืนยันความถูกต้อง ให้ใช้คำสั่ง

```
python --version  
pip --version
```

โดยผลลัพธ์จากคำสั่งควรจะเป็นเวอร์ชันที่เราได้ทำการเลือกไว้ ทั้งนี้คำสั่งของ pyenv โดยละเอียดสามารถสืบค้นได้จาก Repository<sup>7</sup> ของผู้พัฒนาโดยตรง

## 2.2 การใช้งานไฟรอนสคริปต์

เราจะเป็นต้องเตรียมไฟรอนสคริปต์ผ่าน Text Editor เช่น Notepad, Visual Studio Code หรือ Sublime Text เป็นต้น ผู้อ่านสามารถเลือกใช้ได้ตามความต้องการ

1. เปิด Text Editor ที่ผู้อ่านคุ้นเคย และสร้างไฟล์ขึ้นมาใหม่
2. พิมพ์ข้อความ

```
print('Hello World')
```

แล้วทำการบันทึกไฟล์ (Save หรือ Save as) โดยให้มีนามสกุลเป็น .py เช่น hello.py ไฟล์ที่ได้นี้จะเรียกว่าไฟรอนสคริปต์

3. หลังจากบันทึกไฟรอนสคริปต์เรียบร้อยแล้ว เราสามารถสั่งให้สคริปต์นี้ทำงานได้โดยการเปิด Terminal แล้วใช้คำสั่ง

```
python hello.py
```

หรือ

```
python3 hello.py
```

ขึ้นอยู่กับว่าตัวแปลภาษาไฟรอนถูกติดตั้งด้วยวิธีการใดในหัวข้อ 2.1

---

<sup>7</sup><https://github.com/pyenv/pyenv>

## 2.3 รู้จักกับเครื่องมือ Jupyter Notebook



ผลข้อความ กราฟฟิค และสมการคณิตศาสตร์

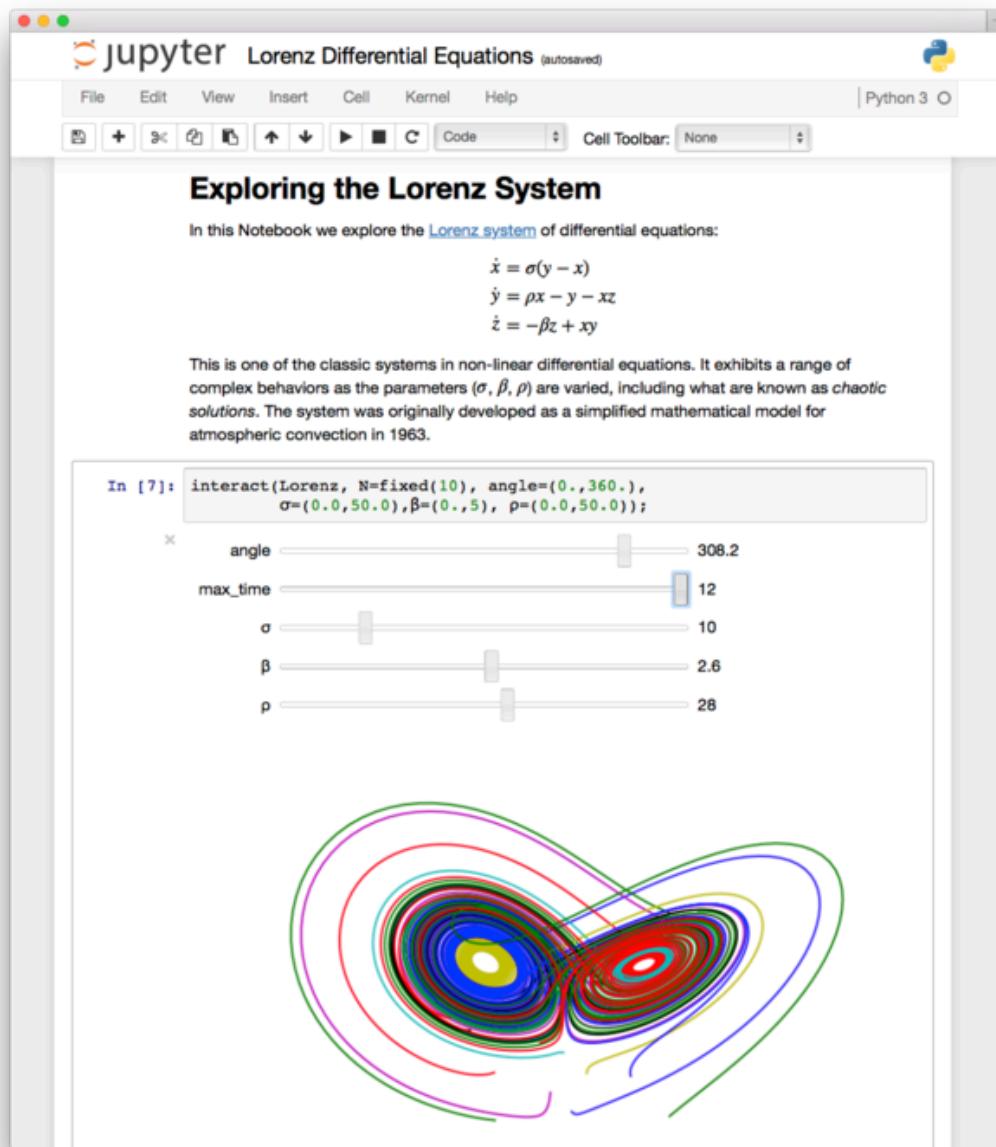
Jupyter Notebook ถูกนิยมใช้ในงาน การทำความสะอาดและแปลงข้อมูล (Data Cleaning and Transformation) การจำลองสถานการณ์เชิงตัวเลข (Numerical Simulation) การทำแบบจำลองทางคณิตศาสตร์ (Mathematical Modeling) การแสดงกราฟฟิคของข้อมูล (Data Visualization) และ การเรียนรู้ของเครื่องจักร (Machine Learning)

องค์ประกอบของเครื่องมือ Jupyter Notebook มีอยู่ด้วยกัน 4 ส่วนคือ

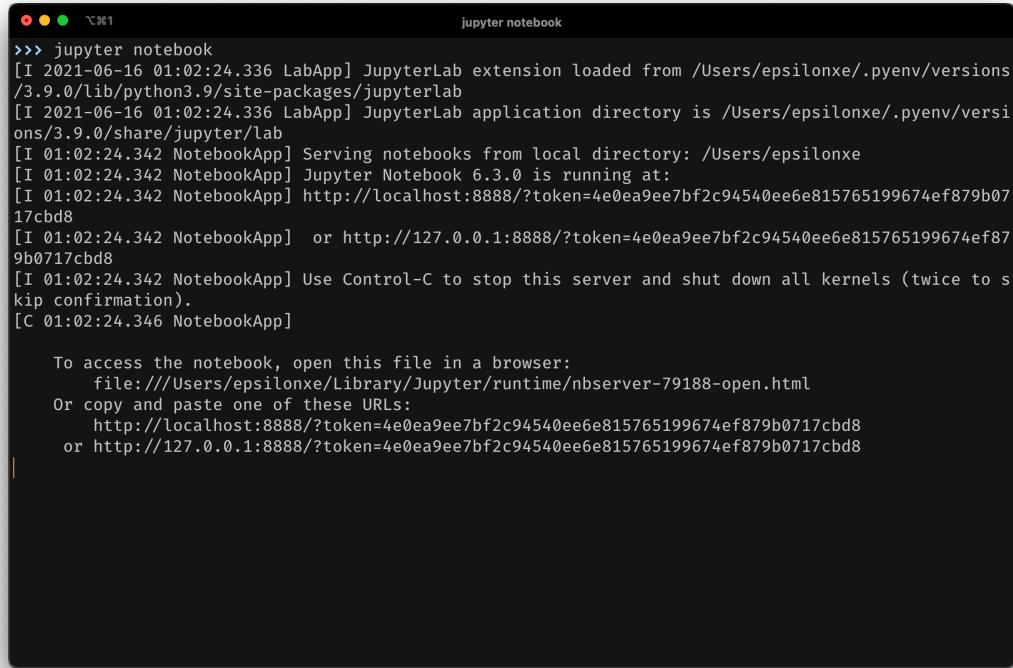
1. Notebook Document หรือเรียกว่า Notebook คือ หน้าเว็บбраузertextที่ใช้ทำการเขียนคำสั่งโปรแกรม เท็กข้อความ คำอธิบาย รูปภาพ กราฟ รวมไปถึงผลลัพธ์ที่ได้จากการประมวลผลคำสั่งโปรแกรม
2. Jupyter Notebook App คือ แอพพลิเคชัน server-client ซึ่งช่วยให้ทำการเปิด Notebook ผ่านเว็บбраузertextโดยไม่ต้องใช้อินเตอร์เน็ต หรือจะติดตั้งให้ทำงานแบบ Server ซึ่งสามารถเข้าใช้งานพร้อมกันหลายคน คุณก็ได้
3. Kernel คือ ตัวประมวลผลคำสั่งโปรแกรม (Computation engine) ของแต่ละภาษาที่มีอยู่ใน Notebook ซึ่งเครื่องมือ Jupyter Notebook ได้ติดตั้ง IPython kernel สำหรับประมวลผลคำสั่งภาษาไพธอนไว้ให้เรียบร้อยแล้ว สำหรับภาษาอื่น ๆ ต้องติดตั้งเพิ่มเติมในภายหลัง
4. Notebook Dashboard คือ หน้าเว็บбраузertextแรกหลังจากสั่งให้ Jupyter Notebook App ทำงาน ซึ่งอำนวยความสะดวกให้กับผู้ใช้งานจัดการสร้างไฟล์เดอร์จัดเก็บไฟล์คำสั่งโปรแกรมที่ถูกเขียน Notebook หรือตรวจสอบไฟล์ที่กำลังประมวลผลคำสั่งโปรแกรมอยู่ ณ ขณะนั้น หรือการเปลี่ยนชื่อไฟล์หรือไฟล์เดอร์รวมไปถึงการลบไฟล์หรือไฟล์เดอร์ด้วย

---

<sup>8</sup>IPython: <https://ipython.org>



รูป 2.4: ตัวอย่างการใช้งาน Jupyter Notebook ในการทำแบบจำลองทางคณิตศาสตร์



```
jupyter notebook
>>> jupyter notebook
[I 2021-06-16 01:02:24.336 LabApp] JupyterLab extension loaded from /Users/epsilonxe/.pyenv/versions/3.9.0/lib/python3.9/site-packages/jupyterlab
[I 2021-06-16 01:02:24.336 LabApp] JupyterLab application directory is /Users/epsilonxe/.pyenv/versions/3.9.0/share/jupyter/lab
[I 01:02:24.342 NotebookApp] Serving notebooks from local directory: /Users/epsilonxe
[I 01:02:24.342 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[I 01:02:24.342 NotebookApp] http://localhost:8888/?token=4e0ea9ee7bf2c94540ee6e815765199674ef879b0717cbd8
[I 01:02:24.342 NotebookApp] or http://127.0.0.1:8888/?token=4e0ea9ee7bf2c94540ee6e815765199674ef879b0717cbd8
[I 01:02:24.342 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 01:02:24.346 NotebookApp]

To access the notebook, open this file in a browser:
  file:///Users/epsilonxe/Library/Jupyter/runtime/nbserver-79188-open.html
Or copy and paste one of these URLs:
  http://localhost:8888/?token=4e0ea9ee7bf2c94540ee6e815765199674ef879b0717cbd8
  or http://127.0.0.1:8888/?token=4e0ea9ee7bf2c94540ee6e815765199674ef879b0717cbd8
```

รูป 2.5: การเรียกใช้งาน Jupyter Notebook

### 2.3.1 การติดตั้งและการเปิดใช้งาน Jupyter Notebook

Jupyter Notebook สามารถติดตั้งผ่านคำสั่ง pip ได้โดยการเปิด Terminal และใช้คำสั่งต่อไปนี้

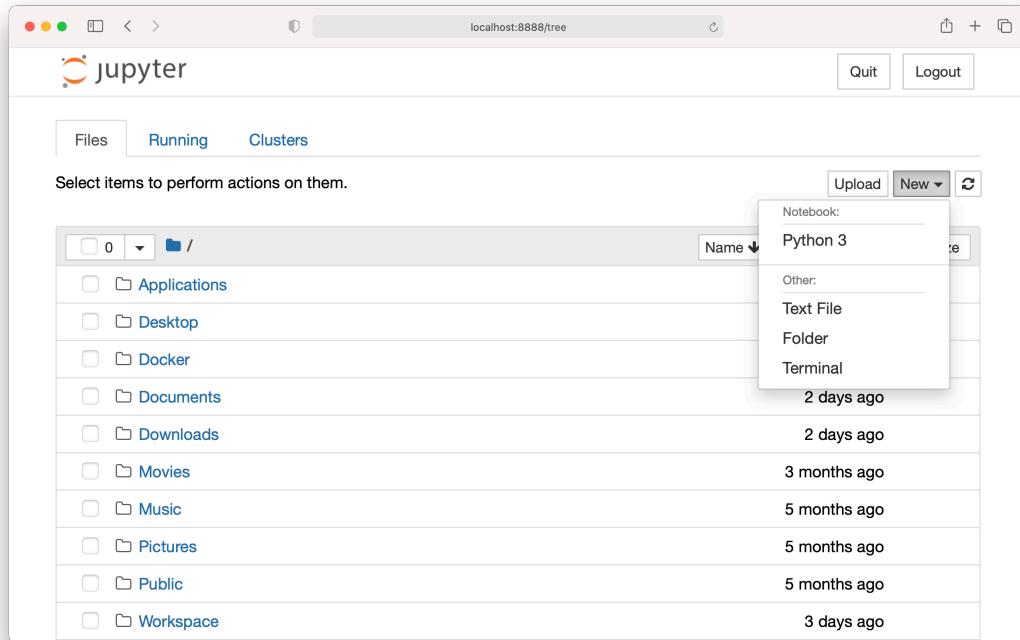
```
pip install --upgrade pip
pip install jupyter
```

หลังจากที่ได้ดาวน์โหลดและติดตั้งเครื่องมือ Jupyter Notebook สำหรับเขียนโปรแกรมด้วยภาษา Python เรียบร้อยแล้ว เราสามารถเปิดการใช้งาน Jupyter Notebook ได้โดยใช้คำสั่ง

```
jupyter notebook
```

หรือ

```
python -m notebook
```



รูป 2.6: Notebook Dashboard ของ Jupyter Notebook

สังเกตจากรูป 2.5 แสดงถึงการเรียกใช้งาน Jupyter Notebook และหลังจากนั้นหน้าต่างของเว็บбраว์เซอร์จะเปิดไปที่หน้า <http://localhost:8888/tree> สังเกตได้ว่ามีเมนูต่าง ๆ ให้ได้เลือกใช้งาน ดังรูป 2.6 โดยในหน้าเวบนี้จะมีส่วนที่ติดต่อผู้ใช้ (User Interface) ที่ทำหน้าที่ต่าง ๆ ดังตาราง 2.1 ทั้งนี้เรายังสามารถใช้งาน Jupyter Notebook ผ่านระบบ Cloud ได้เช่น Google Colab<sup>9</sup> (รูป 2.7), Jupyter Mybinder<sup>10</sup>, COCALC Jupyter Notebook<sup>11</sup> เป็นต้น

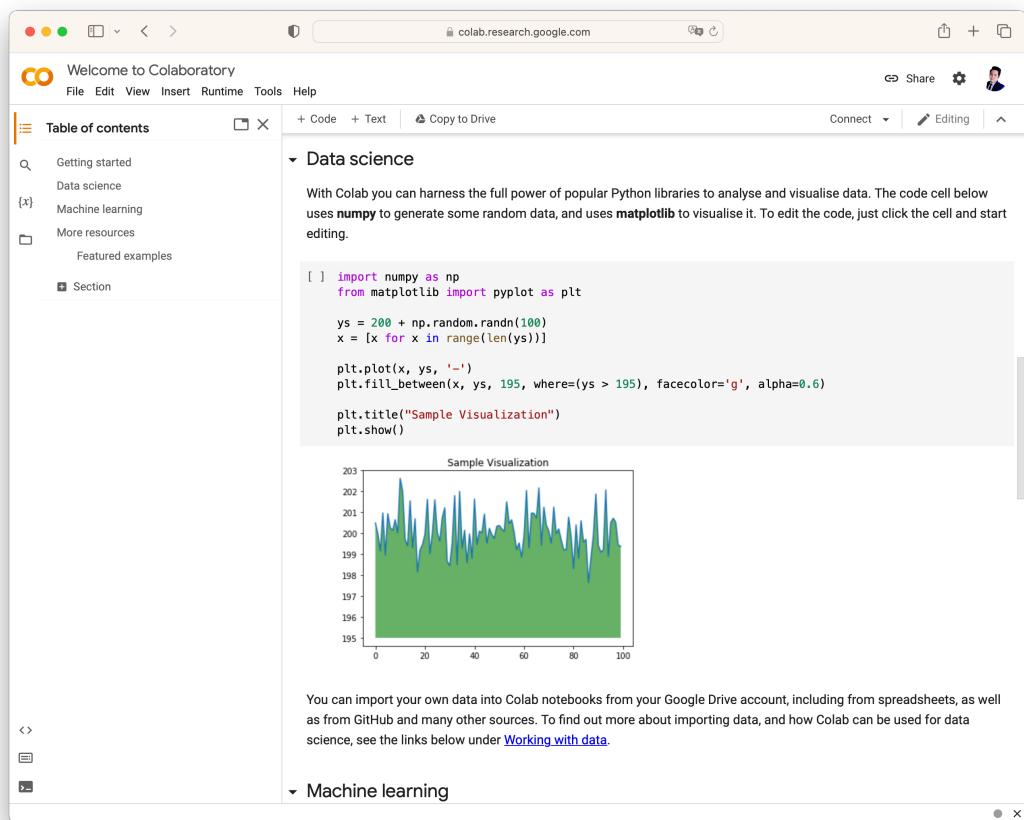
<sup>9</sup>Google Colab: <https://colab.research.google.com>

<sup>10</sup>Jupyter Mybinder: <https://jupyter.org/try>

<sup>11</sup>COCALC Jupyter Notebook: <https://cocalc.com/doc/jupyter-notebook.html>

ส่วนติดต่อผู้ใช้		การใช้งาน
Files		แสดงชื่อ File และ Folder ที่ได้สร้างไว้สำหรับเก็บ File งาน
Running		แสดงชื่อ File ต่าง ๆ ที่กำลังทำงานอยู่
Clusters		ใช้สำหรับกรณีที่มีการติดตั้ง Jupyter Notebook ไว้หลายเครื่องให้ประมวลผลข้อมูลแบบ
Logout		ใช้สำหรับออกจากการเขียนโปรแกรม เมื่อเราสร้างรหัสผ่านเข้าใช้งาน Jupyter Notebook
Quit		ปุ่มปิดการทำงานของ Jupyter Notebook
Upload		นำ File จากแฟล์อื่นเข้ามาใน Jupyter Notebook
New		มีเมนูย่อยให้เลือกใช้งานอีก 3 เมนู สำหรับสร้าง File หรือ Folder และแสดงภาษาโปรแกรมที่สามารถพัฒนาโปรแกรมผ่านเครื่องมือ Jupyter Notebook ได้ จากรูป 2.6 แสดงเพียงแค่ภาษาไพธอนเวอร์ชัน 3 (Python 3) ถ้าติดตั้งหลายภาษาจะปรากฏบนแถบนี้
Text File		ใช้สร้าง Text File ข้อความ
Folder		สำหรับสร้าง Folder เก็บไฟล์คำสั่งโปรแกรมหรือ Text File ที่ได้สร้างขึ้นมาใช้งาน
Terminal		เปิด Terminal บนหน้าเว็บбраузอร์เพื่อเข้าสู่โหมด Command Line

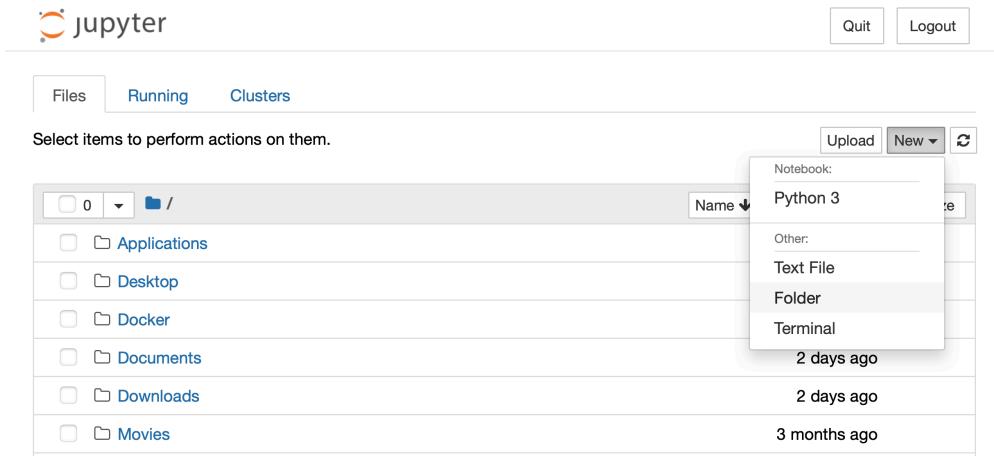
ตาราง 2.1: ส่วนติดต่อผู้ใช้ในหน้า Notebook Dashboard ของ Jupyter Notebook



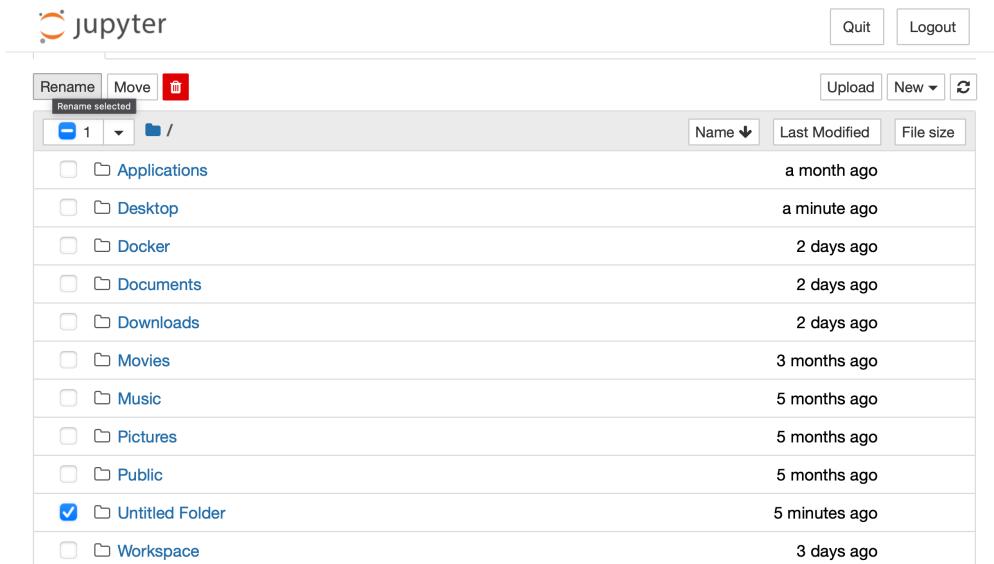
ສູ່ 2.7: Jupyter Notebook ປະຈະບັນຂອງ Google Colab

## 2.4 เริ่มต้นใช้งาน Jupyter Notebook

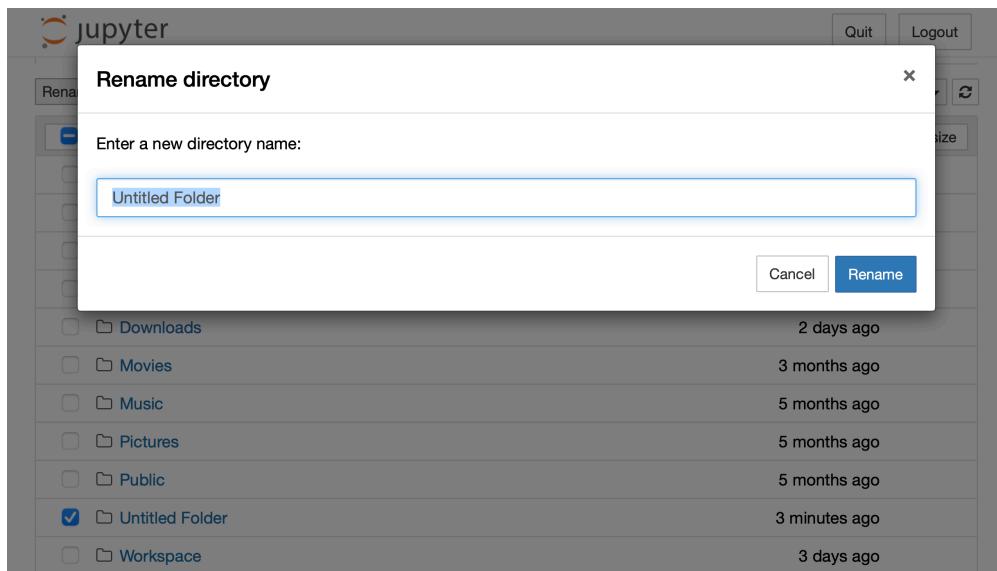
1. จากหน้าต่างหลัก Jupyter Notebook คลิกเมนู New ▶ Folder เพื่อสร้าง Folder สำหรับเก็บ File



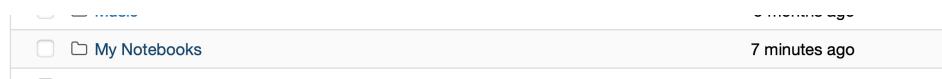
2. ผู้อ่านจะสังเกตเห็น Folder ที่สร้างขึ้นมาใหม่มีชื่อเป็น Untitled Folder จากนั้นให้คลิกที่หน้า Check Box ของโฟลเดอร์ใหม่นั้น แล้วคลิกที่ปุ่ม Rename เพื่อเปลี่ยนชื่อให้เหมาะสม



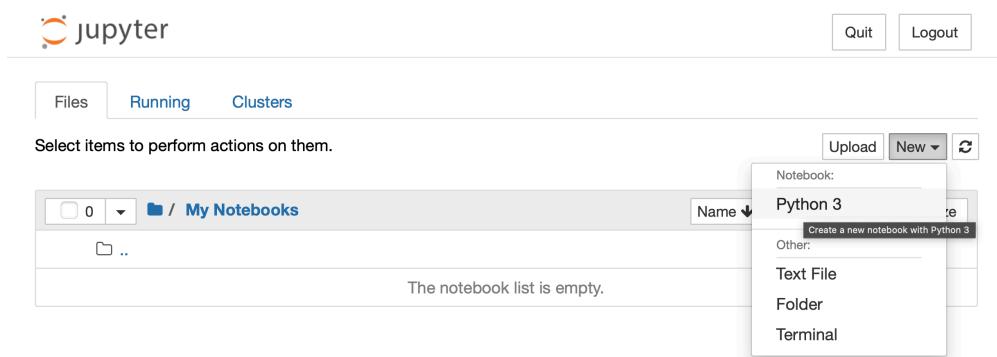
- จะปรากฏหน้าต่างให้ผู้อ่านเปลี่ยนชื่อ Folder ใหม่ เสร็จแล้วให้คลิกปุ่ม Rename หรือกดคีย์ Enter



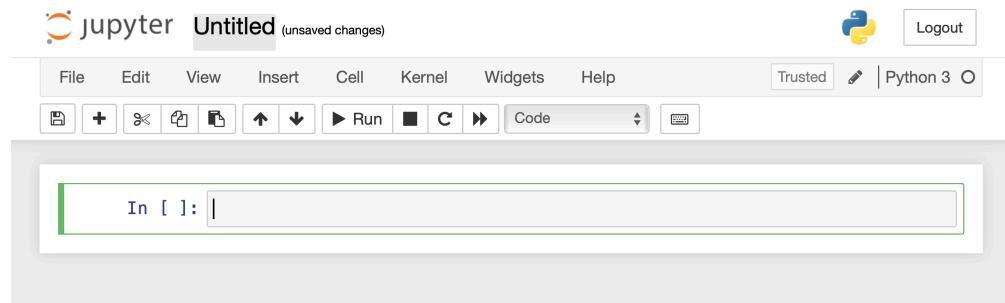
- จะเห็นว่าชื่อ Folder ได้ถูกเปลี่ยนเป็นชื่อใหม่ตามที่กำหนดเรียบร้อยแล้ว



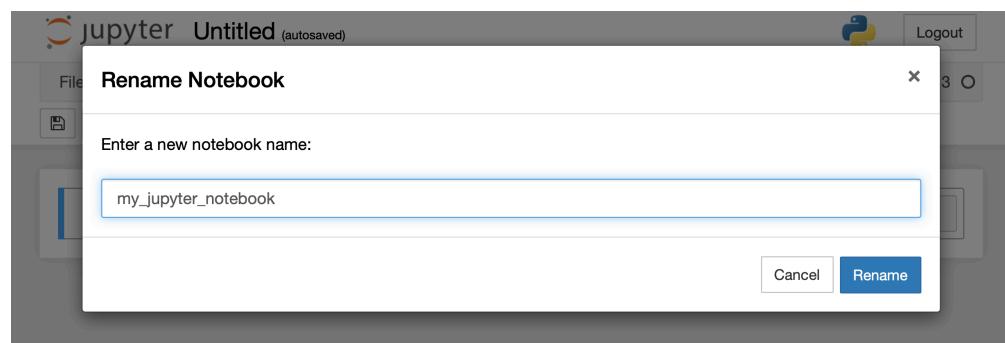
- จากนั้นให้คลิกเลือกที่ไฟล์เดอร์ แล้วคลิกเมนู New ▶ Python 3



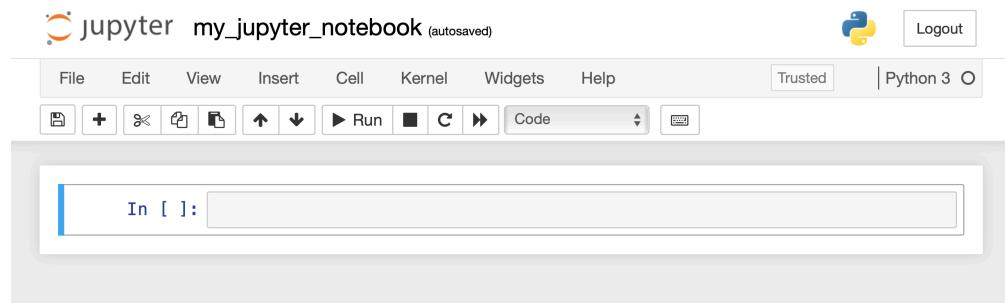
6. เมื่อสร้างไฟล์ใหม่จะมีชื่อเป็น Untitled ผู้อ่านสามารถเปลี่ยนชื่อไฟล์ได้โดยการคลิกที่ชื่อ Untitled



7. เปลี่ยนชื่อใหม่ตามต้องการแล้วคลิกปุ่ม Rename หรือกดคีย์ Enter



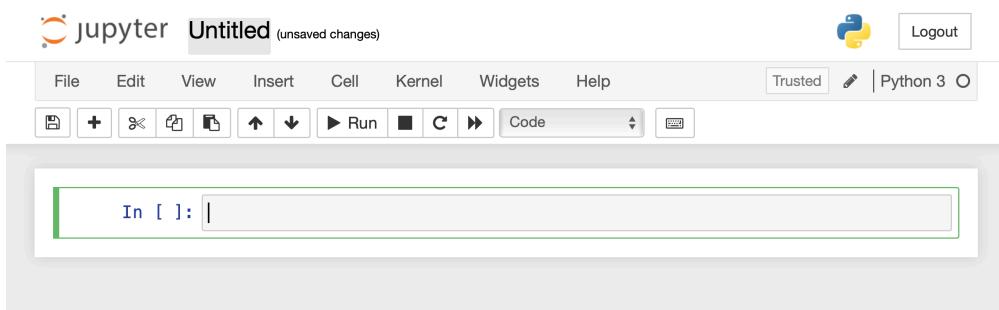
8. หลังจากเปลี่ยนชื่อไฟล์เสร็จ จะได้ชื่อไฟล์ใหม่ดังภาพ



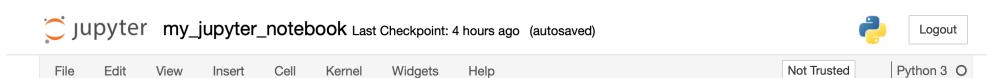
## 2.4.1 แนะนำเมนูต่าง ๆ ก่อนเขียนคำสั่งโปรแกรม

ก่อนการสร้างไฟล์เพื่อเริ่มต้นเขียนโปรแกรมเราจำเป็นต้องสร้าง Folder สำหรับจัดเก็บไฟล์ให้เป็นหมวดหมู่ เพื่อให้ง่ายต่อการค้นหา เมื่อเข้าสู่หน้าต่างการเขียนโปรแกรมซึ่งต่อไปจะเรียกว่า Notebook ซึ่งมีเมนูต่าง ๆ ดังนี้

1. ชื่อ Notebook ผู้อ่านสามารถเปลี่ยนชื่อได้โดยการคลิกที่ชื่อ Untitled



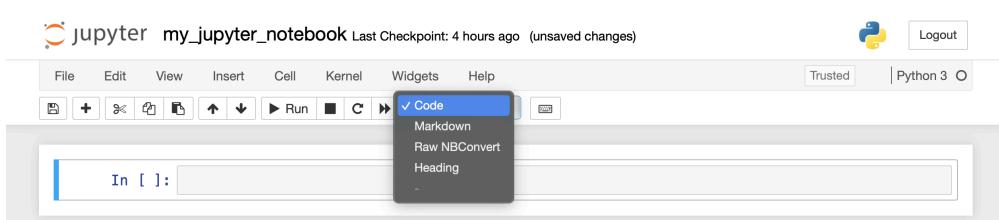
2. Menu Bar สังเกตเห็นว่าจะมีเมนูให้ใช้งานเหมือนกับโปรแกรมทั่ว ๆ ไป ที่ผู้อ่านคุ้นเคยกัน เช่น File, Edit, View, Insert และมีเมนู Cell สำหรับสั่งให้ Cell ที่เลือกนั้นทำการประมวลผล โปรแกรม เมนู Kernel ใช้สำหรับ Start หรือ Shutdown การทำงานของ Jupyter Notebook และยังมี Menu Widgets และ Menu Help ไว้ขอความช่วยเหลือ เพื่อขอคำอธิบาย



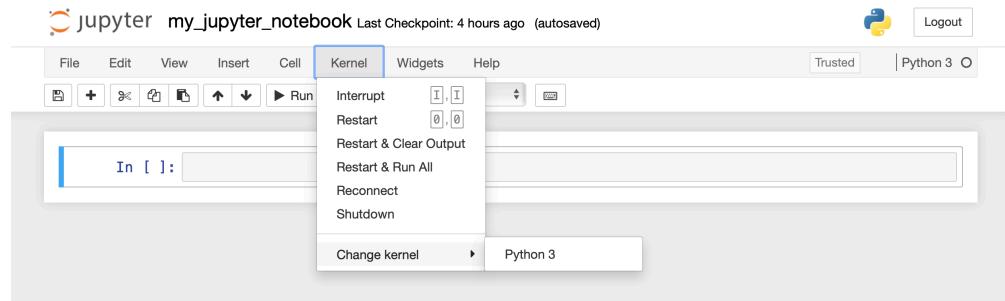
3. Tool Bar เป็นเครื่องมือทางลัดเลือกสั่งงานต่าง ๆ โดยไม่ต้องเข้าไปคลิกเลือกที่ Menu Bar



4. Cell Type จะแสดงให้ทราบถึงในขณะนี้กำลังใช้ Cell ประเภทใดอยู่

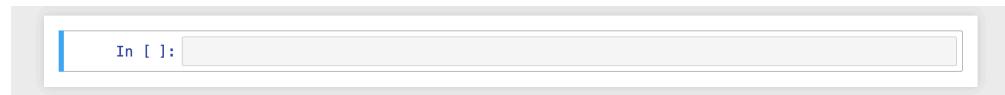


5. Kernel หากได้ทำการติดตั้งหลายภาษาสำหรับเขียนโปรแกรม สามารถเปลี่ยน Kernel ของภาษาได้จากเมนูนี้ หรือสั่งให้ Kernel ภาษาที่พร้อมเริ่มต้นทำงานใหม่



6. Logout จะใช้สำหรับกรณีที่ได้กำหนดรหัสผ่านเข้าใช้งาน Jupyter Notebook

7. Cell แต่ละช่องจะเรียกว่า Cell สำหรับการเขียนคำสั่งโปรแกรม



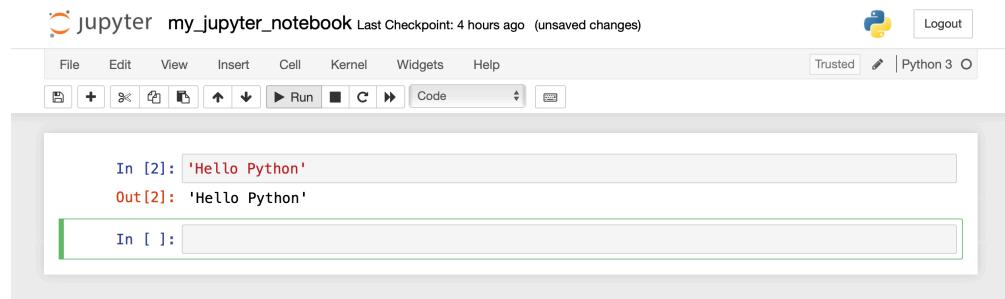
## 2.4.2 การใช้เครื่องมือ Jupyter Notebook เขียนภาษาไพธอน

หลังจากที่ได้รู้จักกับเมนูต่าง ๆ ไปแล้ว ต่อไปจะมาเริ่มต้นเขียนโปรแกรมภาษาไพธอนบน Notebook กัน เพื่อสร้างความคุ้นเคย ก่อนที่จะลงมือปฏิบัติการเขียนโปรแกรมภาษาไพธอนอย่างจริงจัง

1. ให้พิมพ์ข้อความ

'Hello Python'

โดยข้อความทั้งหมดจะต้องอยู่ในเครื่องหมาย Single Quote ('...') ลงในช่อง Cell จากนั้นให้ คลิกปุ่ม Run บน Toolbar หรือคลิกเมนู Cell > Run Cells หรือกดคีย์ลัด ⌘+⏎ แล้ว Notebook จะแสดงผลลัพธ์พร้อมทั้งปรากฏ cell ใหม่ขึ้นมาใหม่อัตโนมัติ



## 2. เพื่อเปรียบเทียบการแสดงผลให้พิมพ์

```
print('Hello Python')
```

ลงในช่อง Cell จากนั้น แล้วคลิกปุ่ม **Run** บน Toolbar หรือคลิกเมนู **Cell** > **Run Cells** หรือกดคีย์ลัด **↑ + ←** จะได้ผลลัพธ์ สังเกตเห็นว่าผลลัพธ์ที่แสดงออกมากจะไม่มีเครื่องหมาย Single quote ('...') ครอบข้อความ

```
In [2]: print('Hello Python')
Hello Python
```

## 3. ผู้อ่านสามารถแก้ไขข้อความใน Cell ที่ได้สั่งให้ทำงานไปแล้วได้ โดยให้คลิกเลือก Cell ที่ต้องการแก้ไขข้อความ จากนั้นสั่ง **Run** ใหม่อีกครั้งจะได้ผลลัพธ์ และจะสังเกตเห็นว่าช่อง Cell จะมีการเพิ่มจำนวนครั้งที่สั่งให้ประมวลผลคำสั่งโปรแกรม

```
In [3]: 'Hello World'
Out[3]: 'Hello World'

In [2]: print('Hello Python')
Hello Python

In [ ]:
```

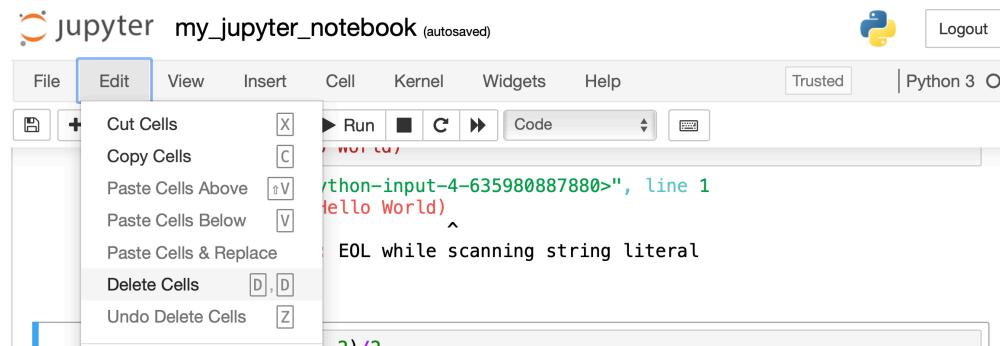
## 4. เมื่อเขียนคำสั่งผิด Jupyter Notebook จะแสดงการแจ้งเตือนข้อผิดพลาด และแสดงตำแหน่งคำสั่งที่เขียนผิด และมีการแจ้งเตือนข้อผิดพลาด (Error) รวมทั้งสาเหตุที่ทำให้เกิด Error

```
In [4]: print('Hello World')
File "<ipython-input-4-635980887880>", line 1
    print('Hello World')
               ^
SyntaxError: EOL while scanning string literal
```

## 5. ในเครื่องมือ Jupyter Notebook แต่ละช่อง Cell สามารถทำการคำนวณพื้นฐาน เช่น การบวก การลบ การคูณ หรือหาร ได้เลย

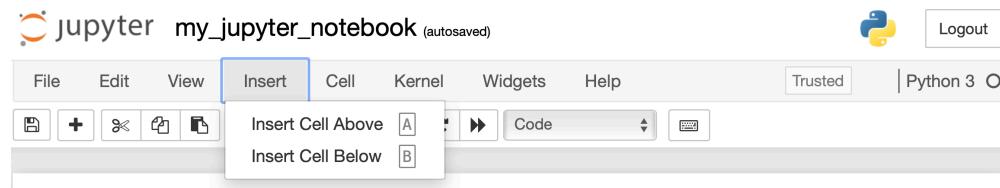
```
In [7]: 8 + 4 * (8 - 2)/2
Out[7]: 20.0
```

6. หากต้องการลบ Cell ที่ไม่ต้องการออก ให้คลิกที่ Cell จากนั้นคลิกเมนู **Edit > Delete Cells** หรือ กดคีย์ลัด **D**, **D** แล้ว Cell ที่ถูกเลือกจะถูกลบออกจาก Notebook



7. หากต้องการเพิ่ม Cell ซึ่งสามารถเพิ่ม Cell ทั้งด้านบนหรือด้านล่าง โดยให้เลือก Cell เดิมก่อน จากนั้นคลิกเมนู

- **Insert > Insert Cell Above** (คีย์ลัด **A**) เพื่อเพิ่ม Cell ด้านบน
- **Insert > Insert Cell Below** (คีย์ลัด **B**) เพื่อเพิ่ม Cell ด้านล่าง



### 2.4.3 การขอความช่วยเหลือ

Jupyter Notebook ได้อี็องวายประโยชน์ให้ผู้ใช้งานได้ขอความช่วยเหลือที่เกี่ยวกับการพัฒนาโปรแกรม ผ่านฟังก์ชัน `help()`

1. ให้ผู้ใช้งานพิมพ์ `help()` ที่ Cell และสั่ง **Run** จะแสดงคำอธิบายให้เราทราบว่า สามารถขอความช่วยเหลืออะไรได้บ้าง หัวข้อที่สามารถให้ความช่วยเหลือได้ คือ modules, keywords และ top-

ics ให้ลองเรียกขอความช่วยเหลือการใช้งาน modules โดยพิมพ์ลงในช่อง Textbox และสั่ง Run อีกครั้ง

```
In [*]: help()  
  
Welcome to Python 3.9's help utility!  
  
If this is your first time using Python, you should definitely check out  
the tutorial on the Internet at https://docs.python.org/3.9/tutorial/.  
  
Enter the name of any module, keyword, or topic to get help on writing  
Python programs and using Python modules. To quit this help utility and  
return to the interpreter, just type "quit".  
  
To get a list of available modules, keywords, symbols, or topics, type  
"modules", "keywords", "symbols", or "topics". Each module also comes  
with a one-line summary of what it does; to list the modules whose name  
or summary contain a given string such as "spam", type "modules spam".  
  
help> modules
```

2. จากผลลัพธ์จะเห็นได้ว่ามี modules ให้เรียกใช้งานจำนวนมาก จากนั้นให้ลองเรียกดูวิธีการใช้งาน module os และสั่ง Run ดูผลลัพธ์

```
_weakrefset      idna          pyparsing        xdrlib  
_xxsubinterpreters  imaplib    pyrsistent      xml  
_xxtestfuzz     imghdr       pytz            xmrpc  
_zoneinfo       imp           qtconsole       xxlimited  
abc              importlib    qtqpy           xssubtype  
aifc             inspect      queue           zipapp  
antigravity    io            quopri          zipfile  
anyio            ipaddress    random          zipimport  
appnope          ipykernel   re               zlib  
argon2           ipykernel_launcher readline      zmq  
argparse         ipython_genutils reprlib        zoneinfo  
  
Enter any module name to get more help. Or, type "modules spam" to search  
for modules whose name or summary contain the string "spam".  
  
help> os|
```

3. ผลลัพธ์จะแสดงรายละเอียดต่าง ๆ เช่น มีฟังก์ชันอะไรบ้าง วิธีการใช้งาน เป็นต้น

```
help> os  
Help on module os:  
  
NAME  
    os - OS routines for NT or Posix depending on what system we're on.  
  
MODULE REFERENCE  
    https://docs.python.org/3.9/library/os  
  
The following documentation is automatically generated from the Python  
source files. It may be incomplete, incorrect or include features that  
are considered implementation detail and may vary between Python  
implementations. When in doubt, consult the module reference at the  
location listed above.  
  
DESCRIPTION  
    This exports:  
        - all functions from posix or nt, e.g. unlink, stat, etc.
```

- หากเราต้องการดูคำอธิบายของฟังก์ชันใดๆ ให้พิมพ์ `help(ชื่อฟังก์ชัน)` และสั่ง `Run` ก็จะแสดงรายละเอียดการใช้งานของฟังก์ชันนั้น ๆ

```
In [5]: help(print)
Help on built-in function print in module builtins:

print(*args, **kwargs)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

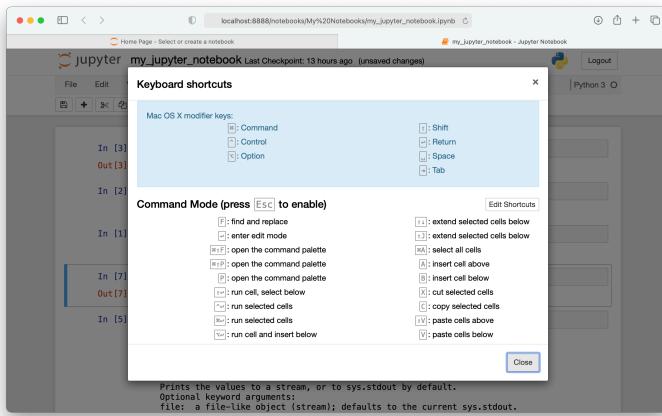
    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
        file: a file-like object (stream); defaults to the current sys.stdout.
        sep:  string inserted between values, default a space.
        end:  string appended after the last value, default a newline.
        flush: whether to forcibly flush the stream.
```

#### 2.4.4 การ Download และ Upload ไฟล์

การ Download เป็นการนำเอาไฟล์โปรแกรมที่ได้พัฒนาบนเครื่องมือ Jupyter Notebook ไปพัฒนาต่อหรือนำไปประมวลผลบนเครื่องคอมพิวเตอร์อื่น ซึ่งอาจจะเป็นระบบปฏิบัติการเดียวกันหรือต่างระบบปฏิบัติการก็ได้ สำหรับการ Upload นั้น เป็นการนำเอาไฟล์โปรแกรมที่พัฒนาจากเครื่องมือ Jupyter Notebook บนเครื่องอื่น เข้าสู่เครื่องมือ Jupyter Notebook เพื่อพัฒนาโปรแกรมต่อหรือประมวลผลบนเครื่องของเราทั้งสองวิธี สามารถทำได้ตามขั้นตอนดังต่อไปนี้

- เมื่อพัฒนาโปรแกรมด้วยเครื่องมือ Jupyter Notebook จะมีนามสกุลไฟล์เป็น `.ipynb` และหากต้องการนำเอาไฟล์โปรแกรมไปประมวลกับเครื่องอื่นด้วยตัวแปลงภาษา Python ต้องสร้างไฟล์ให้เป็นไฟล์สคริปต์ `.py` ก่อน ถึงจะนำไปใช้กับเครื่องอื่นได้ ให้เลือกที่เมนู `File > Download as > Python (.py)` และบันทึกไว้ในตำแหน่งที่ต้องการ แต่หากต้องการนำเอาไฟล์โปรแกรมไปประมวลกับเครื่องอื่นด้วย Jupyter Notebook เมื่อนั้นก็ ให้เลือกที่เมนู `File > Download as > Notebook (.ipynb)`
- เลือกตำแหน่งไฟล์ที่ต้องการจัดเก็บ และเปลี่ยนชื่อไฟล์หากต้องการ จากนั้นคลิกปุ่ม `Save`
- สามารถแสดงผลลัพธ์โดยใช้คำสั่งผ่าน Command line จากไฟล์ที่มีนามสกุล `.py` ด้วยคำสั่ง

```
python script_name.py
```



รูป 2.8: ประมวลผลคำสั่งผ่านคีย์ลัดของ Jupyter Notebook

- เมื่อต้องการนำไฟล์จากเครื่องอื่นมาพัฒนาโปรแกรม ให้คลิกเมนู **Upload** เลือกไฟล์ที่ต้องการแล้ว คลิกปุ่ม **Open**



- เลือกไฟล์ที่ต้องการซึ่งจะต้องเป็นไฟล์ .ipynb เท่านั้น จากนั้นให้คลิกปุ่ม **Upload** เพื่อนำไฟล์เข้าสู่ Jupyter Notebook

#### 2.4.5 ประมวลผลคำสั่งผ่านคีย์ลัด

หากต้องการสี่่ให้โปรแกรมทำงานโดยไม่ใช้งานเมนู สามารถเรียกใช้งานคีย์ลัด (Shortcut key) ได้ ซึ่งโปรแกรม Jupyter Notebook ก็ได้อันวยความสะดวกในส่วนนี้ไว้ให้ เช่นกัน ผู้อ่านสามารถตรวจสอบคีย์ลัดได้จากเมนู **Help > Keyboard Shortcuts**

## 2.5 การแสดงผลและการรับข้อมูล

พื้นฐานอย่างหนึ่งที่ผู้เริ่มพัฒนาโปรแกรมจะต้องรู้คือ การแสดงข้อมูลออกทางจอภาพ และการรับข้อมูลผ่านทางคีย์บอร์ดเพื่อนำข้อมูลที่ป้อนเข้ามาไปประมวลผลต่อ ในส่วนนี้ผู้อ่านจะได้รู้จักกับฟังก์ชันที่ทำหน้าที่ทั้งสอง

### 2.5.1 การแสดงผลออกทางจอภาพ

ฟังก์ชัน `print()` ทำหน้าที่แสดงผลข้อมูลออกทางหน้าจอ ซึ่งเป็นฟังก์ชัน Built-in ที่ภาษาไพธอนได้จัดเตรียมไว้ให้ผู้ใช้งานเรียกใช้งานได้เลย มีรูปแบบการใช้งานได้ดังนี้

```
print(object, sep=' ', end='\n', file=sys.stdout,  
      flush=False)
```

`object` ออบเจ็คที่ต้องการแสดงผล มีได้มากกว่า 1 ออบเจ็ค

`sep` ส่วนที่ใช้แยกแต่ละออบเจ็คออกจากกัน ค่าปกติเป็นช่องว่าง

`end` การกำหนดแสดงผลออบเจ็ค ค่าปกติเป็นการขึ้นบรรทัดใหม่

`file` ออบเจ็คไฟล์ที่ต้องการแสดงผล

`flush` เมื่อดำเนินข้อมูลจากบัฟเฟอร์บันทึกลงไฟล์ ค่าปกติเป็น False

ตัวอย่างต่อไปนี้เป็นการใช้งานฟังก์ชัน `print()` ในการแสดงผลออกทางจอภาพ ผู้อ่านสามารถดูแนวทางการใช้งานโดยสังเขปได้ดังตาราง 2.2

#### ตัวอย่าง 2.1

การใช้งานฟังก์ชัน `print()`

```
1 print('Hello World')  
2 print('Sawasdee Jaa')  
3 print(2000+20+2)
```

Hello World

Sawasdee Jaa

2022



## ตัวอย่าง 2.2

การใช้งานฟังก์ชัน print()

```
1 s = 'Python is an easy programming language.'
2 print(s)
3 x = 5
4 y = 10
5 print('x is ', x, ',', 'y is ', y)
6 print('Calculation: xy is ', x * y)
```

Python is an easy programming language.  
x is 5 , y is 10  
Calculation: xy is 50



## ตัวอย่าง 2.3

การเรียกใช้งานฟังก์ชัน print() แสดงผลในรูปแบบต่าง ๆ

```
1 print('Hello Python', 'Hello Coding')
2 print(50)
3 print('Hello' + 'Python')
4 print('Hello' + ' ' + 'Python')
5 print()
6 print(5*10/2)
```

Hello Python Hello Coding  
50  
HelloPython  
Hello Python  
  
25.0



## 2.5.2 การรับข้อมูล

ฟังก์ชัน `input()` ทำหน้าที่รอรับข้อมูลที่จะถูกป้อนเข้ามาผ่านทางคีย์บอร์ดจากผู้ใช้งาน ไม่ว่าจะเป็นตัวเลขจำนวนเต็ม ตัวเลขจำนวนนทศนิยม หรือข้อความ (String) ซึ่งเป็นประเภทฟังก์ชัน Built-in อีกหนึ่งตัวที่เรียกใช้งานได้เลย แต่เมื่อป้อนข้อมูลไม่ว่าจะเป็นตัวเลขหรือตัวอักษร จะภาษาไทยตอนจะแปลงเป็นชนิดข้อมูลสตริงทั้งหมด (สำหรับการแปลงข้อมูลผู้อ่านจะได้เรียนรู้ในบทต่อไป) การเรียกใช้งานฟังก์ชัน `input()` แสดงดังต่อไปนี้

### ตัวอย่าง 2.4

การใช้งานฟังก์ชัน `input()`

```
1 name = input('Enter your name: ')
2 print('Hello', name)
3 print('Nice to meet you here')
```

```
Enter your name: John
Hello John
Nice to meet you here
```



### ตัวอย่าง 2.5

การใช้งานฟังก์ชัน `input()`

```
1 first_name = input('กรุณาป้อนชื่อของคุณ : ')
2 last_name = input('กรุณาป้อนนามสกุลของคุณ : ')
3 age = input('กรุณาป้อนอายุของคุณ : ')
4 print('-----')
5 print('ชื่อของคุณ คือ ', first_name)
6 print('นามสกุลของคุณ คือ ', last_name)
7 print('อายุของคุณ คือ ', age)
```

```
กรุณาป้อนชื่อของคุณ : ประหยด
กรุณาป้อนนามสกุลของคุณ : จันโอชิท
กรุณาป้อนอายุของคุณ : 61
-----
ชื่อของคุณ คือ ประหยด
นามสกุลของคุณ คือ จันโอชิท
อายุของคุณ คือ 61
```



คำสั่ง	คำอธิบาย
<code>print('...')</code> หรือ <code>print(...)</code>	ให้แสดงผลชนิดข้อมูลอักขระหรือสตริง
<code>print(50)</code>	ให้แสดงผลชนิดข้อมูลจำนวนเต็ม
<code>print(var1, var2, ..., var_n)</code>	ให้แสดงผลค่าข้อมูลที่เก็บอยู่ในตัวแปร
<code>print('...' + '...')</code>	ต้องการให้แสดงผลการเชื่อมข้อมูลชนิดตัวอักขระหรือสตริงต่อกัน
<code>print('...', '...')</code>	ต้องการให้แสดงผลการเชื่อมข้อมูลชนิดตัวอักขระหรือสตริงต่อกันและเว้นวรรค
<code>print('...' + ' ' + '...')</code>	ต้องการให้แสดงผลการเชื่อมข้อมูลชนิดตัวอักขระหรือสตริงต่อกันและเว้นวรรค
<code>print('...', var)</code>	ต้องการให้แสดงผลข้อความและค่าข้อมูลที่เก็บอยู่ในตัวแปร
<code>print()</code>	ต้องการให้เว้นบรรทัดใหม่
<code>print(5+10/2)</code>	ต้องการแสดงผลลัพธ์จากการดำเนินการทำคณิตศาสตร์

ตาราง 2.2: การเขียนคำสั่งแสดงข้อมูลด้วยฟังก์ชัน `print()`

## 2.6 การเขียนคำอธิบายโปรแกรม

เมื่อพัฒนาโปรแกรมไปได้สักระยะหนึ่งและโปรแกรมมีขนาดใหญ่ขึ้น สิ่งที่ตามมาคือไม่ทราบว่าโปรแกรมทำงานอย่างไรบ้าง มีการเรียกใช้งานและส่งค่าข้อมูลระหว่างกันอย่างไร ดังนั้นเพื่อเตือนความจำระหว่างการเขียนโปรแกรม เราจึงต้องเขียนคำอธิบายการทำงานในแต่ละส่วนหรือในแต่ละบรรทัดภายในโปรแกรม (Comment) เพราะจะช่วยให้แก้ไขเมื่อเกิดข้อผิดพลาดขึ้นได้ง่ายและรวดเร็ว หากพัฒนาโปรแกรมร่วมกันหลายคนยิ่งมีประโยชน์ต่อผู้อื่นเป็นอย่างมาก การเขียนคำอธิบายทำได้ 2 วิธีด้วยกันคือ

1. การเขียนคำอธิบายเพียงบรรทัดเดียวให้ใช้เครื่องหมาย **#**
2. การเขียนคำอธิบายแบบหลายบรรทัดให้ใช้เครื่องหมาย triple quotes

( ' ' ' ... ' ' ') หรือ ( " " " ... " " ")

### ตัวอย่าง 2.6

การเขียนคำอธิบายคำสั่งโปรแกรม (Comment) โดยใช้เครื่องหมาย **#**

```
1 print('Hello Python', 'Hello Python programming')
2 print(50) # แสดงตัวเลข 50 ออกทางหน้าจอ
3 print(257.451 + 124.12) # แสดงผลลัพธ์จากการบวก
```

Hello Python Hello Python programming  
50  
381.571



### ตัวอย่าง 2.7

การเขียนคำอธิบายคำสั่งโปรแกรม (Comment) แบบหลายบรรทัด

```
1 '''
2 1. แสดงการใช้ฟังก์ชัน input() รับข้อมูลผ่านทางคีย์บอร์ด
3 2. แสดงการใช้ฟังก์ชัน print() แสดงผลข้อมูล\n',
4 '''
5 msg = input('กรุณาป้อนข้อความ : ')
6 print('ข้อความที่คุณป้อน = ', msg)
```

กรุณาป้อนข้อความ : Python Programming  
ข้อความที่คุณป้อน = Python Programming



จากตัวอย่างการเขียนตัวอย่าง 2.6 และตัวอย่าง 2.7 ผู้อ่านจะสังเกตเห็นว่า ข้อความที่อยู่ด้านหลังเครื่องหมาย `#` หรือที่อยู่ภายใต้เครื่องหมาย triple-quotes (`''' ... '''`) ซึ่งเป็นข้อความคำอธิบายการทำงานของคำสั่งโปรแกรม จะไม่ถูกนับมาแสดงผลเมื่อเราสั่งให้โปรแกรมประมวลผลไฟรอนสคริปต์

## สรุปก่อนจบบท

ในบทนี้เรายังได้เรียนรู้ถึงการติดตั้งตัวแปลภาษาไฟรอน การใช้งานไฟรอนสคริปต์ การใช้งานเครื่องมือ Jupyter Notebook และยังได้รู้จักกับฟังก์ชัน `print()` และ `input()` ที่นำมาแสดงผลข้อมูลและรับข้อมูล ซึ่งเป็นฟังก์ชัน Built-in รวมไปถึงการเขียนคำอธิบายคำสั่งโปรแกรม

## แบบฝึกหัด

1. จงยกตัวอย่างเครื่องมือที่สามารถนำมาพัฒนาโปรแกรมด้วยภาษาไฟรอนได้อย่างน้อย 3 เครื่องมือ
2. จงเขียนโปรแกรมให้แสดงผลข้อมูล: ชื่อ, ที่อยู่, ประวัติการศึกษา, อาชีพ, สีที่ชอบ และ สีที่ไม่ชอบ โดยให้รับข้อมูลเหล่านี้ผ่านทางคีย์บอร์ด
3. จงเขียนโปรแกรมสั้น ๆ (อะเรกีಡี้) พร้อมทั้งเขียนคำอธิบายการทำงานของโปรแกรมนั้น (comment) ทั้งแบบบรรทัดเดียวและหลายบรรทัด



## บทที่ 3

# ตัวแปรและชนิดข้อมูลในภาษาโปรแกรม ไฟรอน

การเขียนโปรแกรมภาษาโปรแกรมไฟรอน สิ่งแรกที่ผู้เริ่มพัฒนาโปรแกรมจำเป็นต้องรู้คือ ตัวแปรและชนิดข้อมูล ซึ่งภาษาโปรแกรมไฟรอนได้จัดเตรียมชนิดข้อมูลไว้ให้ใช้งานอย่างมากมาย ในบทนี้เราจะรู้จักกับวิธีการประกาศสร้างตัวแปร การกำหนดค่าให้กับตัวแปรและชนิดข้อมูลพื้นฐานต่าง ๆ ได้แก่ ชนิดข้อมูลจำนวนเต็มที่มีทั้งตัวเลขจำนวนเต็มบวก จำนวนเต็มลบ ชนิดข้อมูลจำนวนทศนิยม ชนิดข้อมูลค่าความจริงที่ให้ผลลัพธ์เป็นค่าจริง (**True**) หรือเป็นค่าเท็จ (**False**) ชนิดข้อมูลสายอักขระหรือสตริง อีกทั้งยังจะได้รู้จักกับชนิดข้อมูลจำนวนเชิงซ้อน

### 3.1 การตั้งชื่อตัวแปรในภาษาโปรแกรมไฟรอน

ตัวแปร (Variable) คือ ชื่อที่ตั้งขึ้นมาใช้สำหรับเก็บค่าหรือชนิดข้อมูลต่าง ๆ ที่ต้องการ แล้วนำตัวแปรที่ตั้งขึ้นมาไปเขียนเป็นคำสั่งโปรแกรมสำหรับประมวลผล ตัวแปรที่ตั้งขึ้นมาใช้งานนั้นสามารถเปลี่ยนค่าข้อมูลที่เก็บอยู่ในขณะที่โปรแกรมทำการประมวลผลได้ กฎการตั้งชื่อตัวแปรของภาษาโปรแกรมไฟรอนมีดังนี้

- การตั้งชื่อตัวแปรด้วยตัวอักษรพิมพ์เล็กและตัวอักษรพิมพ์ใหญ่ ภาษาโปรแกรมไฟรอนจะถือว่า เป็นคนละชื่อตัวแปร
- ชื่อตัวแปรต้องขึ้นต้นด้วยตัวอักษรภาษาอังกฤษ (A-Z, a-z) เท่านั้น และตามด้วยตัวเลข เช่น Animal, animal\_01, big10, book10group เป็นต้น
- ชื่อตัวแปรต้องไม่มีช่องว่าง จุด และสัญลักษณ์พิเศษ ยกเว้นเครื่องหมาย underscore(\_) เช่น msg\_1 หรือ \_msg\_1
- ควรตั้งชื่อตัวแปรให้สื่อความหมายกับค่าข้อมูลที่จะเก็บ เพราะทำให้อ่านและเข้าใจได้ง่าย เช่น Tax ใช้เก็บค่าภาษี, studentName หรือ st\_name ใช้เก็บชื่อนักเรียน, passwd ใช้เก็บรหัสผ่าน เป็นต้น
- สัญลักษณ์ต่อไปนี้ห้ามนำมาใช้ตั้งชื่อตัวแปร

%^&(\_=\_+-\*\/\}{!\$#

- เมื่อต้องการสมคำตั้งชื่อตัวแปรควรใช้เครื่องหมาย underscore เชื่อม หรือกำหนดให้คำแรกใช้ตัวอักษรภาษาอังกฤษพิมพ์เล็กขึ้นต้น และคำที่สองให้ขึ้นต้นด้วยตัวใหญ่ เช่น Table\_name หรือ tableName เป็นต้น
- การตั้งชื่อตัวแปรต้องไม่ซ้ำกับคำล่วง (Reserved keywords) ซึ่งมีทั้งหมด 33 คำ เราสามารถตรวจสอบคำส่วนได้โดยการเพิ่มคำสั่ง

```
import keyword
```

ลงในส่วนการประกาศแล้วใช้คำสั่ง `print(keyword.kwlist)` มีดังต่อไปนี้

<code>and</code>	<code>as</code>	<code>assert</code>	<code>break</code>	<code>class</code>	<code>continue</code>
<code>def</code>	<code>del</code>	<code>elif</code>	<code>else</code>	<code>except</code>	<code>False</code>
<code>finally</code>	<code>for</code>	<code>from</code>	<code>global</code>	<code>if</code>	<code>import</code>
<code>in</code>	<code>is</code>	<code>lambda</code>	<code>None</code>	<code>nonlocal</code>	<code>not</code>
<code>or</code>	<code>pass</code>	<code>raise</code>	<code>return</code>	<code>True</code>	<code>try</code>
<code>while</code>	<code>with</code>	<code>yield</code>			

ในการตั้งชื่อตัวแปรขึ้นมาใช้งานควรตั้งชื่อให้สื่อความหมายกับชนิดข้อมูลที่ใช้จัดเก็บ เมื่อพัฒนาโปรแกรมไปสักระยะหนึ่งโปรแกรมมีขนาดใหญ่ขึ้น ตัวแปรมีจำนวนมากขึ้นอาจจะทำให้สับสนหากตั้งชื่อตัวแปรไม่สอดคล้อง ทำให้ต้องเสียเวลามากในการแก้ไขโปรแกรม

## 3.2 การประกาศตัวแปร

การประกาศสร้างตัวแปร (Declaration) คือ การสร้างตัวแปรสำหรับเก็บค่าข้อมูล เช่น ตัวเลข ตัวอักษร หรือข้อความ เป็นต้น ก่อนนำไปประมวลผลอีกรอบ ภาษาโปรแกรม Python ไม่จำเป็นต้องกำหนดชนิดข้อมูลให้กับตัวแปร ตัวแปรที่ประกาศสร้างขึ้นมาใช้งานจะกำหนดค่าข้อมูลไว้ หรืออาจสร้างตัวแปรขึ้นมาเป็นค่าว่างไว้ก่อนก็ได้ และจึงกำหนดค่าข้อมูลให้กับตัวแปรทีหลัง ตัวอย่างการสร้างตัวแปรจะเป็นดังต่อไปนี้

### ตัวอย่าง 3.1

#### การประกาศตัวแปร (Declaration)

```
1 msg = 'Python Programming' # ประกาศตัวแปร msg เป็นสตริง (String)
2 num = 2415 # ประกาศตัวแปร num เป็นชนิดข้อมูลจำนวนเต็ม (Integer)
3 score = 78.54 # ประกาศตัวแปร score เป็นชนิดข้อมูลทศนิยม (float)
4 grad = '' # ประกาศตัวแปร grad เก็บสตริงช่องว่าง
```

ชื่อตัวแปรจะอยู่ทางด้านซ้ายมือของเครื่องหมายเท่ากับ (=) ส่วนทางด้านขวา มีของเครื่องหมายเท่ากับคือ ชนิดข้อมูลที่กำหนดให้กับตัวแปร ถ้าเป็นชนิดข้อมูลสายอักขระหรือสตริง หรือเรียกว่า 'สตริง' จะอยู่ในเครื่องหมาย '...' หรือ ' ' ถ้ายังไม่กำหนดค่าให้กับตัวแปรหรือต้องการให้ตัวแปรเก็บค่าว่างให้ตั้งเป็น var = '' หรือถ้ายังไม่ได้กำหนดชนิดข้อมูลให้กับตัวแปรให้ใช้ **None** เช่น var = **None** ถ้าพิจยแต่ตั้งชื่อตัวแปรแล้วสั่งให้โปรแกรมทำงานจะทำให้เกิดการแจ้งเตือนข้อผิดพลาดขึ้น

ในภาษาโปรแกรมไพธอนเวอร์ชัน 3 สามารถสร้างตัวแปรเป็นภาษาไทยได้ แต่ถึงอย่างไรก็ตามเพื่อความเป็นมาตรฐานในการเขียนคำสั่งโปรแกรมและให้ง่ายต่อผู้ที่ต้องการนำโปรแกรมของเราไปพัฒนาต่อในอนาคต ดังนั้นจึงควรสร้างตัวแปรเป็นภาษาอังกฤษและตั้งชื่อให้สื่อความหมาย

การประกาศสร้างตัวแปรเป็นชื่อภาษาไทย แสดงดังตัวอย่างต่อไปนี้

### ตัวอย่าง 3.2

การเขียนคำสั่งโปรแกรมตั้งชื่อตัวแปรเป็นภาษาไทย

```
1 ปากกา = 'สีแดง'      # ประกาศตัวแปรชื่อ 'ปากกา' เป็นชนิดข้อมูลสตริง  
2 โทรศัพท์ = 'Samsung'    # ประกาศตัวแปรชื่อ 'โทรศัพท์' เป็นชนิดข้อมูลสตริง  
3 print('ปากกาสี = ', ปากกา)      # แสดงผลข้อมูลจากค่าตัวแปร 'ปากกา'  
4 print('โทรศัพท์ยี่ห้อ = ', โทรศัพท์)    # แสดงผลข้อมูลจากค่าตัวแปร 'โทรศัพท์'
```

ปากกาสี = สีแดง

โทรศัพท์ยี่ห้อ = Samsung



## 3.3 ตัวแปรชนิดค่าคงที่

ค่าคงที่ (Constant) คือ ตัวแปรที่สร้างขึ้นมาเก็บค่าข้อมูลที่ไม่มีการเปลี่ยนแปลงค่าในระหว่างการประมวลผล เช่น ค่า π มีค่าเท่ากับ 3.14 เราสามารถตั้งชื่อตัวแปรเป็นค่าคงที่ได้เท่ากับ pi = **3.14** หรือค่าภาษีมูลค่าเพิ่มค่าเท่ากับ 7% เราสามารถตั้งชื่อตัวแปรเป็นค่าคงที่ได้เท่ากับ vat = **0.07** เป็นต้น ดังแสดงตัวอย่างต่อไปนี้

### ตัวอย่าง 3.3

การประก取ตัวแปรชนิดค่าคงที่ให้พื้นที่วงกลม ด้วยกำหนดค่า pi = 3.14

```
1 pi = 3.14          # สร้างตัวแปร pi เป็นค่าคงที่
2 area = pi * 5 * 5 # คูณค่าตัวแปร pi เพื่อหาพื้นที่วงกลม
3 circumference = 2 * pi * 10 # คูณค่าตัวแปร pi เพื่อหาเส้นรอบวงกลม
4 print('พื้นที่วงกลม = ', area) # แสดงผลลัพธ์ตัวแปร area
5 print('เส้นรอบวงกลม = ', circumference) # แสดงผลลัพธ์ตัวแปร circumference
```

พื้นที่วงกลม = 78.5  
เส้นรอบวงกลม = 62.800000000000004



## 3.4 ชนิดข้อมูลจำนวนเต็ม

ชนิดข้อมูลจำนวนเต็ม (Integers) ประกอบไปด้วยจำนวนเต็มบวก (Positive Integers) เช่น **10, 15, 255** จำนวนเต็มลบ (Negative Integers) เช่น **-14, -2** และ ศูนย์ (Zero) จำนวนเต็มเหล่านี้สามารถเขียนในรูปเลขฐานสอง (Binary) เลขฐานแปด (Octal) และเลขฐานสิบ (Hexadecimal) จำนวนเต็มเลขฐานสิบประกอบด้วยตัวเลข **0-9** เลขฐานสองประกอบด้วยตัวเลขสองตัว ได้แก่ **0** และ **1** เลขฐานแปดประกอบด้วยตัวเลขแปดตัว ได้แก่ **0-7** และเลขฐานสิบหกประกอบด้วยตัวเลข สิบหกตัว ได้แก่ **0-9** และตัวอักษรอีกหก ตัว ได้แก่ **A-F** เพื่อความสะดวกในการแทนค่า **10-15** ซึ่งเป็นเลขสองหลัก

ในภาษาโปรแกรมไม่ได้กำหนดความยาวของชนิดข้อมูลจำนวนเต็ม นั่นหมายความว่าผู้เขียนโปรแกรมสามารถประกอบตัวแปรเก็บชนิดข้อมูลจำนวนเต็มให้มีความยาวเท่าไหร่ก็ได้ แต่ก็มีผลกระทบต่อการประมวลผลหากเครื่องคอมพิวเตอร์มีหน่วยความจำไม่เพียงพอ

### ตัวอย่าง 3.4

การดำเนินการกับชนิดข้อมูลจำนวนเต็มที่ความยาวไม่จำกัด

```
1 num1 = 546135478945123456214 # สร้างตัวแปร num1 เป็นชนิดข้อมูลจำนวนเต็ม
2 num2 = 14567489412457845679 # สร้างตัวแปร num2 เป็นชนิดข้อมูลจำนวนเต็ม
3 print('ผลบวกของ num1 กับ num2 = ', num1 + num2) # แสดงผลลัพธ์ num1 + num2
4 print('ผลลบของ num1 กับ num2 = ', num1 - num2) # แสดงผลลัพธ์ num1 - num2
```

ผลบวกของ num1 กับ num2 = 560702968357581301893  
ผลลบของ num1 กับ num2 = 531567989532665610535



การดำเนินการกับชนิดข้อมูลจำนวนเต็มเลขฐานสอง ฐานแปด และฐานสิบหก มีวิธีการดำเนินการดังต่อไปนี้

- ถ้าต้องการกำหนดเลขจำนวนเต็มเป็นเลขฐานสองให้ใช้สัญลักษณ์ **0b** นำหน้า เช่น

**0b1001011 + 0b1101101**

- ถ้าต้องการกำหนดเลขจำนวนเต็มเป็นเลขฐานแปดให้ใช้สัญลักษณ์ **0o** นำหน้า เช่น

**0o12451 + 0o54125**

- ถ้าต้องการกำหนดเลขจำนวนเต็มเป็นเลขฐานสิบหกให้ใช้สัญลักษณ์ **0x** นำหน้า เช่น

**0x76ade + 0x97c7a**

นอกจากนี้สามารถเรียกใช้ฟังก์ชันแปลงเลขฐานให้เป็นเลขฐานอื่น ๆ ได้โดยรูปแบบการใช้งานคือ ชื่อฟังก์ชัน(ตัวแปร)

- ฟังก์ชัน **int()** เป็นฟังก์ชันแปลงเลขฐานที่ต้องการให้เป็นเลขฐานสิบ
- ฟังก์ชัน **bin()** เป็นฟังก์ชันแปลงเลขฐานที่ต้องการให้เป็นเลขฐานสอง
- ฟังก์ชัน **oct()** เป็นฟังก์ชันแปลงเลขฐานที่ต้องการให้เป็นเลขฐานแปด
- ฟังก์ชัน **hex()** เป็นฟังก์ชันแปลงเลขฐานที่ต้องการให้เป็นเลขฐานสิบหก

### ตัวอย่าง 3.5

การแปลงเลขฐานสิบเป็นเลขฐานสอง ฐานแปด และฐานสิบหก

```
1 x = 6749 # สร้างตัวแปร x เก็บชนิดข้อมูลจำนวนเต็มเลขฐานสิบ
2 print('แปลงเลขฐานสิบเป็นฐานสอง = ', bin(x)) # ฟังก์ชันแปลงเลขฐานสิบเป็นฐานสอง
3 print('แปลงเลขฐานสิบเป็นฐานแปด = ', oct(x)) # ฟังก์ชันแปลงเลขฐานสิบเป็นฐานแปด
4 print('แปลงเลขฐานสิบเป็นฐานสิบหก = ', hex(x)) # ฟังก์ชันแปลงเลขฐานสิบเป็นฐานสิบหก
```

แปลงเลขฐานสิบเป็นฐานสอง = **0b1101001011101**

แปลงเลขฐานสิบเป็นฐานแปด = **0o15135**

แปลงเลขฐานสิบเป็นฐานสิบหก = **0x1a5d**



### ตัวอย่าง 3.6

การแปลงเลขฐานสอง ฐานแปด และฐานสิบหก เป็นเลขฐานสิบ

```
1 a = 0b100111011001 # สร้างตัวแปร a เก็บชนิดข้อมูลจำนวนเต็มเลขฐานสอง
2 b = 0o6453 # สร้างตัวแปร b เก็บชนิดข้อมูลจำนวนเต็มเลขฐานแปด
3 c = 0xAC41 # สร้างตัวแปร c เก็บชนิดข้อมูลจำนวนเต็มเลขฐานสิบหก
4 print('แปลงเลขฐานสองเป็นฐานสิบ = ', int(a)) # แสดงผลลัพธ์การแปลงเลขฐาน
5 print('แปลงเลขฐานแปดเป็นฐานสิบ = ', int(b))
6 print('แปลงเลขฐานสิบหกเป็นฐานสิบ = ', int(c))
```

แปลงเลขฐานสองเป็นฐานสิบ = 2521  
แปลงเลขฐานแปดเป็นฐานสิบ = 3371  
แปลงเลขฐานสิบหกเป็นฐานสิบ = 44097



### ตัวอย่าง 3.7

การบวกเลขฐานสองและการแปลงผลลัพธ์จากเลขฐานสิบ เป็นเลขฐานสอง ฐานแปด และฐานสิบหก

```
1 a = 0b1001011; b = 0b1101101 # สร้างตัวแปร a และ b
2 c = a + b # บวกค่าตัวแปร a และ b
3 print('ผลบวกของ a กับ b = ', c) # แสดงผลลัพธ์ตัวแปร C
4 print('แปลงผลลัพธ์จากผลบวก a กับ b เป็นเลขฐานสอง = ', bin(c)) # แสดงผลลัพธ์
5 print('แปลงผลลัพธ์จากผลบวก a กับ b เป็นเลขฐานแปด = ', oct(c))
6 print('แปลงผลลัพธ์จากผลบวก a กับ b เป็นเลขฐานสิบหก = ', hex(c))
```

ผลบวกของ a กับ b = 184  
แปลงผลลัพธ์จากผลบวก a กับ b เป็นเลขฐานสอง = 0b10111000  
แปลงผลลัพธ์จากผลบวก a กับ b เป็นเลขฐานแปด = 0o270  
แปลงผลลัพธ์จากผลบวก a กับ b เป็นเลขฐานสิบหก = 0xb8



## 3.5 ชนิดข้อมูลจำนวนทศนิยม

ชนิดข้อมูลประเภทจำนวนทศนิยม (Float) ประกอบด้วยตัวเลข 2 ส่วน โดยมีเครื่องหมายจุด (.) คั่นระหว่างตัวเลขที่อยู่ด้านหน้าและด้านหลัง ตัวเลขด้านหน้าเป็นตัวเลขจำนวนเต็ม และตัวเลขที่อยู่ด้านหลังจุดเรียกว่า ทศนิยม การเขียนเลขทศนิยมสามารถเขียนได้สองแบบคือ แบบแรก เช่น 15.5, 20.451,

4.5134000000 เป็นต้น และแบบที่สอง เช่น

$$4.517458E - 2 = 4.517458 \times 10^{-2} = 0.04517458$$

หรือ

$$1.46547e4 = 1.465473 \times 10^4 = 14654.73$$

เป็นต้น ในแบบที่สองจะสังเกตเห็นว่ามีตัวอักษร E และ e ซึ่งเป็นการเขียนในรูปแบบของเลขยกกำลังสิบ (Exponential Form)

### ตัวอย่าง 3.8

การแสดงผลการดำเนินการบวก ลบ กับชนิดข้อมูลจำนวนทศนิยม

```
1 # สร้างตัวแปร float
2 a = 341.451E-3
3 b = 251.147e-2
4 x = 10.5
5 y = 17.5
6 d = a - b # ลบแล้วเก็บผลลัพธ์ไว้ที่ตัวแปร d
7 z = x + y # ลบแล้วเก็บผลลัพธ์ไว้ที่ตัวแปร z
8 print('ค่าตัวเลขจำนวนทศนิยมของ b = ', b) # แสดงผลลัพธ์ตัวแปร b
9 print('ผลลัพธ์จากการลบ a กับ b = ', d) # แสดงผลลัพธ์ตัวแปร d
10 print('ผลลัพธ์จากการบวก x กับ y = ', z) # แสดงผลลัพธ์ตัวแปร z
```

ค่าตัวเลขจำนวนทศนิยมของ  $b = 2.51147$

ผลลัพธ์จากการลบ  $a$  กับ  $b = -2.170019$

ผลลัพธ์จากการบวก  $x$  กับ  $y = 28.0$



ในกรณีที่ต้องการแปลงชนิดข้อมูลจำนวนเต็ม ให้เป็นชนิดข้อมูลจำนวนทศนิยม ให้ใช้ฟังก์ชัน `float()` มีวิธีการเรียกใช้งานแสดงดังตัวอย่างที่ 3.8

### ตัวอย่าง 3.9

การแปลงชนิดข้อมูลอักขระหรือสตริง เป็นชนิดข้อมูลจำนวนทศนิยม

```
1 a = 56; b = 55 # สร้างตัวแปร a และ b เก็บชนิดข้อมูลจำนวนเต็ม
2 print('ผลลัพธ์จากการแปลงค่าจากตัวแปร a = ', float(a)) # แปลงค่า a เป็น float
3 print('ผลลัพธ์จากการแปลงค่าจากตัวแปร b = ', float(b)) # แปลงค่า b เป็น float
4 print('ผลลัพธ์จากการบวกค่าของ a + b = ', float(a) + float(b)) # นำ a + b
5 print('ผลลัพธ์จากการคูณค่าของ a * b = ', float(a) * float(b)) # นำ a * b
```

ผลลัพธ์จากการแปลงค่าจากตัวแปร a = 56.8  
ผลลัพธ์จากการแปลงค่าจากตัวแปร b = 55.8  
ผลลัพธ์จากการบวกค่าของ a + b = 111.0  
ผลลัพธ์จากการคูณค่าของ a \* b = 3080.0



## 3.6 ชนิดข้อมูลจำนวนเชิงซ้อน

จำนวนเชิงซ้อน (Complex) ได้ถูกนำมาใช้งานในสาขาต่าง ๆ เช่น Electrical Engineering, Fluid Dynamics, Quantum Mechanics, Computer Graphics, Dynamic Systems และสาขาอื่น ๆ อีกด้วยเฉพาะอย่างยิ่งในด้านวิทยาศาสตร์และวิศวกรรม จำนวนเชิงซ้อนจะเขียนอยู่ในรูปของคู่  $x + yi$  เช่น  $5 + 2i$ ,  $30 - 6i$  เป็นต้น โดยเรียก  $x$  เป็นจำนวนจริง (Real Part) และ  $y$  เป็นส่วนจินตภาพ (Imaginary Part) ภาษาโปรแกรมไพธอนจะใช้ตัวอักษร  $j$  แทนตัวอักษร  $i$

การใช้ตัวดำเนินการกับชนิดข้อมูลจำนวนเชิงซ้อนทำได้ปกติเหมือนกับชนิดข้อมูลจำนวนเต็มและชนิดข้อมูลทศนิยม แสดงดังตัวอย่างต่อไปนี้

### ตัวอย่าง 3.10

การใช้ตัวดำเนินการบวก ลบ กับชนิดข้อมูลเชิงซ้อน

```
1 a = 20 + 5j; b = 15 + 6j; c = 6 - 7j # สร้างตัวแปรเป็นชนิดข้อมูลเชิงซ้อน
2 x = a + b # บวกค่าตัวแปร a กับ b
3 y = b * c # คูณค่าตัวแปร b กับ c
4 print('ผลลัพธ์จากการบวกระหว่าง a + b = ', x) # แสดงผลลัพธ์ตัวแปร x
5 print('ผลลัพธ์จากการคูณระหว่าง b * c = ', y) # แสดงผลลัพธ์ตัวแปร y
```

ผลลัพธ์จากการบวกระหว่าง a + b = (35+11j)  
ผลลัพธ์จากการคูณระหว่าง b \* c = (132-69j)



### 3.7 ชนิดข้อมูลตรรกะ

ชนิดข้อมูลตรรกะหรือชนิดข้อมูลค่าความจริง (Boolean) ให้ผลลัพธ์เพียงสองค่า คือ ค่าจริง (**True**) หรือค่าเท็จ (**False**) อย่างโดยอย่างหนึ่ง เราสามารถกำหนดค่า **True** หรือ **False** ให้กับตัวแปรเพื่อนำไปเปรียบเทียบได้เลย

#### ตัวอย่าง 3.11

การเขียนคำสั่งโปรแกรมใช้งานชนิดข้อมูลตรรกะ

```
1 x = True; y = False; z = True # สร้างตัวแปรเป็นชนิดข้อมูลตรรกะ
2 print('ผลการเปรียบเทียบ x == y หรือไม่ = ', x == y) # แสดงผลการเปรียบเทียบ x, y
3 print('ผลการเปรียบเทียบ y == z หรือไม่ = ', y == z) # แสดงผลการเปรียบเทียบ y, z
4 print('ผลการเปรียบเทียบ z == True หรือไม่ = ', z == True) # แสดงผลการเปรียบเทียบ z
```

ผลการเปรียบเทียบ x == y หรือไม่ = False

ผลการเปรียบเทียบ y == z หรือไม่ = False

ผลการเปรียบเทียบ z == True หรือไม่ = True



สามารถเรียกใช้งานฟังก์ชัน **bool()** ตรวจสอบค่าข้อมูลในตัวแปรที่สร้างขึ้นมาใช้งานว่าได้เก็บค่าข้อมูลไว้หรือไม่ ถ้าตัวแปรได้มีการกำหนดค่าข้อมูลไว้จะได้ผลลัพธ์เป็น **True** แต่ถ้าตัวแปรได้ไม่มีการกำหนดค่าหรือกำหนดค่าเป็น **0**, **None** หรือ **''** จะให้ผลลัพธ์เป็น **False** ถ้าตัวแปรประกาศเป็นช่องว่าง **''** ผลลัพธ์ที่ได้เป็น **True** พิจารณาการใช้งานฟังก์ชัน **bool()** จากตัวอย่าง 3.11

#### ตัวอย่าง 3.12

การเรียกใช้งานฟังก์ชัน **bool()** ตรวจสอบค่าข้อมูลในตัวแปร

```
1 b = ''; c = None; n = 245
2 print('มีข้อมูลอยู่ในตัวแปร b หรือไม่ = ', bool(b)) # แสดงผลตรวจสอบค่าตัวแปร b
3 print('มีข้อมูลอยู่ในตัวแปร c หรือไม่ = ', bool(c)) # แสดงผลตรวจสอบค่าตัวแปร c
4 print('มีข้อมูลอยู่ในตัวแปร n หรือไม่ = ', bool(n)) # แสดงผลตรวจสอบค่าตัวแปร n
5 print('มีข้อมูลอยู่ในตัวแปร num หรือไม่ = ', bool(num)) # แสดงผลตรวจสอบค่าตัวแปร num
```

มีข้อมูลอยู่ในตัวแปร b หรือไม่ = False

มีข้อมูลอยู่ในตัวแปร c หรือไม่ = True

มีข้อมูลอยู่ในตัวแปร n หรือไม่ = False

มีข้อมูลอยู่ในตัวแปร num2 หรือไม่ = True



## 3.8 ชนิดข้อมูลสายอักขระหรือสตริง

ชนิดข้อมูลสายอักขระหรือสตริง (String) หรือเรียกสั้น ๆ ว่า ชนิดข้อมูลสตริง เป็นการนำตัวอักษรหลาย ๆ ตัวมาเรียงต่อกันเป็นข้อความหรือประโยค การประกาศชนิดข้อมูลนี้ขึ้นมาใช้งานจะอยู่ในเครื่องหมาย Single Quote ('...') หรือ Double Quote ("...") ให้เลือกใช้รูปแบบใดรูปแบบหนึ่ง แสดงดังตัวอย่างต่อไปนี้

### ตัวอย่าง 3.13

การประกาศสร้างตัวแปรชนิดข้อมูลสตริงขึ้นมาใช้งาน

```
1 msg = 'Python Programming' # สร้างตัวแปร msg เก็บชนิดข้อมูลสตริงใช้เครื่องหมาย ''  
2 msg1 = "Language" # สร้างตัวแปร msg1 เก็บชนิดข้อมูลสตริงใช้เครื่องหมาย ''  
3 print(msg) # แสดงผลลัพธ์ตัวแปร msg  
4 print(msg1) # แสดงผลลัพธ์ตัวแปร msg1  
5 print(msg, msg1) # แสดงผลลัพธ์ตัวแปร msg และ msg1
```

Python Programming  
Language  
Python Programming Language



## 3.9 การแปลงชนิดข้อมูล

ในการพัฒนาโปรแกรมบางครั้งเราไม่มีความจำเป็นต้องแปลงชนิดข้อมูล (Data Type Conversion) ก่อน นำไปดำเนินการ หากไม่มีการแปลงชนิดข้อมูลก่อนจะทำให้เกิดข้อผิดพลาดขึ้นมาได้ ยกตัวอย่างการป้อนข้อมูลผ่านทางแป้นพิมพ์ด้วยฟังก์ชัน `input()` ถึงแม้จะป้อนข้อมูลเป็นตัวเลขจำนวนเต็มหรือจำนวนทศนิยม เมื่อนำไปดำเนินการกับชนิดข้อมูลจำนวนเต็มหรือชนิดข้อมูลทศนิยมจะทำให้เกิดการแจ้งเตือนข้อผิดพลาดขึ้น เนื่องจากตัวเลขที่ป้อนเข้ามานั้นเป็นชนิดข้อมูลสตริง แสดงดังตัวอย่างต่อไปนี้

### ตัวอย่าง 3.14

การแจ้งเตือนข้อผิดพลาดเมื่อนำชนิดข้อมูลสตริงดำเนินการกับชนิดข้อมูลจำนวนเต็ม

```
1 num = input('Enter number = ') # กำหนดให้ป้อนข้อมูลผ่านคีย์บอร์ด
2 print(50 + num) # แสดงผลลัพธ์การบวก 50 กับค่าตัวแปร num
```

```
Enter number = 60
TypeError Traceback (most recent call last)
<ipython-input-1-43111086d092> in <module>
  1 num = input('Enter number = ') # กำหนดให้ป้อนข้อมูลผ่านคีย์บอร์ด ---->
  2 print(50 + num) # แสดงผลลัพธ์การบวก 50 กับค่าตัวแปร num
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

จากตัวอย่างเห็นได้ว่าเกิดการแจ้งเตือนข้อผิดพลาดขึ้น เนื่องจากตัวแปร num เป็นชนิดข้อมูลสตริง ซึ่งไม่สามารถนำไปดำเนินการบวกกับ **50** ในบรรทัดที่ 2 ได้ เพราะเป็นชนิดข้อมูลจำนวนเต็ม ดังนั้นจึงต้องแปลงชนิดข้อมูลก่อนโดยการเรียกใช้งานฟังก์ชันแปลงค่าชนิดข้อมูลก่อนนำไปดำเนินการต่อ

- ถ้าต้องการแปลงเป็นชนิดข้อมูลจำนวนเต็มให้เรียกใช้งานใช้ฟังก์ชัน **int()**
- ถ้าต้องการแปลงเป็นชนิดข้อมูลจำนวนทศนิยมให้เรียกใช้งานใช้ฟังก์ชัน **float()**
- ถ้าต้องการแปลงเป็นชนิดข้อมูลจำนวนเชิงซ้อนให้เรียกใช้งานใช้ฟังก์ชัน **complex()**

### ตัวอย่าง 3.15

การแปลงชนิดข้อมูลสตริงเป็นชนิดข้อมูลจำนวนเต็ม

```
1 num = int(input('กรุณาป้อนตัวเลขของคุณ : ')) # รับข้อมูลผ่านคีย์บอร์ดพร้อมแปลงข้อมูล
2 x = '60' # สร้างตัวแปร x เป็นชนิดข้อมูลสตริง
3 y = int(x) + num # แปลงค่าตัวแปร x เป็นชนิดข้อมูลจำนวนเต็มแล้วบวกกับค่าตัวแปร num
4 print('ผลลัพธ์จากการบวก x + num = ', y) # แสดงผลลัพธ์จากการบวก y
```

```
กรุณาป้อนตัวเลขของคุณ : 50
ผลลัพธ์จากการบวก x + num = 110
```

### ตัวอย่าง 3.16

การแปลงชนิดข้อมูลสตริงเป็นชนิดข้อมูลจำนวนทศนิยม

```
1 flo = float(input('กรุณาป้อนตัวเลขของคุณ : ')) # รับข้อมูลผ่านคีย์บอร์ดพร้อมแปลงข้อมูล
2 x = '152.124' # สร้างตัวแปร x เป็นชนิดข้อมูลสตริง
3 y = float(x) + flo # แปลงค่าตัวแปร x เป็นชนิดข้อมูลจำนวนทศนิยมบวกกับค่าตัวแปร flo
4 print('ผลลัพธ์จากการบวก x + flo = ', y) # แสดงผลลัพธ์จากค่าตัวแปร y
```

กรุณาป้อนตัวเลขของคุณ : 345.21  
ผลลัพธ์จากการบวก x + flo = 497.33399999999995



### ตัวอย่าง 3.17

การแปลงชนิดข้อมูลสตริงเป็นชนิดข้อมูลจำนวนเชิงซ้อน

```
1 a = '20+5j'; b = '15+6j'; # สร้างตัวแปร a และ b เป็นชนิดข้อมูลเชิงซ้อน
2 x = complex(a) # แปลงค่าตัวแปร a เป็นชนิดข้อมูลเชิงซ้อน
3 y = complex(b) # แปลงค่าตัวแปร b เป็นชนิดข้อมูลเชิงซ้อน
4 z = complex(7,5) # แปลงค่าจำนวนจริงเป็นชนิดข้อมูลเชิงซ้อน
5 print('ผลลัพธ์การแปลงชนิดข้อมูลจำนวนเชิงซ้อนจากตัวแปร a = ', x) # แสดงผลลัพธ์ x
6 print('ผลลัพธ์การแปลงชนิดข้อมูลจำนวนเชิงซ้อนจากตัวแปร b = ', y) # แสดงผลลัพธ์ y
7 print('ค่าจำนวนจริงของ c = ', x.real, 'และค่าจินตภพของ c = ', x.imag)
8 # แสดงผลลัพธ์การใช้ฟังก์ชัน real() และ imag()
9 print('ผลลัพธ์การแปลงเครื่องหมายตัวดำเนินการของตัวแปร y = ', y.conjugate())
10 # แสดงผลลัพธ์การใช้ฟังก์ชัน conjugate()
11 print('ผลลัพธ์การสร้างคู่อันดับจากตัวแปร z = ', z) # แสดงผลลัพธ์ z
```

ผลลัพธ์การแปลงชนิดข้อมูลจำนวนเชิงซ้อนจากตัวแปร a = (20+5j)  
ผลลัพธ์การแปลงชนิดข้อมูลจำนวนเชิงซ้อนจากตัวแปร b = (15+6j)  
ค่าจำนวนจริงของ c = 20.0 และค่าจินตภพของ c = 5.0  
ผลลัพธ์การแปลงเครื่องหมายตัวดำเนินการของตัวแปร y = (15-6j)  
ผลลัพธ์การสร้างคู่อันดับจากตัวแปร z = (7+5j)



### ตัวอย่าง 3.18

การแปลงชนิดข้อมูลสตริงเป็นชนิดข้อมูลเชิงซ้อน กรณีเกิดการแจ้งเตือนข้อผิดพลาด

```
1 c = '25 + 9j'; # สร้างตัวแปรเก็บชนิดข้อมูลสตริงเก็บไว้ในตัวแปร c ที่มีช่องว่าง
2 print(complex(c)) # แสดงผลลัพธ์ตัวแปร c
```

```
ValueError      Traceback (most recent call last)
<ipython-input-2-e8d7c7a0a2cf> in <module>
1 c = '25+9j ; # สร้างตัวแปรเก็บชนิดข้อมูลสตริงเก็บไว้ในตัวแปร c ที่มีช่องว่าง ---->
2 print(complex(c)) # แสดงผลลัพธ์ตัวแปร c

ValueError: complex() arg is a malformed string
```



## 3.10 การกำหนดค่าให้กับตัวแปร

ตัวแปรที่ประกาศสร้างขึ้นมาใช้งานจะมีการกำหนดค่าให้ ในภาษาโปรแกรมไพธอนมีรูปแบบการกำหนดค่าให้กับตัวแปรหลากหลายวิธี เราสามารถศึกษาได้จากตัวอย่างดังต่อไปนี้

### ตัวอย่าง 3.19

การสร้างตัวแปรครั้งละหลาย ๆ ตัว และกำหนดเป็นชนิดข้อมูลเดียวกัน

```
1 num1 = num2 = num3 = 50 # สร้างตัวแปร num1, num2 และ num3 เป็นชนิดข้อมูลจำนวนเต็ม
2 print('ค่าข้อมูลในตัวแปร num1 = ', num1) # แสดงผลลัพธ์ตัวแปร num1
3 print('ค่าข้อมูลในตัวแปร num2 = ', num2) # แสดงผลลัพธ์ตัวแปร num2
4 print('ค่าข้อมูลในตัวแปร num3 = ', num3) # แสดงผลลัพธ์ตัวแปร num3
```

ค่าข้อมูลในตัวแปร num1 = 50

ค่าข้อมูลในตัวแปร num2 = 50

ค่าข้อมูลในตัวแปร num3 = 50



จากตัวอย่างในบรรทัดที่ 1 ได้ประกาศตัวแปร num1, num2 และ num3 พร้อมทั้งกำหนดค่าให้กับตัวแปรทั้ง 3 ตัว มีค่าเท่ากับ **50** และแสดงค่าข้อมูลที่เก็บอยู่ในตัวแปรทั้ง 3 ด้วยฟังก์ชัน **print()** ในบรรทัดที่ 2-4 ตามลำดับ

### ตัวอย่าง 3.20

การสร้างตัวแปรรึ่งละหลาย ๆ ตัว และกำหนดชนิดข้อมูลจำนวนเต็มและชนิดข้อมูลสตริง

```
1 num1, num2, msg = 50, 100, 'The world is beautiful.' n',
2 print('ค่าข้อมูลในตัวแปร num1 = ', num1)
3 print('ค่าข้อมูลในตัวแปร num2 = ', num2)
4 print('ค่าข้อมูลในตัวแปร msg = ', msg)
```

ค่าข้อมูลในตัวแปร num1 = 50  
ค่าข้อมูลในตัวแปร num2 = 100  
ค่าข้อมูลในตัวแปร msg = The world is beautiful.



จากตัวอย่างได้ประกาศสร้างตัวแปร num1, num2 และ msg แต่ละตัวจะถูกคืนด้วยเครื่องหมาย comma (,) และกำหนดเป็นชนิดข้อมูลจำนวนเต็มและชนิดข้อมูลสตริงมีค่าเท่ากับ 50, 100 และ 'The world is beautiful.' ตามลำดับ การประกาศตัวแปรเข่นนี้จะมีการเรียงลำดับตัวแปรในการเก็บค่าข้อมูลที่ได้กำหนดไว้ สังเกตได้จากเมื่อใช้ฟังก์ชัน print() แสดงผลค่าข้อมูลที่เก็บอยู่ในแต่ละตัวแปร

### ตัวอย่าง 3.21

การสร้างตัวแปรรึ่งละหลาย ๆ ตัว แต่แยกเก็บชนิดข้อมูลอยู่ในบรรทัดเดียวกัน

```
1 x = 200; y = 300; msg = 'Python programming language'
2 print('ค่าข้อมูลในตัวแปร x = ', x)
3 print('ค่าข้อมูลในตัวแปร y = ', y)
4 print('ค่าข้อมูลในตัวแปร msg = ', msg)
```

ค่าข้อมูลในตัวแปร x = 200  
ค่าข้อมูลในตัวแปร y = 300  
ค่าข้อมูลในตัวแปร msg = Python programming language



จากตัวอย่างการเขียนคำสั่งโปรแกรมได้ประกาศสร้างตัวแปร x และ y พร้อมทั้งกำหนดค่าข้อมูลเท่ากับ 200 และ 300 ตามลำดับ เป็นชนิดข้อมูลจำนวนเต็ม อีกทั้งยังได้ประกาศสร้างตัวแปร msg กำหนดค่าข้อมูลเป็นชนิดข้อมูลสตริง จะเห็นว่าเมื่อสร้างตัวแปรให้อยู่ในบรรทัดเดียวกันจะมีเครื่องหมาย semicolon (:) คั่นระหว่างตัวแปร

### 3.11 การตรวจสอบชนิดข้อมูล

การพัฒนาโปรแกรมที่มีขนาดใหญ่และมีการประมวลผลร่วมกันจำนวนมาก อาจจะทำให้ผู้พัฒนาโปรแกรมหลงลืมว่าตัวแปรเก็บชนิดข้อมูลประเภทใดไว้ เพื่อป้องกันการนำตัวแปรมาประมวลผลหรือดำเนินการผิดพลาด เราอาจจะต้องตรวจสอบชนิดข้อมูลของตัวแปรนั้นก่อน โดยการเรียกใช้งานฟังก์ชัน `type()` ซึ่งเป็นฟังก์ชัน Built-in แสดงตัวอย่างการเรียกใช้งานดังต่อไปนี้

#### ตัวอย่าง 3.22

การตรวจสอบชนิดข้อมูลด้วยฟังก์ชัน `type()`

```
1 msg = 'Python Programming language'
2 num = 254
3 float_ = 354.213
4 print(type(msg))
5 print(type(num))
6 print(type(float_))
```

```
<class 'int'>
<class 'str'>
<class 'float'>
```



จากตัวอย่างการเขียนคำสั่งโปรแกรม ได้ประกาศสร้างตัวแปร `msg`, `num` และ `float_` หากต้องการอยากรู้ว่าทั้ง 3 ตัวแปรเก็บชนิดข้อมูลประเภทใดไว้ ให้เรียกใช้งานฟังก์ชัน `type()` ดังแสดงในบรรทัดที่ 4-6

### 3.12 นิพจน์

นิพจน์ (Expressions) ในภาษาโปรแกรมไพธอน คือ ข้อความที่แสดงการดำเนินการเพื่อคำนวณหรือทำการเปรียบเทียบหาค่าต่าง ๆ โดยที่ในการดำเนินการจะประกอบด้วยค่าคงที่ หรือตัวแปร สิงเหล่านี้เรียกว่า ตัวถูกดำเนินการ (Operand) และตัวดำเนินการ (Operator)

พิจารณา尼พจน์ต่อไปนี้

```
z = a / b * (c + d)
```

นิพจน์นี้ประกอบด้วยตัวถูกดำเนินการ 5 ตัวคือ  $a$ ,  $b$ ,  $c$ ,  $d$  และ  $z$  สำหรับตัวดำเนินการได้แก่ เครื่องหมาย  $+$ ,  $/$ ,  $*$  และ  $=$

ตัวดำเนินการในภาษาโปรแกรมไพธอนมีหลายแบบให้เลือกใช้งาน และแต่ละแบบทำหน้าที่แตกต่าง กันออกไป เช่น  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $<$ ,  $>$ ,  $=$  เป็นต้น เราสามารถที่จะนำตัวดำเนินการมาผสมเป็น นิพจน์ให้ทำงานร่วมกันได้ โดยแบ่งตัวดำเนินการออกเป็นกลุ่มได้ดังทวัญตอนต่อไป

### 3.13 ตัวดำเนินการทางคณิตศาสตร์

ภาษาไพธอนมีตัวดำเนินการทางคณิตศาสตร์ ได้แก่  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$ ,  $//$ ,  $**$  ซึ่ง ตาราง 3.1 ได้แสดงสัญลักษณ์ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators) ความหมายของสัญลักษณ์ ตัวอย่างการใช้งาน และผลลัพธ์ที่ได้จากการคำนวณ ซึ่งผลลัพธ์ที่ได้จากการคำนวณจะแสดงผลลัพธ์ของ ชนิดข้อมูลที่ไม่เหมือนกัน ขึ้นอยู่กับการทำหนดชนิดข้อมูลให้กับตัวแปร จากตารางกำหนดค่าตัวแปร  $a$  และ  $b$  เป็นชนิดข้อมูลจำนวนเต็ม (Integer) ผลลัพธ์ที่ได้จากการจะแสดงผลลัพธ์มาได้เป็นชนิดข้อมูล จำนวนทศนิยม (float) ส่วนผลลัพธ์ที่เหลือจะแสดงผลเป็นชนิดข้อมูลจำนวนเต็ม (Integer)

สัญลักษณ์	ความหมาย	ตัวอย่างการใช้	ผลลัพธ์ ( $a=5$ , $b=3$ )
+	บวก (Addition)	$c = a + b$	$c = 8$
-	ลบ (Subtraction)	$c = a - b$	$c = 2$
*	คูณ (Multiplication)	$c = a * b$	$c = 15$
/	หาร (Division)	$c = a / b$	$c = 1.66666667$
%	หารเอาเศษ (Modulo)	$c = a \% b$	$c = 2$
//	หารปัดเศษ (Floor Division)	$c = a // b$	$c = 1$
**	ยกกำลัง (Exponent)	$c = a ** b$	$c = 125$

ตาราง 3.1: สัญลักษณ์ตัวดำเนินการคำนวณทางคณิตศาสตร์

### ตัวอย่าง 3.23

การนำตัวดำเนินการทางคณิตศาสตร์ต่าง ๆ มาใช้คำนวณหาผลลัพธ์

```

1 a = 5; b = 3; # สร้างตัวแปร a และ b เป็นชนิดข้อมูลจำนวนเต็ม
2 c = a + b # บวกค่าตัวแปร a กับ b
3 d = a / b # หารค่าตัวแปร a กับ b
4 x = a ** b # ยกกำลังค่าตัวแปร a กับ b
5 z = a // b # หารเอาเศษค่าตัวแปร a กับ b
6 print('ผลลัพธ์ของ a + b คือ ', c) # แสดงผลลัพธ์ที่ค่าตัวแปร c
7 print('ผลลัพธ์ของ a / b คือ ', d) # แสดงผลลัพธ์ค่าตัวแปร d
8 print('ผลลัพธ์ของ a ** b คือ ', x) # แสดงผลลัพธ์ค่าตัวแปร x
9 print('ผลลัพธ์ของ a // b คือ ', z) # แสดงผลลัพธ์ค่าตัวแปร z

```

ผลลัพธ์ของ  $a + b$  คือ 8  
 ผลลัพธ์ของ  $a / b$  คือ 1.6666666666666667  
 ผลลัพธ์ของ  $a ** b$  คือ 125  
 ผลลัพธ์ของ  $a // b$  คือ 1



### 3.14 ตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการเปรียบเทียบ (Comparison Operator) จะทำการเปรียบเทียบค่าข้อมูลที่ลงทะเบียนไว้ 2 ตัว ด้วยเครื่องหมายการเปรียบเทียบ ดังแสดงในตาราง 3.2 ผลลัพธ์ที่ได้จะมีเพียง 2 ค่าคือ จริง (**True**) และเท็จ (**False**) ซึ่งเป็นรูปแบบของตรรกะ

จากตัวอย่างที่ 3.23 แสดงการเขียนคำสั่งโปรแกรมโดยใช้ตัวดำเนินการเปรียบเทียบแบบต่าง ๆ จะสังเกตเห็นว่าในบรรทัดที่ 9 สามารถประยุกต์ใช้ตัวดำเนินการเปรียบเทียบกับตัวถูกดำเนินการได้มากกว่า 2 ตัว โดยการทำงานจะเปรียบเทียบตัวถูกดำเนินการคู่ทางด้านซ้ายมือก่อน จากนั้นจึงเปรียบเทียบกับตัวถูกดำเนินการทางด้านขวาเมื่อ เช่น  $b == d$  หรือไม่ ผลลัพธ์คือ จริง และ  $d < a$  หรือไม่ คำตอบคือ จริง ทำให้ผลลัพธ์การเปรียบเทียบเป็นจริง

#### ตัวอย่าง 3.24

การใช้งานตัวดำเนินการเปรียบเทียบและผลลัพธ์

```
1 a = 5; b = 3; c = 15; d = 3 # สร้างตัวแปร a, b, c และ d เป็นชนิดข้อมูลจำนวนเต็ม
2 w = (a == c) # เปรียบเทียบค่าตัวแปร a เท่ากับ c หรือไม่
3 x = (d <= a) # เปรียบเทียบค่าตัวแปร d น้อยกว่าหรือเท่ากับ a หรือไม่
4 y = (b >= c) # เปรียบเทียบค่าตัวแปร b มากกว่าหรือเท่ากับ c หรือไม่
5 z = (b == d < a) # เปรียบเทียบค่าตัวแปร b เท่ากับ d และน้อยกว่า a หรือไม่
6 print('a เท่ากับ c หรือไม่ = ', w) # แสดงผลลัพธ์ที่เก็บอยู่ในค่าตัวแปร w
7 print('d น้อยกว่าหรือเท่ากับ a หรือไม่ = ', x) # แสดงผลลัพธ์ค่าตัวแปร x
8 print('b มากกว่าหรือเท่ากับ c หรือไม่ = ', y) # แสดงผลลัพธ์ค่าตัวแปร y
9 print('b เท่ากับ d และน้อยกว่า a หรือไม่ = ', z) # แสดงผลลัพธ์ค่าตัวแปร z
```

a เท่ากับ c หรือไม่ = False  
d น้อยกว่าหรือเท่ากับ a หรือไม่ = True  
b มากกว่าหรือเท่ากับ c หรือไม่ = False  
b เท่ากับ d และน้อยกว่า a หรือไม่ = True



สัญลักษณ์	ความหมาย	ตัวอย่างการใช้	ผลลัพธ์ ( $a=5$ , $b=3$ )
$==$	เท่ากับ (Equal)	$a == b$	<b>False</b>
$!=$	ไม่เท่ากับ (Not Equal)	$a != b$	<b>True</b>
$<$	น้อยกว่า (Less than)	$a < b$	<b>False</b>
$<=$	น้อยกว่าหรือเท่ากับ (Less than or Equal)	$a <= b$	<b>False</b>
$>$	มากกว่าหรือเท่ากับ (Greater than or Equal)	$a >= b$	<b>True</b>

ตาราง 3.2: สัญลักษณ์ตัวดำเนินการเปรียบเทียบ

### 3.15 ตัวดำเนินการกำหนดค่า

ตัวดำเนินการกำหนดค่า (Assignment Operator) เป็นตัวดำเนินการที่ทำหน้าที่กำหนดค่าข้อมูล หรือ ข้อมูลที่เก็บไว้ในตัวแปรที่อยู่ทางด้านขวา มีอีกส่วนหนึ่งที่มีสัญลักษณ์ของตัวดำเนินการประเภทนี้แสดงในตาราง 3.3

#### ตัวอย่าง 3.25

การใช้งานตัวดำเนินการกำหนดค่าและผลลัพธ์

```

1 a = 9; b = 5; c = 4; d = 6 # สร้างตัวแปร a, b, c และ d เป็นชนิดข้อมูลจำนวนเต็ม
2 a = b # กำหนดค่าตัวแปร a เท่ากับ b
3 c += b # บวกแล้วเก็บผลลัพธ์ไว้ที่ตัวแปร c
4 d -= b # ลบแล้วเก็บผลลัพธ์ไว้ที่ตัวแปร b
5 print('ผลลัพธ์ a = b :', a) # แสดงผลลัพธ์ค่าตัวแปร a
6 print('ผลลัพธ์ c += b :', c) # แสดงผลลัพธ์ค่าตัวแปร c
7 print('ผลลัพธ์ d -= b :', d) # แสดงผลลัพธ์ค่าตัวแปร d

```

ผลลัพธ์ a = b : 5

ผลลัพธ์ c += b : 9

ผลลัพธ์ d -= b : 1



สัญลักษณ์	ความหมาย	ตัวอย่างการใช้	ผลลัพธ์ ( $a=5$ , $b=3$ )
=	กำหนดค่า	$a = b$ นำค่าตัวแปร $b$ มาใส่ในตัวแปร $a$	$a = 3$
+=	บวกก่อนแล้ว กำหนดค่า	$a += b$ คำนวณ $a + b$ และเอาผลมาเก็บไว้ที่ $a$	$a = 8$
-=	ลบก่อนแล้ว กำหนดค่า	$a -= b$ คำนวณ $a - b$ และเอาผลมาเก็บไว้ที่ $a$	$a = 2$
*=	คูณก่อนแล้ว กำหนดค่า	$a *= b$ คำนวณ $a * b$ และเอาผลมาเก็บไว้ที่ $a$	$a = 15$
/=	หารก่อนแล้ว กำหนดค่า	$a /= b$ คำนวณ $a / b$ และเอาผลมาเก็บไว้ที่ $a$	$a = 1.667$
%=	หารเอาเศษ แล้วกำหนดค่า	$a \%= b$ คำนวณ $a \% b$ และเอาผลมาเก็บไว้ที่ $a$	$a = 2$
**=	ยกกำลังแล้ว กำหนดค่า	$a **= b$ คำนวณ $a ** b$ และเอาผลมาเก็บไว้ที่ $a$	$a = 125$
//=	หารเอาส่วน แล้วกำหนดค่า	$a // = b$ คำนวณ $a // b$ และเอาผลมาเก็บไว้ที่ $a$	$a = 1$

ตาราง 3.3: สัญลักษณ์ตัวดำเนินการกำหนดค่า

### 3.16 ตัวดำเนินการตรรกศาสตร์

ตัวดำเนินการตรรกศาสตร์ (Logical Operator) เป็นตัวดำเนินการที่ใช้เปรียบเทียบระหว่างตัวถูกดำเนินการ 2 ตัว เพื่อตัดสินใจว่าเป็นจริง (**True**) หรือเป็นเท็จ (**False**) มีตัวดำเนินการประเภทนี้ในภาษาโปรแกรมไพธอนให้ใช้งานอยู่ 3 ชนิด ได้แก่ **and**, **or** และ **not** สามารถนำตารางความจริง (Truth table) มาประยุกต์เปรียบเทียบระหว่างตัวถูกดำเนินการได้

ในตาราง 3.5 เราได้แสดงค่าความจริงเมื่อนำตัวดำเนินการตรรกศาสตร์มาดำเนินการกับตัวถูกดำเนินการ **p** และ **q** เมื่อใช้ตัวดำเนินการ **and** จะได้ผลลัพธ์ **True** เมื่อ **p** และ **q** เป็น **True** เท่านั้น แต่หากใช้ดำเนินการ **or** จะได้ผลลัพธ์ **False** เมื่อ **p** และ **q** เป็น **False** ทั้งสอง สำหรับการใช้ตัวดำเนินการ **not** เป็นการเปลี่ยนค่าตรงข้าม **True** เป็น **False** และ **False** เป็น **True**

#### ตัวอย่าง 3.26

การใช้งานตัวดำเนินการตรรกศาสตร์และการประยุกต์ใช้เปลี่ยนคำสั่งโปรแกรม

```
1 a = 5; b = 3; c = 15 # สร้างตัวแปร a, b และ c เป็นชนิดข้อมูลจำนวนเต็ม
2 # ใช้ตัวดำเนินการเปรียบเทียบและตรรกศาสตร์เปรียบเทียบค่าตัวแปร
3 x = a == b and c < b
4 y = b > a or a > c
5 z = not (b > a or c < a)
6
7 print('ผลลัพธ์ของ a == b and c < b คือ ', x) # แสดงผลลัพธ์ตัวแปร x
8 print('ผลลัพธ์ของ b > a or a > c คือ ', y) # แสดงผลลัพธ์ตัวแปร y
9 print('ผลลัพธ์ของ not (b > a or c < a) คือ ', z) # แสดงผลลัพธ์ตัวแปร z
```

ผลลัพธ์ของ **a == b and c < b** คือ **False**  
ผลลัพธ์ของ **b > a or a > c** คือ **False**  
ผลลัพธ์ของ **not (b > a or c < a)** คือ **True**



ตัวดำเนินการ	ความหมาย	อธิบายการใช้งาน	ผลลัพธ์ ( <b>a=True, b=False</b> )
<b>and</b>	และ	<b>a and b</b>	<b>False</b>
<b>or</b>	หรือ	<b>a or b</b>	<b>True</b>
<b>not</b>	นิเสธ (ไม่)	<b>not a</b>	<b>False</b>
		<b>not b</b>	<b>True</b>

ตาราง 3.4: ตัวดำเนินการตรรกศาสตร์ในภาษาโปรแกรมไพธอน

<b>p</b>	<b>q</b>	<b>p and q</b>	<b>p or q</b>	<b>not p</b>	<b>not q</b>
True	True	True	True	False	False
True	False	False	True	False	True
False	True	False	True	True	False
False	False	False	False	True	True

ตาราง 3.5: ตารางค่าความจริง (Truth Table)

### 3.17 ตัวดำเนินการระดับบิต

ตัวดำเนินการระดับบิต (Bitwise Operator) เป็นตัวดำเนินการเปรียบเทียบที่มูลในระดับบิตที่มีค่า **0** และ **1** โดยไม่คำนึงถึงว่าชนิดข้อมูลเป็นอะไรภาษาโปรแกรมใดก็ตามจะแปลงข้อมูลให้อยู่ในรูปแบบเลขฐานสอง คือ **0** และ **1** ก่อน แล้วจึงนำไปดำเนินการหาผลลัพธ์การดำเนินการระหว่างตัวดำเนินการและตัวถูกดำเนินการจะให้ค่าเป็นเท็จ (**0**) หรือจริง (**1**) เราสามารถเปรียบเทียบการคำนวณได้กับตาราง 3.4 สำหรับสัญลักษณ์ต่าง ๆ ที่ใช้ดำเนินการระดับบิตแสดงในตารางที่ 3.6

#### ตัวอย่าง 3.27

การใช้งานตัวดำเนินการระดับบิต

```

1 a = 5
2 b = 3
3 c = 5
4 x = a & b
5 print('ผลลัพธ์ของ a & b คือ ', x)
6 y = a ^ c
7 print('ผลลัพธ์ของ a ^ c คือ ', y)
8 z = a } b
9 print('ผลลัพธ์ของ a } b คือ ', z)

```

ผลลัพธ์ของ a & b คือ 1

ผลลัพธ์ของ a ^ b คือ 0

ผลลัพธ์ของ a } b คือ 7



สัญลักษณ์	ความหมาย	ตัวอย่างการใช้	ผลลัพธ์ ( $a=5$ , $b=3$ )
$\&$	และ (and)	$a \& b$	$a = 0101$ , $b = 0011$ ( $a \& b = 1$ )
$ $	หรือ (or)	$a   b$	$a = 0101$ , $b = 0011$ ( $a   b = 7$ )
$^$	xor	$a ^ b$	$a = 0101$ , $b = 0011$ ( $a ^ b = 6$ )
$\sim$	คอมพลีเมนต์	$\sim a$	$a = 0101$ , $b = 0011$ ( $\sim a = -6$ , $\sim b = 4$ )
$<<$	เลื่อนบิตไปทางซ้าย	$a << b$	$a=0101$ , $b = 0011$ ( $a << 1 = 10$ , $b << 1 = 6$ )
$>>$	เลื่อนบิตไปทางขวา	$a >> b$	$a = 0101$ , $b = 0011$ ( $a >> 1 = 2$ , $b >> 1 = 1$ )

ตาราง 3.6: สัญลักษณ์ตัวดำเนินการระดับบิตที่ใช้ในภาษาโปรแกรมเพร่อน

### 3.18 ตัวดำเนินการตรวจสอบสมาชิก

ตัวดำเนินการตรวจสอบสมาชิก (Membership Operator) ใช้สำหรับดำเนินการเปรียบเทียบคันหาค่าที่กำหนดในตัวแปรที่สนใจ เป็นสมาชิกในตัวแปรที่จะทำการเปรียบเทียบที่หรือไม่ใช้กับชนิดข้อมูลสตริง (String), ลิสต์ (List) และ ทูเพิล (Tuple) ซึ่งเราจะกล่าวถึงชนิดข้อมูลนี้ในบทที่ 4 และ 5 โดยละเอียด ในตาราง 3.7 เราได้แสดงตัวดำเนินการตรวจสอบสมาชิก และตัวอย่างการใช้งาน เมื่อกำหนดให้

```
cars = ['Honda', 'Toyota', 'BMW', 'Mercedes-Benz', 'Nissan',  
       'Mazda']  
  
และ  
  
x = 'Ford'
```

#### ตัวอย่าง 3.28

การใช้งานตัวดำเนินการตรวจสอบสมาชิก

```
1 cars = ['Honda', 'Toyota', 'BMW', 'Mercedes-Benz', 'Nissan', 'Mazda']  
2 my_car = 'Ford'  
3 x = my_car in cars  
4 y = my_car not in cars  
5 print('ผลลัพธ์ของ my_car in cars = ', x)  
6 print('ผลลัพธ์ของ my_car not in cars = ', y)
```

```
ผลลัพธ์ของ my_car in cars = False  
ผลลัพธ์ของ my_car not in cars = True
```



สัญลักษณ์	ความหมาย	การใช้งาน	อธิบายผลลัพธ์
in	เป็นสมาชิก	x in cars	ให้ค่าเป็น <b>False</b> เพราะ 'Ford' ไม่มีอยู่ในลิสต์ของ cars
not in	ไม่เป็นสมาชิก	x not in cars	ให้ค่าเป็น <b>True</b> เพราะ 'Ford' ไม่มีอยู่ในลิสต์ของ cars

ตาราง 3.7: ตัวดำเนินการตรวจสอบสมาชิก

### 3.19 ตัวดำเนินการเอกลักษณ์

ตัวดำเนินการเอกลักษณ์ (Identity Operator) เป็นตัวดำเนินการที่นำมาใช้เพื่อเปรียบเทียบข้อมูลระหว่างตัวแปร 2 ตัว มีให้เลือกใช้งานอยู่ 2 ตัวคือ **is** และ **is not** เพื่อบอกว่ามีความเท่ากันหรือไม่เท่ากัน ตัวดำเนินการ **is** จะให้คำตอบเหมือนกับการใช้ตัวดำเนินการ **==** ส่วน **is not** ให้คำตอบเหมือนกับการใช้ตัวดำเนินการ **!=**

#### ตัวอย่าง 3.29

การใช้งานตัวดำเนินการเอกลักษณ์เปรียบเทียบค่าตัวแปร

```
1 a = 5
2 b = 3
3 x = a is b
4 y = a is not b
5 print('ผลลัพธ์ของ a is b คือ ', x)
```

ผลลัพธ์ของ a is b คือ False



#### ตัวอย่าง 3.30

การใช้งานตัวดำเนินการเอกลักษณ์เปรียบเทียบค่าตัวแปร

```
1 a = 5
2 b = 5
3 x = a is b
4 y = a is not b
5 print('ผลลัพธ์ของ a is b คือ ', x)
```

ผลลัพธ์ของ a is b คือ True



สัญลักษณ์	ความหมาย	ตัวอย่างการใช้	ผลลัพธ์ (a=5, b=3)
<b>is</b>	เป็น, อยู่	a is b	ให้ค่าเป็น <b>False</b> เพราะ a ไม่เท่ากับ b
<b>is not</b>	ไม่เป็น, ไม่อยู่	a is not b	ให้ค่าเป็น <b>True</b> เพราะ a ไม่เท่ากับ b

ตาราง 3.8: ตัวดำเนินการเอกลักษณ์ที่ใช้ในภาษาโปรแกรมไพธอน

## 3.20 ลำดับความสำคัญตัวดำเนินการ

ในหนึ่งนิพจน์อาจจะมีมากกว่าหนึ่งตัวดำเนินการแต่ละตัวดำเนินการมีลำดับความสำคัญในการทำงานก่อนหลัง (Operator precedence) แตกต่างกัน บางตัวดำเนินการมีความสำคัญในการทำงานที่เท่ากัน เราจึงมีความจำเป็นที่จะต้องทราบถึงลำดับการทำงานของตัวดำเนินการ เพื่อนำไปเขียนเป็นคำสั่งโปรแกรมให้ทำงานได้ถูกต้อง และป้องกันข้อผิดพลาดจากการทำงานของโปรแกรม

การหาลำดับจากนิพจน์เริ่มจากทางด้านซ้ายไปทางด้านขวาเสมอ ในทางคอมพิวเตอร์ก็ทำแบบนี้ เช่นเดียวกัน แต่มีลำดับความสำคัญของเครื่องหมายเข้ามาเกี่ยวข้อง เพื่อควบคุมการหาผลลัพธ์จำเป็นต้องใช้เครื่องหมายวงเล็บ ( . . . ) ครอบนิพจน์ที่เราต้องการให้ดำเนินการก่อน เพราะเครื่องหมายวงเล็บ มีลำดับความสำคัญที่สุด พิจารณาตัวอย่างต่อไปนี้

### ตัวอย่าง 3.31

การเขียนคำสั่งโปรแกรมเปรียบเทียบการจัดลำดับความสำคัญของตัวดำเนินการ

```
1 x = 10 + 4 / 2 * 9  
2 y = ((10 + 4) / 2) * 9  
3 print('ผลลัพธ์ของ x = 10 + 4 / 2 * 9 คือ ', x)  
4 print('ผลลัพธ์ของ ((10+4) / 2) * 9 คือ ', y)
```

ผลลัพธ์ของ  $x = 10 + 4 / 1 * 9$  คือ 28.0  
ผลลัพธ์ของ  $((10+4) / 2) * 9$  คือ 63.0



จากตัวอย่างการเขียนคำสั่งโปรแกรม แสดงให้เห็นถึงลำดับการประมวลผลของตัวดำเนินการในบรรทัด 1 จะเริ่มจากการหารก่อน คือ  $4 / 2$  ได้เท่ากับ 2 จากนั้นคูณกับ 9 ได้เท่ากับ 18 และบวกกับ 10 ผลลัพธ์ที่ได้จึงเท่ากับ 28.0

ในบรรทัดที่ 2 มีการจัดลำดับการประมวลผลโดยการใช้เครื่องหมายวงเล็บ โดยเริ่มจากวงเล็บด้านในสุดคือ  $10+4$  ได้เท่ากับ 14 และจึงหารด้วย 2 ได้เท่ากับ 7 จากนั้นคูณกับ 9 ผลลัพธ์ที่ได้จึงเท่ากับ 63.0

ลำดับ	ตัวดำเนินการ	คำอธิบาย
1	( ... )	ใช้เพื่อแบ่งนิพจน์และลำดับการทำงาน
2	**	สัญลักษณ์ยกกำลัง
3	~, +, -	คอมพลีเมนต์, unary plus, unary minus
4	*, /, %, //	การคูณ, การหาร, การหารเอาเศษ, การหารเอาส่วน
5	+, -	การบวกและการลบ
6	>>, <<	การเลื่อนบิตทางขวา, การเลื่อนบิตทางซ้าย
7	&	and ในระดับบิต
8	^,	xor, or ในระดับบิต
9	<=, <, >, >=	ตัวดำเนินการเปรียบเทียบ
10	==, !=	เท่ากับและไม่เท่ากับ
11	=, %=, /=, //=, -=, +=, *=, **=	ตัวดำเนินการกำหนดค่า
12	is, is not	อยู่/ไม่อยู่ ตัวดำเนินการเอกลักษณ์
13	in, not in	อยู่ใน/ไม่อยู่ใน ตัวดำเนินการความเป็นสมาชิก
14	not, or, and	ตัวดำเนินการตรรกศาสตร์

ตาราง 3.9: ลำดับความสำคัญของตัวดำเนินการ

## **สรุปก่อนจบบท**

ในบทนี้เราได้รู้จักกับชนิดข้อมูลต่าง ๆ การสร้างตัวแปรและกำหนดค่าข้อมูลก่อนนำไปใช้งาน การแปลงชนิดข้อมูลหนึ่งให้เป็นอีกชนิดข้อมูลหนึ่ง นอกจากนี้ยังได้รู้จักฟังก์ชันที่ใช้สำหรับแปลงเลขฐานสอง ฐานแปด ฐานสิบ และฐานสิบหก และยังได้รู้จักกับฟังก์ชันที่ใช้ตรวจสอบชนิดข้อมูลกับนิพจน์และตัวดำเนินการประเภทต่าง ๆ เช่น ตัวดำเนินการทางคณิตศาสตร์ ตัวดำเนินการเปรียบเทียบ เป็นต้น

## แบบฝึกหัด

1. จงบอกชนิดข้อมูลจากการประมวลผลตัวแปรดังต่อไปนี้

```
a = 'Bla Bla Blaaa'  
b = 9 + 8 + 7 + 6 + 5  
c = (b ** 2) / 6  
d = b <= 6 * c  
e = 2.71828 * 10000  
f = 9.0  
g = 'Bye ' * b  
h = 2 * 3.06456E-4
```

2. จงเขียนโปรแกรมคำนวณภาษีที่ต้องจ่าย โดยมีอัตราภาษีเท่ากับ 7% กำหนดให้ตัวแปรอัตราภาษี เป็นค่าคงที่ และให้โปรแกรมรับยอดเงินรายได้สุทธิผ่านทางแป้นพิมพ์
3. จงเขียนโปรแกรมคำนวณอายุปัจจุบัน โดยให้โปรแกรมรับข้อมูลวันเกิดผ่านทางแป้นพิมพ์
4. จงเขียนผลลัพธ์ที่ได้จากโปรแกรมต่อไปนี้

```
x = 345  
print(bin(x))  
print(oct(x))  
print(hex(x))
```

5. จงเขียนผลลัพธ์ที่ได้จากโปรแกรมต่อไปนี้

```
x = 25 + 32 / 2 + 5 * 2 + 15 * 4  
y = 45 + 2 ** 4 + 5 * 2 * 3 - 1  
print(x)  
print(y)
```



## บทที่ 4

# ชนิดข้อมูลสตริง

ในบทที่ผ่านมาเร่าได้รู้จักกับชนิดข้อมูลพื้นฐาน เช่น ชนิดข้อมูลจำนวนเต็ม ชนิดข้อมูลจำนวนทศนิยม ชนิดข้อมูลตรรกะ และชนิดข้อมูลสตริงไปบ้างแล้ว สำหรับชนิดข้อมูลสตริงเป็นชนิดข้อมูลที่มีความสำคัญมากอีกหนึ่งชนิดข้อมูล เพราะการเขียนโปรแกรมต้องมีการดำเนินการกับสตริงหรือข้อความอยู่เสมอ เช่น การตัดข้อความ การเชื่อมต่อข้อความ การแทนที่ข้อความ เป็นต้น บทนี้เราจะได้รู้จักวิธีการจัดการกับชนิดข้อมูลสตริง รวมทั้งการเรียกใช้งานเมธอด (Method) ต่าง ๆ ที่นำมาจัดการกับชนิดข้อมูลสตริง

### 4.1 รู้จักกับชนิดข้อมูลสตริง

ชนิดข้อมูลสตริง (String) สามารถเป็นได้ทั้งตัวอักษรหรือข้อความที่ประกอบด้วยตัวอักษรหลาย ๆ ตัวที่นำมารวมกัน ในภาษาไพธอนการประกาศสร้างตัวแปรเก็บชนิดข้อมูลสตริงจะอยู่ในเครื่องหมาย ('...') หรือ ("...") สำหรับภาษาไพธอนเวอร์ชัน 3 ได้เปลี่ยนการจัดเก็บสตริงจาก ASCII ที่ใช้ในภาษาไพธอนเวอร์ชัน 2 มาเป็นแบบ Unicode ซึ่งรองรับการจัดเก็บอักษรในหลากหลายภาษา ตัวอย่างการประกาศตัวแปรชนิดข้อมูลสตริง แสดงตังตัวอย่างต่อไปนี้

#### ตัวอย่าง 4.1

การเขียนคำสั่งโปรแกรมประกาศตัวแปรเก็บชนิดข้อมูลสตริง

```
1 # ใช้เครื่องหมาย '...' เป็นชนิดข้อมูลสตริง
2 str1 = 'Hello Python'
3 # ใช้เครื่องหมาย "..." เป็นชนิดข้อมูลสตริง
4 str2 = "สวัสดีเพร่อน"
5 print(str1) # แสดงผลค่าตัวแปร str1
6 print(str2) # แสดงผลค่าตัวแปร str2
```

Hello Python  
สวัสดีเพร่อน



หากต้องการแปลงชนิดข้อมูลใด ๆ ให้เป็นชนิดข้อมูลสตริง ให้เรียกใช้งานฟังก์ชัน `str()` ดังตัวอย่าง ต่อไปนี้

### ตัวอย่าง 4.2

การแปลงชนิดข้อมูลจำนวนเต็มเป็นชนิดข้อมูลสตริง ด้วยฟังก์ชัน `srt()`

```
1 num = 245 # ประกาศตัวแปรชนิดข้อมูลจำนวนเต็ม
2 str1 = str(num) # แปลงค่าตัวแปร num เป็นชนิดข้อมูลสตริง
3 print(str1) # แสดงผลค่าตัวแปร str1
4 print('ชนิดข้อมูล num คือ', type(num)) # แสดงประเภทข้อมูลค่าตัวแปร num
5 print('ชนิดข้อมูลของ str1 คือ', type(str1)) # แสดงประเภทข้อมูลค่าตัวแปร str1
```

245

ชนิดข้อมูล num คือ <class 'int'>  
ชนิดข้อมูลของ str1 คือ <class 'str'>



หลังจากมีการแปลงชนิดข้อมูลจำนวนเต็มเป็นชนิดข้อมูลสตริงแล้ว และนำตัวดำเนินการคณิตศาสตร์มาดำเนินการกับชนิดข้อมูลทั้งสองจะทำให้เกิดการแจ้งเตือนข้อผิดพลาดขึ้น แสดงดังต่อไปนี้

### ตัวอย่าง 4.3

การเขียนคำสั่งโปรแกรมดำเนินการกับชนิดข้อมูลสตริง

```
1 num = 347
2 num1 = 245
3 str1 = str(num) # แปลงค่าตัวแปร num เป็นชนิดข้อมูลสตริง
4 result = str1 + num1 # บวกค่าตัวแปร str1 กับ num1
5 print(result) # แสดงผลลัพธ์ตัวแปร result
```

```
TypeError Traceback (most recent call last)
<ipython-input-1-06da36824501> in <module>
  1 num = 347
  2 num1 = 245
  3 str1 = str(num) # แปลงค่าตัวแปร num เป็นชนิดข้อมูลสตริง
  4 result = str1 + num1 # บวกค่าตัวแปร str1 กับ num1
  5 print(result) # แสดงผลลัพธ์ตัวแปร result
```



TypeError: can only concatenate str (not 'int') to str

จากตัวอย่างโปรแกรมจะเกิดการแจ้งเตือนข้อผิดพลาดขึ้น เนื่องจากค่าตัวแปร num ซึ่งถูกแปลงเป็นชนิดข้อมูลสตริง ซึ่งไม่สามารถดำเนินการบวกกับค่าตัวแปร num1 ซึ่งเป็นชนิดข้อมูลจำนวนเต็มได้

## 4.2 การเขื่อมต่อสตริง

เมื่อต้องการเชื่อมชนิดข้อมูลสตริงเข้าด้วยกัน สามารถนำค่าตัวแปรมาต่อ กันในฟังก์ชัน `print()` แล้วคั่นด้วยเครื่องหมาย Comma ( , ) หรือใช้ตัวดำเนินการ (+) เชื่อมต่อสตริงเข้าด้วยกัน และเมื่อต้องการเว้นวรรคระหว่างข้อความให้ใช้สตริงช่องว่าง ' ' (White Space String) แสดงดังตัวอย่างต่อไปนี้

### ตัวอย่าง 4.4

การเขียนคำสั่งโปรแกรมเชื่อมชนิดข้อมูลสตริงเข้าด้วยกัน

```
1 str1 = 'Python'
2 str2 = 'is a programming'
3 str3 = 'language.'
4 print(str1, str2, str3) # แสดงผลลัพธ์จากตัวแปร
5 print(str1 + ' ' + str2 + ' ' + str3) # เทิ่อมข้อมูลจากค่าตัวแปรเข้าด้วยกัน
```

Python is a programming language.  
Python is a programming language.



กรณีที่เราต้องการทำซ้ำข้อความ ให้ใช้ตัวดำเนินการ \* และกำหนดจำนวนครั้งที่ต้องการทำซ้ำ แสดงดังตัวอย่างต่อไปนี้

### ตัวอย่าง 4.5

การเขียนคำสั่งโปรแกรมทำซ้ำชนิดข้อมูลสตริง ตามจำนวนครั้งที่กำหนด

```
1 str1 = 'Python '
2 star = '*'
3 print(str1 * 3) # แสดงผลลัพธ์ตัวแปร str1 จำนวน 3 ครั้ง
4 print(3 * str1) # แสดงผลลัพธ์ตัวแปร str1 จำนวน 3 ครั้ง
5 print (3 * star) # แสดงผลลัพธ์ตัวแปร star จำนวน 3 ครั้ง
```

Python Python Python  
Python Python Python  
\*\*\*



การหาจำนวนอักขระที่อยู่ในข้อความให้เรียกใช้งานด้วยฟังก์ชัน `len()` แสดงดังตัวอย่างต่อไปนี้

## ตัวอย่าง 4.6

การเขียนคำสั่งโปรแกรมแสดงจำนวนอักขระในข้อความ ด้วยฟังก์ชัน `len()`

```
1 str1 = 'Python'
2 str2 = 'Python '
3 str3 = ''
4 str4 = ''
5 str5 = 'IXOHOXI'
6 print('จำนวนอักขระใน str1 =', len(str1)) # แสดงจำนวนอักขระของ str1
7 print('จำนวนอักขระใน str2 =', len(str2)) # แสดงจำนวนอักขระของ str2
8 print('จำนวนอักขระใน str3 =', len(str3)) # แสดงจำนวนอักขระของ str3
9 print('จำนวนอักขระใน str4 =', len(str4)) # แสดงจำนวนอักขระของ str4
10 print('จำนวนอักขระใน str5 =', len(str5)) # แสดงจำนวนอักขระของ str5
```

```
จำนวนอักขระใน str1 = 6
จำนวนอักขระใน str2 = 7
จำนวนอักขระใน str3 = 0
จำนวนอักขระใน str4 = 1
จำนวนอักขระใน str5 = 7
```



จากตัวอย่างจะเห็นว่าตัวแปร `str3` และ `str4` ไม่มีตัวอักขระใด ๆ แต่ผลลัพธ์ที่ได้จากการตัวแปร `str4` เท่ากับ 1 เนื่องจากเก็บค่าซองว่างเอาไว้

## 4.3 การเข้าถึงตำแหน่งตัวอักขระในชนิดข้อมูลสตริง

การเข้าถึงตัวอักขระในชนิดข้อมูลสตริง (Indexing String Operator) จะใช้เครื่องหมาย `[ ]` โดยการระบุตำแหน่งหรือ Index ตำแหน่งแรกเริ่มต้นที่ 0 หากต้องการเข้าถึงตำแหน่งตัวอักขระจากด้านท้ายสุด ตำแหน่งแรกจะเป็น -1 แสดงตำแหน่งการเก็บตัวอักขระในสตริงตั้งต่อไปนี้

Negative Index	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
<code>str =</code>	H	E	L	L	O		P	Y	T	H	O	N
Positive Index	0	1	2	3	4	5	6	7	8	9	10	11

การระบุตำแหน่งที่ต้องการ ผลลัพธ์ที่ได้จะแสดงตัวอักขระที่อยู่ ณ ตำแหน่งที่ได้ระบุไว้ มีรูปแบบการใช้งานดังนี้

**str[index]**

**str** ตัวแปรชนิดข้อมูลสตริง  
**index** ตำแหน่งที่ต้องการแสดงตัวอักษร

ถ้ากำหนดให้ **str = 'HELLO PYTHON'** เราสามารถเข้าถึงอักษรตำแหน่งแรกของตัวแปร **str** ด้วยคำสั่ง **str[0]** ซึ่งก็คือ '**H**' นั่นเอง ในทำนองเดียวกันเราสามารถเข้าถึงอักษรตำแหน่ง สุดท้ายและตำแหน่งรองสุดท้ายของตัวแปร **str** ด้วยคำสั่ง **str[-1]** และ **str[-2]** ซึ่งก็คือ '**N**' และ '**O**' ตามลำดับ และอักษรของตัวแปร **str** ที่มี Index เป็น 5 และ -7 คือ สตริงช่องว่าง (White Space String) ทั้งนี้ภาษาโปรแกรมไพธอนไม่อนุญาตให้เราเข้าถึงอักษรของตัวแปร **str** ที่มี Index สูงกว่า **len(str)** ซึ่งก็คือ 11 และ Index ต่ำกว่า **-len(str)-1** ซึ่งก็คือ -12

#### ตัวอย่าง 4.7

การเขียนคำสั่งโปรแกรมแสดงตัวอักษรโดยการระบุตำแหน่ง

```
1 str1 = 'ABCDEFGHIJKLMNPQRSTUVWXYZ' # สร้างตัวแปรชนิดข้อมูลสตริง
2 print('ตำแหน่งที่ 0 คือ', str1[0]) # แสดงตัวอักษรในตำแหน่งที่ 0
3 print('ตำแหน่งที่ -1 คือ', str1[-1]) # แสดงตัวอักษรในตำแหน่งที่ -1
4 print('ตำแหน่งที่ 5 คือ', str1[5]) # แสดงตัวอักษรในตำแหน่งที่ 5
5 print('ตำแหน่งที่ -5 คือ', str1[-5]) # แสดงตัวอักษรในตำแหน่งที่ -5
6 print('ตำแหน่งที่ -15 คือ', str1[-15]) # แสดงตัวอักษรในตำแหน่งที่ -15
```

ตำแหน่งที่ 0 คือ A  
ตำแหน่งที่ -1 คือ Z  
ตำแหน่งที่ 5 คือ F  
ตำแหน่งที่ -5 คือ V  
ตำแหน่งที่ -15 คือ L



### ตัวอย่าง 4.8

การเข้าถึงอักขระโดยการระบุตำแหน่งนอกขอบเขต

```
1 str1 = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'
2 n = len(str1)
3 print(str1[n+1]) #Index out of range
4 print(str1[-n-1]) #Index out of range too
```

Traceback (most recent call last):

```
  File "/Users/user/src/test.py", line 3, in <module>
    print(str1[n+1])
IndexError: string index out of range
```



การระบุตำแหน่งเริ่มต้นและตำแหน่งสุดท้าย ผลลัพธ์ที่ได้จะแสดงตัวอักขระตั้งแต่ตำแหน่งเริ่มต้นที่ได้ระบุจนถึงตำแหน่งสุดท้าย มีรูปแบบการใช้งานนี้

**str[start:stop]**

**str** ตัวแปรชนิดข้อมูลสตริง

**start** ตำแหน่งเริ่มต้นที่ต้องการแสดงตัวอักขระ

**stop** ตำแหน่งสุดท้ายที่ต้องการแสดงตัวอักขระ

## ตัวอย่าง 4.9

การเปลี่ยนคำสั่งโปรแกรมแสดงตัวอักษรโดยการระบุตำแหน่งเริ่มต้นและตำแหน่งสุดท้าย

```
1 str1 = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'
2 print('ผลลัพธ์ที่ได้จากการตัดคำ 0:4 =', str1[0:4]) # แสดง str1 ตัวที่ 0 ถึง 4
3 print('ผลลัพธ์ที่ได้จากการตัดคำ -6:-1 =', str1[-6:-1]) # แสดง str1 ตัวที่ -6 ถึง -1
4 print('ผลลัพธ์ที่ได้จากการตัดคำ 6:12 =', str1[6:12]) # แสดง str1 ตัวที่ 6 ถึง 12
5 print('ผลลัพธ์ที่ได้จากการตัดคำ 0:5 =', str1[:5]) # แสดง str1 ตัวที่ 0 ถึง 5
6 print('ผลลัพธ์ที่ได้จากการตัดคำ =', str1[:]) # แสดงอักขระ str1 ทั้งหมด
```

ผลลัพธ์ที่ได้จากการตัดคำ 0:4 = ABCD

ผลลัพธ์ที่ได้จากการตัดคำ -6:-1 = UVWXYZ

ผลลัพธ์ที่ได้จากการตัดคำ 6:12 = GHIJKL

ผลลัพธ์ที่ได้จากการตัดคำ 0:5 = ABCDE

ผลลัพธ์ที่ได้จากการตัดคำ = ABCDEFGHIJKLMNOPQRSTUVWXYZ



การระบุตำแหน่งเริ่มต้น ระบุตำแหน่งสุดท้าย และการก้าวกระโดด ผลลัพธ์ที่ได้จะแสดงตัวอักษร ตั้งแต่ตำแหน่งเริ่มต้น แต่จะมีการกระโดดข้ามตัวอักษรตามที่กำหนดไว้จนถึงตำแหน่งสุดท้าย มีรูปแบบ การใช้งานดังนี้

**str[start:stop:step]**

**str** ตัวแปรชนิดข้อมูลสตริง

**start** ตำแหน่งเริ่มต้นที่ต้องการแสดงตัวอักษร

**stop** ตำแหน่งสุดท้ายที่ต้องการแสดงตัวอักษร

**step** จำนวนการกระโดดข้ามตำแหน่ง

### ตัวอย่าง 4.10

การเขียนคำสั่งโปรแกรมโดยการระบุตำแหน่งเริ่มต้น ตำแหน่งสุดท้าย และการกำหนด

```
1 str1 = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'
2
3 # แสดงตัวอักษรจากตำแหน่งที่ -12 ถึง -1 โดยกรอบด้วยวงเล็บ
4 print('ผลลัพธ์ที่ได้จากการตัดคำ -12::2 =', str1[-12::2])
5
6 # แสดงตัวอักษรจากตำแหน่งที่ 2 ถึงตำแหน่งสุดท้าย โดยกรอบด้วยวงเล็บ 2 ตำแหน่ง
7 print('ผลลัพธ์ที่ได้จากการตัดคำ 1::2 =', str1[1::2])
```

ผลลัพธ์ที่ได้จากการตัดคำ -12::2 = OQSUWY

ผลลัพธ์ที่ได้จากการตัดคำ 1::2 = BDFHJLN PRTVXZ



## 4.4 การควบคุมการแสดงผลออกทางจอภาพ

โดยทั่วไปการแสดงผลลัพธ์ออกทางจอภาพ (Control Output) จะเรียกใช้งานฟังก์ชัน `print()` และ ข้อความที่อยู่ในเครื่องหมาย Single Quote (`'...'`) หรือ Double Quote (`"..."`) ถ้าข้อความ มีข้าความยาวจนต้องขึ้นบรรทัดใหม่ หรือต้องการให้แสดงผลออกทางหน้าจอหลายบรรทัด ให้ใช้ เครื่องหมาย Triple Quote (`'''...'''` หรือ `'''...'''`) แสดงดังตัวอย่างต่อไปนี้

### ตัวอย่าง 4.11

การเขียนคำสั่งโปรแกรมแสดงผลข้อความแบบหลายบรรทัด

```
1 str1 = '''Python is simple and easy to learn.
2 Python is simple and easy to learn.
3 Python is simple and easy to learn.'''
4 print(str1)
```

Python is simple and easy to learn.

Python is simple and easy to learn.

Python is simple and easy to learn.



ถ้ามีข้อความยาวจนต้องขึ้นบรรทัดใหม่ แต่ต้องการให้แสดงผลอยู่ในบรรทัดเดียวกัน ให้ใส่ เครื่องหมาย Backslash (\) ปิดท้ายก่อนขึ้นบรรทัดใหม่ เมื่อแสดงผลจะทำให้ข้อความอยู่ในบรรทัดเดียวกัน แสดงดังตัวอย่างต่อไปนี้

รหัสพิเศษ	ความหมาย
\a	ให้ส่งเสียงเตือน
\b	ให้เลื่อนเครื่อร์เซอร์โดยหลังไป 1 ตัวอักษร
\e	ให้ยกเลิกคำสั่งสุดท้าย
\f	ให้ขีนบรรทัดใหม่
\n	ให้ขีนบรรทัดใหม่หลังจากแสดงผล
\r	ให้เครื่อร์เซอร์อยู่ทางซ้ายมือ
\t	ให้แสดงแท็บตามแนวอน
\v	ให้แสดงแท็บตามแนวตั้ง
\	ให้แสดงผลเป็นบรรทัดเดียว
\\\	ให้แสดงเครื่องหมาย \
\'	ให้แสดงเครื่องหมาย '
\"	ให้แสดงเครื่องหมาย "

ตาราง 4.1: รหัสพิเศษควบคุมการแสดงผลชนิดข้อมูลสตริง ( Escape Sequences)

#### ตัวอย่าง 4.12

การเขียนคำสั่งโปรแกรมที่มีข้อความหลายบรรทัดแต่กำหนดให้แสดงผลในบรรทัดเดียวกัน

```

1 str1 = 'Hello world, \
2 Python is simple and easy to learn. \
3 Everyone loves me.'
4 print (str1)

```

Hello world, Python is simple and easy to learn. Everyone loves me. 

ในภาษาไพธอนสามารถควบคุมการแสดงผลของชนิดข้อมูลสตริงด้วย Escape Sequences ซึ่งเป็นรหัสพิเศษที่ใช้เครื่องหมาย Backslash (\) และตามด้วยตัวอักษร แสดงในตารางที่ 4.1

### ตัวอย่าง 4.13

การเขียนคำสั่งโปรแกรมแสดงผล

```
1 str1 = 'Python was conceived in 1980s by Guido van Rossum \
2 at Centrum Wiskunde & Informatica (CWI) in the Netherlands. \
3 Python has feature highlights such as: \n\
4 \t -> Easy-to-learn\n\
5 \t -> Easy-to-read\n\
6 \t -> Easy-to-maintain\n\
7 and \'Everyone can learn and practice in a short time\'.' \
8 print(str1)
```

Python was conceived in 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands. Python has feature highlights such as:  
-> Easy-to-learn  
-> Easy-to-read  
-> Easy-to-maintain  
and 'Everyone can learn and practice in a short time'.



#### 4.4.1 การจัดรูปแบบแสดงผล

การเปลี่ยนรูปแบบแสดงผล (Formatting) เป็นการจัดการข้อความ ตัวเลขจำนวนเต็ม ตัวเลขทศนิยม ให้อยู่ในลักษณะรูปแบบต่าง ๆ เช่น กำหนดให้มีทศนิยมกี่ตำแหน่ง กำหนดเลขยกกำลัง เป็นต้น โดยการใช้เครื่องหมายเปอร์เซ็นต์ (%) และตามด้วยค่าตัวแปร

โครงสร้างการจัดรูปแบบแสดงผลโดยใช้เครื่องหมาย %

```
print('Str1: %s, Str2: %f' %(Arg1, Arg2))
```

สัญลักษณ์การจัดรูปแบบชนิดข้อมูลสตริง (String Formatting Symbol)

สัญลักษณ์	ความหมาย
%	ใช้จัดรูปแบบแสดงผลและเพิ่ม % ด้วยคำสั่ง '%%'
%c	ใช้จัดรูปแบบแสดงผลตัวอักษร
%d, %i	ใช้จัดรูปแบบแสดงผลเลขฐานสิบแบบมีเครื่องหมาย (8, 2, 0, 4)
%e	ใช้จัดรูปแบบแสดงผลเลขยกกำลังตัวอักษรเล็ก (110.15e+03)
%E	ใช้จัดรูปแบบแสดงผลเลขยกกำลังตัวอักษรใหญ่ (110.15E+03)
%f, %F	ใช้จัดรูปแบบแสดงผลตัวเลขทศนิยม
%g	ใช้จัดรูปแบบแสดงผลแบบสั้นของ %f และ %e
%G	ใช้จัดรูปแบบแสดงผลแบบสั้นของ %f และ %E
%o	ใช้จัดรูปแบบแสดงผลเลขฐานแปด
%s	ใช้จัดรูปแบบแสดงผลตัวอักษรหรือสตริงผ่านฟังก์ชัน str()
%u	ใช้จัดรูปแบบแสดงผลเลขฐานสิบแบบไม่มีเครื่องหมาย (8, 2, 0, 4)
%x	ใช้จัดรูปแบบแสดงผลเลขฐานสิบหกแสดงตัวอักษรเล็ก (43acd)
%X	ใช้จัดรูปแบบแสดงผลเลขฐานสิบหกแสดงตัวอักษรใหญ่ (43ACD)

โครงสร้างการจัดรูปแบบแสดงผลโดยใช้เครื่องหมาย {} และ (:) โดยเรียกใช้ฟังก์ชัน format()

```
print('Str1: {p0},, Str2: (p1), Str3: (p2)'  
      .format(p0, p1, p2))
```

โครงสร้างการจัดรูปแบบแสดงผลโดยใช้เครื่องหมาย {} และ (:) โดยเรียกใช้ฟังก์ชัน format() และการใช้ keyword อ้างอิงตำแหน่งอาร์กิวเมนต์

```
print('Str1: {k0},, Str2: (k1), Str3: (k2)'  
      .format(k0=v1, k1=v1, k2=v2))
```

#### ตัวอย่าง 4.14

การเปรียบเทียบการเขียนคำสั่งโปรแกรมจัดรูปแบบแสดงผลโดยใช้เครื่องหมาย % กับเครื่องหมาย { }

```
1 num1 = 4945; num2 = 45 # สร้างตัวแปรเป็นชนิดข้อมูลจำนวนเต็ม
2 result = num1/num2 # หารค่าตัวแปร num1 กับ num2 เก็บผลลัพธ์ไว้ใน result
3 print('จัดรูปแบบแสดงผลโดยใช้เครื่องหมาย % = ', '%f' %(result))
4         # ใช้เครื่องหมาย % จัดรูปแบบแสดงผล
5 print('จัดรูปแบบแสดงผลโดยใช้เครื่องหมาย {} = ', '{:f}'.format(result))
6         # ใช้เครื่องหมาย {} จัดรูปแบบแสดงผลร่วมกับฟังก์ชัน .format()
```

จัดรูปแบบแสดงผลโดยใช้เครื่องหมาย % = 109.888889  
จัดรูปแบบแสดงผลโดยใช้เครื่องหมาย{} = 109.888889



จากตัวอย่างเป็นการเปรียบเทียบการเขียนคำสั่งโปรแกรมการจัดรูปแบบแสดงผลโดยใช้เครื่องหมาย % และ {} ผลลัพธ์ที่ได้ออกมาแสดงผลได้เหมือนกันแต่จะมีวิธีการเรียกใช้งานที่แตกต่างกัน

#### ตัวอย่าง 4.15

การเขียนคำสั่งโปรแกรมจัดรูปแบบแสดงผลอ้างอิงตามตำแหน่งค่าอาร์กิวเม้นต์

```
1 n = 4945; o = 45; x = 1453; s = 'Python'
2 # ประกาศตัวแปร n, o, x เป็นชนิดข้อมูลจำนวนเต็ม และ s เป็นชนิดข้อมูลสตริง
3 print('จัดรูปแบบแสดงผลข้อความ = ', '%10s' % s)
4 # ให้เยื่องค่าตัวแปร s ไปทางขวารวม 10 ตัวอักษร
5 print('จัดรูปแบบแสดงผลเป็นเลขทศนิยม = ', '.3f' % n)
6 # กำหนดให้แสดงผลเป็นเลขทศนิยม 3 ตำแหน่ง
7 print('จัดรูปแบบแสดงผลเป็นเลขฐานแปด = ', '%o' % o)
8 # กำหนดให้แสดงผลเป็นเลขฐาน 8
9 print('จัดรูปแบบแสดงผลเป็นเลขฐานสิบหก = ', '%X' % x)
10 # กำหนดให้แสดงผลเป็นเลขฐาน 16
```

จัดรูปแบบแสดงผลข้อความ = Python  
จัดรูปแบบแสดงผลเป็นเลขทศนิยม = 4945.000  
จัดรูปแบบแสดงผลเป็นเลขฐานแปด = 55  
จัดรูปแบบแสดงผลเป็นเลขฐานสิบหก = 5AD



หากต้องการเขียนคำสั่งโปรแกรมจัดรูปแบบแสดงผลโดยใช้เครื่องหมาย () สามารถทำได้ ดังตัวอย่างต่อไปนี้

### ตัวอย่าง 4.16

การเขียนคำสั่งโปรแกรมจัดรูปแบบการแสดงผลโดยใช้เครื่องหมาย

```
1 n = 4945; o = 45; x = 1453; s = 'Python'
2 print('จัดรูปแบบแสดงผลข้อความ = ', '{:20s}'.format(s))
3 print('จัดรูปแบบแสดงผลเป็นเลขทศนิยม = ', '{:.3f}'.format(n))
4 print('จัดรูปแบบแสดงผลเป็นเลขฐานแปด = ', '{:o}'.format(o))
5 print('จัดรูปแบบแสดงผลเป็นเลขฐานสิบหก = ', '{:X}'.format(x))
6 print('แยกจำนวนตัวเลขด้วยเครื่องหมาย , = {:,}'.format(45609823465))'
```

จัดรูปแบบแสดงผลข้อความ = Python  
จัดรูปแบบแสดงผลเป็นเลขทศนิยม = 4945.000  
จัดรูปแบบแสดงผลเป็นเลขฐานแปด = 55  
จัดรูปแบบแสดงผลเป็นเลขฐานสิบหก = 5AD  
แยกจำนวนตัวเลขด้วยเครื่องหมาย , = 45, 609, 823 , 465



จากตัวอย่างการเขียนคำสั่งโปรแกรมในบรรทัดที่ 6 มีการใช้เครื่องหมาย (,) เข้ามาช่วยจัดแสดงผลโดยทำหน้าที่แบ่งตัวเลขออกเป็นกลุ่มละ 3 ตัว โดยเริ่มจัดกลุ่มจากข้างหลังมายังด้านหน้า

### ตัวอย่าง 4.17

การเขียนคำสั่งโปรแกรมจัดรูปแบบการแสดงผลอ้างอิงตามตำแหน่งค่าอาร์กิวเม้นต์  
เครื่องหมาย %

ด้วย

```
1 test1 = 23; test2 = 24.5; test3 = 30; subject = 'Programming'; grade =
   → 'B+'
2 print('สมชาย เรียนวิชา %s ได้คะแนน\n\
3 ทดสอบครั้งที่ 1 = %d\n\
4 ทดสอบครั้งที่ 2 = %.2f\n\
5 ทดสอบครั้งที่ 3 = %d\n\
6 ได้เกรด = %s' %(subject, test1, test2, test3, grade))
7 # แสดงผลค่าอาร์กิวเม้นต์ทั้งหมด (subject, test1, test2, test3, test4)
```

สมชาย เรียนวิชา Programming ได้คะแนน  
ทดสอบครั้งที่ 1 = 23  
ทดสอบครั้งที่ 2 = 24.50  
ทดสอบครั้งที่ 3 = 30  
ได้เกรด = B+



### ตัวอย่าง 4.18

การเขียนคำสั่งโปรแกรมจัดรูปแบบแสดงผล ด้วยฟังก์ชัน `format()` โดยอ้างอิงตำแหน่ง อาร์กิวเม้นต์

```
1 print('สมใจ เรียนวิชา {} ได้คะแนน\n\
2 ทดสอบครั้งที่ 1 = {}\\n\
3 ทดสอบครั้งที่ 2 = {:.2f}\\n\
4 ทดสอบครั้งที่ 3 = {:.2f}\\n\
5 ได้เกรด = {}' .format('Database System', 30, 25.50, 25.50, 'A'))
```

```
สมใจ เรียนวิชา Database System ได้คะแนน
ทดสอบครั้งที่ 1 = 30
ทดสอบครั้งที่ 2 = 25.50
ทดสอบครั้งที่ 3 = 25.50
ได้เกรด = A
```



จากตัวอย่างโปรแกรมใช้ฟังก์ชัน `format()` จัดรูปแบบแสดงผลโดยใช้เครื่องหมายปีกกา (`....`) แทรกเข้าไปในส่วนของข้อความ เพื่ออ้างอิงตำแหน่งของอาร์กิวเม้นต์ภายในเครื่องหมายวงเล็บหลัง ฟังก์ชัน `format()` จากตัวอย่างโปรแกรมมีอาร์กิวเม้นต์ทั้งหมด 5 ตัว และเมื่อเราอ้างอิงถึง อาร์กิวเม้นต์ต้องมีเครื่องหมายปีกกา 5 ตัว เช่นกัน ถ้ามีจำนวนไม่เท่ากันจะทำให้เกิดแจ้งเตือนข้อผิด พลาดขึ้น นอกจากนี้เรายังสามารถจัดแสดงผลตัวเลขได้เหมือนกับการใช้เครื่องหมายเปอร์เซ็นต์ `%` แสดง ตั้งตัวอย่างในบรรทัดที่ 3 และบรรทัดที่ 4 มีการกำหนดให้แสดงผลศูนย์สิบตำแหน่ง

### ตัวอย่าง 4.19

การเปลี่ยนคำสั่งโปรแกรมจัดรูปแบบการแสดงผล ด้วยฟังก์ชัน `format()` โดยอ้างอิงตำแหน่งคีย์เวิร์ด

```
1 text = 'สมใจ เรียนวิชา {s} ได้คะแนน\n\
2 ทดสอบครั้งที่ 1 = {T1}\n\
3 ทดสอบครั้งที่ 2 = {T2:.2f}\n\
4 ทดสอบครั้งที่ 3 = {T3:.2f}\n\
5 'ได้เกรด = {G}'\n\
6 print(text.format(s='Database System', T1 = 30, T2 = 25.50, T3 =
    → 25.50, G ='A'))
```

```
สมใจ เรียนวิชา Database System ได้คะแนน
ทดสอบครั้งที่ 1 = 30
ทดสอบครั้งที่ 2 = 25.50
ทดสอบครั้งที่ 3 = 25.50
ได้เกรด = A
```



จากตัวอย่างโปรแกรมใช้วิธีการจัดรูปแบบการแสดงผลโดยการอ้างอิงตำแหน่งคีย์เวิร์ด ลักษณะการอ้างอิงแบบนี้ภายในเครื่องหมายวงเล็บปีกภาษาจะต้องระบุชื่อคีย์เวิร์ดที่ต้องการแสดงผลข้อมูลที่เก็บค่าไว้และภายในเครื่องหมายวงเล็บปีกภาษาสามารถจัดรูปแบบการแสดงผลตัวเลขได้เหมือนกับวิธีการอ้างอิงอาร์กิวเมนต์

#### 4.4.2 การจัดรูปแบบแสดงผลด้วย f-string

f-string เป็นวิธีการจัดรูปแบบการแสดงผลในภาษาไพธอนอีกรูปแบบหนึ่งที่ใช้งานง่ายและสะดวก โดยถูกเพิ่มเข้ามาให้ใช้งานได้ตั้งแต่ไพร่อนเวอร์ชัน 3.6 การแสดงผลจะมี f ข้างหน้าเครื่องหมาย '...' ส่วนที่นำมาแสดงผลจะอยู่ภายใต้เครื่องหมายปีกกา { ... } ซึ่งภายในจะเป็นตัวแปร หรือการดำเนินการต่างๆ ก็ได้มีรูปแบบวิธีการใช้งานดังนี้

โครงสร้างคำสั่งการจัดรูปแบบการแสดงผลด้วย f-string

```
print(f'... {var1[:format]} ... {varn[:format]}')
```

##### ตัวอย่าง 4.20

การเขียนคำสั่งโปรแกรมจัดรูปแบบการแสดงผลด้วย f-string

```
1 r = float(input('กรอกรัศมีของวงกลม: '))
2 pi = 3.14
3 area = pi * r * r
4 perimeter = 2 * pi * r
5 print(f'พื้นที่ของวงกลมนี้ คือ {area} ตารางหน่วย')
6 print(f'ความยาวเส้นรอบวงของวงกลมนี้ คือ {perimeter} หน่วย')
```

กรอกรัศมีของวงกลม: 3.5

พื้นที่ของวงกลมนี้ คือ 38.465 ตารางหน่วย

ความยาวเส้นรอบวงของวงกลมนี้ คือ 21.98 หน่วย



##### ตัวอย่าง 4.21

การเขียนคำสั่งโปรแกรมจัดรูปแบบการแสดงผลด้วย f-string

```
1 text = 'Hello World'
2 print(f'Length of \'{text}\'' is {len(text)})
```

Length of 'Hello World' is 11



### ตัวอย่าง 4.22

การเขียนคำสั่งโปรแกรมจัดรูปแบบการแสดงผลด้วย f-string

```
1 f = int(input('Enter first score: '))
2 s = int(input('Enter second score: '))
3 t = int(input('Enter third score: '))
4 total = f + s + t
5 print(f'Total Score = {f} + {s} + {t} = {total}')
```

```
Enter first score: 25
Enter second score: 15
Enter third score: 12
Total Score = 25 + 15 + 12 = 52
```



### ตัวอย่าง 4.23

การเขียนคำสั่งโปรแกรมจัดรูปแบบการแสดงผลด้วย f-string

```
1 s = 'Database System'
2 T1 = 30; T2 = 25.50; T3 = 25.50
3 G = 'A'
4 print(f'สมใจเรียนวิชา {s} ได้คะแนน\n \
5 ทดสอบครั้งที่ 1 = {T1}\n \
6 ทดสอบครั้งที่ 2 = {T2:.2f}\n \
7 ทดสอบครั้งที่ 3 = {T3:.4f}\n \
8 ได้เกรด = {G}' )
```

```
สมใจเรียนวิชา Database System ได้คะแนน
ทดสอบครั้งที่ 1 = 30
ทดสอบครั้งที่ 2 = 25.50
ทดสอบครั้งที่ 3 = 25.5000
ได้เกรด = A
```



สัญลักษณ์	การแสดงผล
<	ใช้จัดรูปแบบแสดงผลให้อยู่ทางด้านซ้ายมือ
>	ใช้จัดรูปแบบแสดงผลให้อยู่ทางด้านขวามือ
=	ใช้กำหนดช่องว่างหน้าข้อความ และใช้ได้กับเฉพาะตัวเลขเท่านั้น
^	ใช้จัดรูปแบบให้อยู่กึ่งกลาง

ตาราง 4.2: สัญลักษณ์กำหนดตำแหน่งรูปแบบแสดงผลข้อความ

#### 4.4.3 การจัดตำแหน่งแสดงผลข้อความ

โดยปกติข้อความที่แสดงผลออกมาจะถูกกำหนดให้อยู่ทางซ้ายมือ ถ้ามีหลายบรรทัดและต้องการให้แสดงแต่ละคอลัมน์ให้ตรงกันจะเกิดปัญหาขึ้น เนื่องจากขนาดข้อความในแต่ละคอลัมน์ไม่เท่ากัน ดังนั้นจึงควรใช้สัญลักษณ์เข้ามากำหนดค่าในการแสดงผลตำแหน่ง ตาราง 4.2 แสดงสัญลักษณ์กำหนดตำแหน่งระยะห่างข้อความที่ต้องการแสดงผล หากต้องการระยะห่างแต่ละคอลัมน์ให้ใส่ตัวเลขต่อท้ายหลังสัญลักษณ์

##### ตัวอย่าง 4.24

###### การจัดตำแหน่งแสดงผลข้อความ

```

1 f_name = 'Joe'
2 l_name = 'Biden'
3 code = '1234'
4 print('{:<10s} {:^15s} {}'.format('First Name', 'Last name', 'Code'))
5 print('{:<10s} {:^15s} {}'.format(f_name, l_name, code))

```

First Name	Last name	Code
Joe	Biden	1234



จากตัวอย่าง 4.24 เราอธิบายการทำงานโปรแกรมได้ดังนี้

- บรรทัดที่ 1 สร้างตัวแปร `f_name` สำหรับเก็บชื่อ
- บรรทัดที่ 2 สร้างตัวแปร `l_name` สำหรับเก็บนามสกุล
- บรรทัดที่ 3 สร้างตัวแปร `code` เก็บรหัส
- บรรทัดที่ 4 จัดรูปแบบแสดงผลด้วยฟังก์ชัน `.format()` ซึ่งเป็นการกำหนดหัวคอลัมน์ได้แก่ อาร์กิวเมนต์ `f_name` กำหนดให้ชิดซ้ายและให้จัดรูปแบบแสดงผลแบบสตริง อาร์กิวเมนต์

สัญลักษณ์	การจัดรูปแบบชนิดข้อมูลจำนวนเต็ม
+	ใช้จัดรูปแบบแสดงผลให้มีเครื่องหมาย + ออยู่ด้านหน้า
-	ใช้จัดรูปแบบแสดงผลให้มีเครื่องหมาย - ออยู่ด้านหน้า
space	ใช้กำหนดช่องว่างหน้าข้อความ

ตาราง 4.3: สัญลักษณ์กำหนดเครื่องหมายชนิดข้อมูลจำนวนเต็ม

`l_name` กำหนดให้อยู่กึ่งกลางและให้มีระยะห่าง 15 ตัวอักษรพร้อมจัดรูปแบบแสดงผลแบบสตริง ส่วนอาร์กิวเมนต์ `code` จัดตำแหน่งให้ชิดขวาและจัดรูปแบบแสดงผลแบบสตริง

- บรรทัดที่ 5 จัดรูปแบบแสดงผลด้วยฟังก์ชัน `.format()` โดยการนำเอาตัวแปร `f_name`, `l_name` และ `code` มาแสดงผลในรูปแบบสตริงและกำหนดระยะห่าง

เมื่อต้องการจัดรูปแบบแสดงผลให้มีเครื่องหมาย `+` และ `-` เครื่องหมายนำหน้าจำนวนเต็มบวกหรือลบ หรือต้องการให้มีช่องว่างนำหน้าจำนวนเต็มทั้งสอง สามารถนำเครื่องหมายในตาราง 4.3 เข้ามาช่วยจัดรูปแบบแสดงผลได้

#### ตัวอย่าง 4.25

การจัดรูปแบบแสดงผลเครื่องหมายหน้าชนิดข้อมูลจำนวนเต็ม

```

1 x = 987.485; y = -225.154
2 print('{:+f}; {:.+f}'.format(x, y))
3 # กำหนดให้มีเครื่องหมาย + นำอยู่ด้านหน้า
4 print('{:-.3f}; {:-.3f}'.format(x, y))
5 # กำหนดให้มีจำนวนทศนิยม 3 ตำแหน่ง และเครื่องหมาย - ออยู่ด้านหน้า
6 print('{:.3f}; {:.3f}'.format(x, y))
7 # กำหนดให้มีจำนวนทศนิยม 3 ตำแหน่ง และให้มีช่องว่างด้านหน้า

```

```
+987.485000; -225.154000
987.485; -225.154
987.485; -225.154
```



## 4.5 เมธอดที่ใช้กับชนิดข้อมูลกลุ่มอักขระหรือสตริง

ในส่วนนี้จะยกตัวอย่างบางเมธอดที่นำมาใช้งานสำหรับจัดการกับชนิดข้อมูลสตริง หากต้องการดูข้อมูลรายละเอียดเพิ่มเติมของเมธอดอื่น ๆ ที่นำมาใช้งานร่วมกับชนิดข้อมูลสตริง ให้ใช้คำสั่ง `help(str)` เพื่อดูวิธีการเรียกใช้งาน

### 4.5.1 เมธอดการเปลี่ยนลักษณะสตริง

เมธอดในกลุ่มนี้จะทำหน้าที่เปลี่ยนลักษณะสตริง เช่น จากตัวอักษรพิมพ์เล็กให้เป็นพิมพ์ใหญ่ หรือจากตัวอักษรพิมพ์ใหญ่ให้เป็นพิมพ์เล็ก หรือจัดการแสดงผลสลับตัวอักษรพิมพ์ใหญ่และพิมพ์เล็ก

#### ตัวอย่าง 4.26

การเขียนคำสั่งโปรแกรมเปลี่ยนลักษณะสตริง

```
1 str = 'python is the best Programming Language.'
2 print('เปลี่ยนตัวอักษรแรกให้เป็นตัวพิมพ์ใหญ่ = ', str.capitalize())
3 print('เปลี่ยนตัวอักษรให้เป็นพิมพ์เล็กทั้งหมด = ', str.lower())
4 print('เปลี่ยนตัวอักษรแรกของคำให้เป็นตัวพิมพ์ใหญ่ = ', str.title())
5 print('เปลี่ยนตัวอักษรให้เป็นตัวพิมพ์ใหญ่ทั้งหมด = ', str.upper())
6 print('สลับตัวอักษรตัวพิมพ์เล็กและใหญ่ในประโยค = ', str.swapcase())
```

เปลี่ยนตัวอักษรแรกให้เป็นตัวพิมพ์ใหญ่ = Python is the best programming language.▶

เปลี่ยนตัวอักษรให้เป็นพิมพ์เล็กทั้งหมด = python is the best programming language.

เปลี่ยนตัวอักษรแรกของคำให้เป็นตัวพิมพ์ใหญ่ = Python Is The Best Programming

↪ Language.

เปลี่ยนตัวอักษรให้เป็นตัวพิมพ์ใหญ่ทั้งหมด = PYTHON IS THE BEST PROGRAMMING LANGUAGE.

สลับตัวอักษรตัวพิมพ์เล็กและใหญ่ในประโยค = PYTHON IS THE BEST pROGRAMMING lANGUAGE.

### 4.5.2 เมธอดการจัดตำแหน่งสตริง

เมธอดในกลุ่มนี้ทำหน้าที่จัดวางตำแหน่งของสตริงให้อยู่ด้านซ้าย ขวา หรือตรงกลางจากค่าที่กำหนด ตำแหน่งแสดงผล

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>capitalize()</code>	ใช้สำหรับแปลงตัวอักษรแรกของประโยคให้เป็นตัวอักษรพิมพ์ใหญ่	<code>str.capitalize()</code>
<code>lower()</code>	ใช้เปลี่ยนสตริงให้เป็นตัวอักษรพิมพ์เล็กทั้งหมด	<code>str.lower()</code>
<code>upper()</code>	ใช้สำหรับเปลี่ยนสตริงเป็นตัวอักษรพิมพ์ใหญ่ทั้งหมด	<code>str.upper()</code>
<code>title()</code>	ใช้สำหรับเปลี่ยนตัวอักษรแรกของแต่ละคำให้เป็นตัวอักษรพิมพ์ใหญ่ (เช่น The Best)	<code>str.title()</code>

ตาราง 4.4: เมธอดสำหรับเปลี่ยนลักษณะสตริง เมื่อ `str` คือ ตัวแปรชนิดข้อมูลสตริง

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>center()</code>	ใช้สำหรับจัดข้อความให้อยู่ตรงกลาง	<code>str.center(width, [fillchar])</code>
<code>ljust()</code>	ใช้สำหรับจัดสตริงให้ซิดซ้าย	<code>str.ljust(width, [fillchars])</code>
<code>rjust()</code>	ใช้สำหรับจัดสตริงให้ซิดขวา	<code>str.rjust(length, [fillchars])</code>

ตาราง 4.5: เมธอดจัดตำแหน่งสตริง เมื่อ `str` คือ ตัวแปรเก็บชนิดข้อมูลสตริง `width` คือ กำหนดขนาดความกว้างที่รวมข้อความ และ `fillchar` คือ ตัวอักษรหรือสัญลักษณ์ที่เว้นให้เต็มความยาว ปกติจะเป็นช่องว่าง จะระบุหรือไม่ก็ได้

### ตัวอย่าง 4.27

การเขียนคำสั่งโปรแกรมตรวจสอบตัวเลขภายในชนิดข้อมูลสตริงด้วยเมธอด `isalnum()`

```
1 str = 'Programming'
2 print('จัดตำแหน่งให้อ瑜ด้านซ้าย = ', str.ljust(20))
3 print('จัดตำแหน่งให้อ瑜ด้านกลาง = ', str.center(20))
4 print('จัดตำแหน่งให้อ瑜ด้านขวา = ', str.rjust(20, '-'))
```

จัดตำแหน่งให้อ瑜ด้านซ้าย = Programming  
จัดตำแหน่งให้อ瑜ด้านกลาง = Programming  
จัดตำแหน่งให้อ瑜ด้านขวา = -----Programming



### 4.5.3 เมธอดสำหรับตรวจสอบสตริง

เมธอดในกลุ่มนี้ทำหน้าที่ตรวจสอบข้อความหรือสตริงว่าประกอบด้วยส่วนประกอบอะไรบ้าง

### ตัวอย่าง 4.28

การเขียนคำสั่งโปรแกรมตรวจสอบสตริง

```
1 str1 = 'Programming'; str2 = 'Python2019'
2 str3 = 'Python programming Language'
3 print('ประกอบด้วยตัวอักษรทั้งหมดหรือไม่ = ', str1.isalpha())
4 print('ประกอบด้วยตัวอักษรหรือตัวเลขทั้งหมดหรือไม่ = ', str2.isalnum())
5 print('แต่ละคำชื่นต้นตัวอักษรพิมพ์ใหญ่ทั้งหมดหรือไม่ = ', str3.istitle())
```

ประกอบด้วยตัวอักษรทั้งหมดหรือไม่ = True  
ประกอบด้วยตัวอักษรหรือตัวเลขทั้งหมดหรือไม่ = True  
แต่ละคำชื่นต้นตัวอักษรพิมพ์ใหญ่ทั้งหมดหรือไม่ = False



เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>isalnum()</code>	คืนค่า True ถ้าสตริงประกอบด้วยตัวอักษรปนกับตัวเลข หรือแค่ตัวอักษรตัวเลขอย่างใดอย่างหนึ่งเพียงอย่างเดียว ถ้ามีสัญลักษณ์อื่น หรือซึ่งว่างจะคืนค่า False	<code>str.isalnum()</code>
<code>isalpha()</code>	คืนค่า True ถ้าสตริงประกอบด้วยตัวอักษรทั้งหมด ถ้ามีสัญลักษณ์ตัวเลข หรือซึ่งว่างจะคืนค่า False	<code>str.isalpha()</code>
<code>isdigit()</code>	คืนค่า True ถ้าสตริงประกอบด้วยตัวเลขทั้งหมด ถ้ามีตัวอักษร สัญลักษณ์ หรือซึ่งว่าง จะคืนค่า False	<code>str.isdigit()</code>
<code>isdecimal()</code>	ใช้สำหรับตรวจสอบตัวเลข แสดงผลลัพธ์ เมื่อ้อนเมธอด isdigit()	<code>str.isdecimal()</code>
<code>isnumber()</code>	ใช้สำหรับตรวจสอบตัวเลข แสดงผลลัพธ์ เมื่อ้อนเมธอด isdigit()	<code>str.isnumber()</code>
<code>islower()</code>	คืนค่า True ถ้าสตริงประกอบด้วยตัวพิมพ์เล็กทั้งหมด ถ้ามีตัวอักษรพิมพ์ใหญ่ประกอบด้วย จะคืนค่า False	<code>str.islower()</code>
<code>isupper()</code>	คืนค่า True ถ้าสตริงประกอบด้วยตัวพิมพ์ใหญ่ทั้งหมด ถ้ามีตัวอักษรพิมพ์เล็กประกอบด้วย จะคืนค่า False	<code>str.isupper()</code>
<code>istitle()</code>	คืนค่า True ถ้าตัวอักษรแรกแต่ละคำในสตริงเป็นตัวอักษรพิมพ์ใหญ่ ถ้าขึ้นต้นด้วยตัวอักษรพิมพ์เล็ก จะคืนค่า False	<code>str.istitle()</code>
<code>isspace()</code>	คืนค่า True ถ้าสตริงเก็บค่าว่าง ถ้ามีข้อมูลอยู่ จะคืนค่า False	<code>str.space()</code>

ตาราง 4.6: เมธอดใช้สำหรับตรวจสอบสตริง

#### 4.5.4 เมธอดสำหรับการนับค่า ค้นหา และแก้ไขสตริง

เมธอดในกลุ่มนี้ทำหน้าที่นับตัวอักษรหรือคำที่มีอยู่ในสตริง หรือค้นหาคำที่ต้องการได้ระบุ รวมไปถึง เมธอดที่นำมาใช้สำหรับแก้ไขค่าข้อมูลในสตริง

##### ตัวอย่าง 4.29

การเขียนคำสั่งโปรแกรมการตรวจสอบ การแสดง และการนับตำแหน่ง

```
1 str1 = 'Python is the best programming language.'
2 print('คำสุดท้ายในประโยคคือ language. = ', str1.endswith('language.'))
3 print('language. เริ่มจากตำแหน่งที่ = ', str1.index('language.'))
4 print('มีตัวอักษร e ทั้งหมด = ', str1.count('e'))
```

คำสุดท้ายในประโยคคือ language. = True  
language. เริ่มจากตำแหน่งที่ = 31  
มีตัวอักษร e ทั้งหมด = 3



#### 4.5.5 เมธอดสำหรับรวม แยก และตัดสตริง

เมธอดในกลุ่มนี้ทำหน้าทารวมสตริงเข้าด้วยกันหรือนำสัญลักษณ์เข้ามาใช้งานร่วมกับสตริง หรือใช้ เมธอดทำการตัดตัวอักษร ซึ่งว่าง ออกจากสตริง

##### ตัวอย่าง 4.30

การเขียนคำสั่งโปรแกรม การดำเนินการเปรียบเทียบกับชนิดข้อมูลสตริง

```
1 str_1 = '__Python Programming__'
2 str_2 = 'Python is the best Programming.'
3 str_rst = str_1.rstrip('_')
4 str_rsp = str_1.rsplit('__')
5 str_sp = str_2.split(' ')
6 print(str_rst)
7 print(str_rsp)
8 print(str_sp)
```

\_\_Python Programming  
['\_\_Python Progra', '', 'ing\_\_']  
['Python', 'is', 'the', 'best', 'Programming.']}



เมธอด	ความหมาย	รูปแบบการใช้งาน
count()	ใช้สำหรับนับจำนวนตัวอักษรหรือคำในสตริง	<code>str.count(x, start, stop)</code>
endswith()	คืนค่า True ถ้าลงท้ายด้วยตัวอักษรที่ระบุ ถ้าไม่ตรงกับที่ระบุจะคืนค่า False	<code>str.endswith(x, start, stop)</code>
find()	ใช้สำหรับค้นหาตำแหน่งอักษรที่ระบุในสตริง ถ้าไม่พบจะคืนค่าเป็น -1	<code>str.find(x, start, stop)</code>
index()	ใช้สำหรับตำแหน่งข้อความในสตริง	<code>str.index(x, start, stop)</code>
startswith()	คืนค่า True ถ้าขึ้นต้นด้วยตัวอักษรที่ระบุ ถ้าไม่ตรงกับที่ระบุ จะคืนค่า False	<code>str.startswith(x, start, stop)</code>
rfind()	ใช้สำหรับค้นหาตำแหน่งอักษรที่ระบุในสตริง จะแสดงตำแหน่งสุดท้ายที่ค้นพบ ถ้าไม่พบจะคืนค่าเป็น -1	<code>str.rfind(x, start, stop)</code>
replace()	ใช้สำหรับแทนที่ข้อความที่ต้องการตามที่ระบุ	<code>str.replace(old, new [, count])</code>

ตาราง 4.7: เมธอดที่ใช้สำหรับการนับ ค้นหา และแก้ไขค่าสตริง เมื่อ `str` คือ ตัวแปรชนิดข้อมูลสตริง ที่ต้องการนับจำนวนตัวอักษร `x` คือ ตัวอักษรหรือคำที่ระบุ `start` คือ ตำแหน่งเริ่มต้น `stop` คือ ตำแหน่งสุดท้าย `old` คือ ข้อความเดิม `new` คือ ข้อความใหม่ และ `count` คือ จำนวนครั้งที่ต้องการแทนที่จะระบุหรือไม่ก็ได้

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>join()</code>	ใช้สำหรับเชื่อมสตริงที่เป็นชนิดข้อมูลแบบเรียงลำดับให้ต่อกัน	<code>str.join(seq)</code>
<code>lstrip()</code>	ใช้สำหรับลบช่องว่างหรือตัวอักษรทางด้านซ้ายของสตริง	<code>str.lstrip([chars])</code>
<code>partition()</code>	ใช้สำหรับแยกสตริงผลลัพธ์ที่ได้เก็บอยู่ในรูปแบบชนิดข้อมูลทุกเพิล	<code>str.partition(x)</code>
<code>rstrip()</code>	ใช้สำหรับตัดช่องว่างหรือตัวอักษรทางด้านหลัง(ด้านขวา)ของสตริง	<code>str.rstrip([chars])</code>
<code>split()</code>	ใช้สำหรับแยกข้อความผลลัพธ์ที่ได้อยู่ในรูปแบบชนิดข้อมูลลิสต์	<code>str.split(sep)</code>
<code>strip()</code>	ใช้สำหรับลบข้อความหรือตัวอักษรและช่องว่างทั้งด้านหน้าและด้านหลังสตริง	<code>str.strip([chars])</code>

ตาราง 4.8: เมธอดที่ใช้สำหรับรวม แยก และตัดสตริง เมื่อ `str` คือ ตัวแปรเก็บชนิดข้อมูลสตริงที่ต้องการนับจำนวนตัวอักษร, `seq` คือ ชนิดข้อมูลแบบเรียงลำดับที่ต้องการเชื่อมต่อกัน, `chars` คือ ตัวอักษรที่ต้องการลบ ค่าปกติเป็นช่องว่าง, `x` คือ ข้อความหรือเครื่องหมายที่กำหนดให้แยกสตริง และ `sep` คือ สัญลักษณ์ที่กำหนดให้แยกข้อความ ค่าปกติเป็นช่องว่าง

## 4.6 การดำเนินการกับชนิดข้อมูลอักขระหรือสตริง

สามารถนำเอาตัวดำเนินการต่าง ๆ มาใช้กับชนิดข้อมูลอักขระหรือสตริงได้เหมือนกับชนิดข้อมูลอื่น ๆ เช่น การเปรียบเทียบ การเขื่อม การตรวจสอบ การมีอยู่หรือไม่มีอยู่ เป็นต้น แสดงดังตัวอย่างต่อไปนี้

### ตัวอย่าง 4.31

การเขียนคำสั่งโปรแกรม การดำเนินการเปรียบเทียบกับชนิดข้อมูลสตริง

```
1 str1 = 'Hello Python'; str2 = 'Programming'; str3 = 'Programming'  
2 print('str1 < str2 หรือไม่ =', str1 < str2)  
3 print('str1 > str2 หรือไม่ =', str1 > str2)  
4 print('str1 >= str2 หรือไม่ =', str1 >= str2)  
5 print('str1 != str2 หรือไม่ =', str1 != str2)  
6 print('str2 == str3 หรือไม่ =', str2 == str3)
```

```
str1 < str2 หรือไม่ = True  
str1 > str2 หรือไม่ = False  
str1 >= str2 หรือไม่ = False  
str1 != str2 หรือไม่ = True  
str2 == str3 หรือไม่ = True
```



นอกจากนี้ยังสามารถใช้ตัวดำเนินการ **in** และ **not in** ตรวจสอบว่ามีหรือไม่มีตัวอักขระที่ระบุในข้อความผลลัพธ์ที่ได้จะเป็น **True** หรือ **False** แสดงดังตัวอย่างต่อไปนี้

### ตัวอย่าง 4.32

การเขียนคำสั่งโปรแกรม การดำเนินการตรวจสอบชนิดข้อมูลสตริง

```
1 str1 = 'Python Programming'
2 str2 = 'อักขระหรือสตริง'
3
4 # ตรวจสอบตัวอักษร i ในค่าตัวแปร str1
5 print('มี i อยู่ใน str1 หรือไม่ = ', 'i' in str1)
6
7 # ตรวจสอบตัวอักษร A ในค่าตัวแปร str1
8 print('มี A อยู่ใน str1 หรือไม่ = ', 'A' in str1)
9
10 # ตรวจสอบตัวอักษร w ในค่าตัวแปร str2
11 print('มี w อยู่ใน str2 หรือไม่ = ', 'w' in str2)
12
13 # ตรวจสอบ 'สตริง' ในค่าตัวแปร str2
14 print('มี ' + 'สตริง' + ' อยู่ใน str2 หรือไม่ = ', 'สตริง' in str2)
```

มี i อยู่ใน str1 หรือไม่ = True  
มี A อยู่ใน str1 หรือไม่ = False  
มี w อยู่ใน str2 หรือไม่ = False  
มี 'สตริง' อยู่ใน str2 หรือไม่ = True



## สรุปก่อนจบบท

ชนิดข้อมูลล้วนอักขระหรือสตริงคือ ตัวอักษรที่นำมาเรียงต่อกันเป็นข้อความหรือประโยค เมื่อประกาศ ตัวแปรเก็บชนิดข้อมูลนี้จะอยู่ในเครื่องหมาย Single Quote ('...') หรือเครื่องหมาย Double Quote ("...") ซึ่งในภาษาไพธอนใช้ได้ทั้งสองเครื่องหมาย ในบทนี้เราได้เรียนรู้วิธีการประกาศชนิดข้อมูลสตริง การเข้าถึงตำแหน่งสตริง การจัดรูปแบบข้อความ รวมไปถึงสกุลកซ์แม่และรหัสพิเศษที่นำมาใช้ควบคุมการแสดงผลลัพธ์และได้แนะนำเมธอดต่าง ๆ ที่นำมาใช้งานร่วมกับชนิดข้อมูลสตริง

## แบบฝึกหัด

- กำหนดให้

```
alphabet = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'
```

ให้เขียนคำสั่งโปรแกรมแสดงผลเฉพาะสระภาษาอังกฤษและแสดงตำแหน่งที่อยู่ของสระแต่ละตัว

- จากคำสั่งโปรแกรมต่อไปนี้

```
alphabet = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'  
x1 = alphabet[?:?]  
x2 = alphabet[?:?:?]  
x3 = alphabet[?:?:?]  
print(f'x1 = {x1}, x2 = {x2}, x3 = {x3}')
```

จงเติมตัวเลขที่ถูกต้องลงในตำแหน่ง **?** เพื่อให้โปรแกรมแสดงผลลัพธ์เป็น

x1 = FGHIJ, x2 = LNPRTVX, x3 = ZWTQ

- จงเขียนโปรแกรมรับข้อมูลผ่านทางแป้นพิมพ์ และจัดรูปแบบแสดงผลตามที่กำหนดให้

Sent from	To
Name _____	Name _____
Address _____	Address _____
Zip code _____	Zip code _____

4. จงเขียนโปรแกรมเพื่อรับ ชื่อเต็ม (Full Name) ทางแป้นพิมพ์โดยให้รับค่าด้วยตัวแปรตัวเดียว  
เท่านั้น โดยใช้การเว้นวรรคเพื่อแยกชื่อแรกและนามสกุล จำนวนนี้ให้ทำการแยกชื่อแรก และ<sup>ชื่อ</sup> นามสกุลและแสดงจำนวนอักขระของชื่อแรกและนามสกุลดังตัวอย่างที่กำหนดให้

```
Enter the full name: Tony Woodsome
-----
Item      |   Value    | String Length
-----
First Name |  Tony      |   4
Last Name  | Woodsome  |   8
-----
```

5. จากโปรแกรมในข้อ 4 จงแก้ไขให้โปรแกรมรองรับเมื่อชื่อเต็มอาจประกอบไปด้วย ชื่อแรก ชื่อกลาง และนามสกุล และในกรณีที่ไม่มีชื่อกลางให้โปรแกรมแสดงผลลัพธ์ดังตัวอย่างที่กำหนดให้

```
Enter the full name: Tony Sinatra Woodsome
-----
Item      |   Value    | String Length
-----
First Name |  Tony      |   4
Middle Name | Sinatra  |   7
Last Name  | Woodsome  |   8
-----
```

```
Enter the full name: Tony Woodsome
-----
Item      |   Value    | String Length
-----
First Name |  Tony      |   4
Middle Name | -         |   0
Last Name  | Woodsome  |   8
-----
```

## บทที่ 5

# โครงสร้างข้อมูลภาษาโปรแกรมไพธอน

ในบทนี้เราจะได้รู้จักกับโครงสร้างข้อมูลภาษาโปรแกรมไพธอน (Python Data Structure) ซึ่งประกอบด้วยข้อมูลชนิดลิสต์ (List) ข้อมูลชนิดทูเพล (Tuple) ข้อมูลชนิดดิกชันนารี (Dictionary) และข้อมูลชนิดเซต (Set) ที่ใช้จัดเก็บข้อมูลแบบลำดับ (Sequence) โดยข้อมูลชนิดเหล่านี้สามารถเก็บข้อมูลต่างประเภทกันได้ ทั้งข้อมูลชนิดจำนวนเต็ม และข้อมูลชนิดสตริง นอกจากนี้ยังสามารถที่จะเก็บข้อมูลชนิดต่างชนิดซ้อนกัน เช่น ลิสต์ซ้อนลิสต์ ทูเพลซ้อนทูเพล หรือลิสต์ซ่อนทูเพลก็ได้ แต่ข้อมูลชนิดเหล่านี้อาจจะมีความแตกต่างกันในการเพิ่ม ลบ แก้ไขข้อมูล

### 5.1 รู้จักกับข้อมูลชนิดลิสต์

ข้อมูลชนิดลิสต์ (List) คล้ายกับข้อมูลอาร์เรย์ในภาษาอื่น ๆ การประกาศสร้างตัวแปรขึ้นมาเก็บข้อมูลชนิดลิสต์ไม่แตกต่างจากการสร้างตัวแปรข้อมูลชนิดอื่น ๆ แต่จะใช้เครื่องหมาย `[ ... ]` จัดเก็บข้อมูล และใช้เครื่องหมาย Comma (,) คั่นระหว่างสมาชิกในลิสต์ หากเป็นข้อมูลชนิดสตริงให้ใส่เครื่องหมาย '`...`' หรือ "`...`" ครอบ นอกจากนี้สมาชิกใน List จะสามารถประกอบด้วยตัวแพรหหลายชนิดรวมกันได้ เช่น จำนวนเต็ม, ศศนิยม, สตริง พิจารณาจากตัวอย่างต่อไปนี้

#### ตัวอย่าง 5.1

##### การประกาศตัวแปรข้อมูลชนิดลิสต์

```
1 books_lst = ['Python', 'Java', 'C', 'C++', 'C#']
2 numbers_lst = [1, 2, 3, 4, 5, 6]
3 book_num_lst = [1, 'Python', 2, 'Java', 3 , 'C']
4 print('รายชื่อหนังสือใน books_lst = ', books_lst)
5 print('สมาชิกใน numbers_lst = ', numbers_lst)
6 print('สมาชิกใน book_num_lst = ', book_num_lst)
```

รายชื่อหนังสือใน books\_1st = ['Python', 'Java', 'C', 'C++', 'C#']  
สมาชิกใน numbers\_1st = [1, 2, 3, 4, 5, 6]  
สมาชิกใน book\_num\_1st = [1, 'Python', 2, 'Java', 3, 'C']



### 5.1.1 การเข้าถึงตำแหน่งข้อมูลของข้อมูลชนิดลิสต์

เราสามารถเข้าถึงตำแหน่งข้อมูลที่เก็บอยู่ภายในข้อมูลชนิดลิสต์ได้สองทางคือ จากทางด้านหน้าและจากทางด้านท้ายสุด ด้วยการระบุตำแหน่ง (Index) หากเข้าถึงข้อมูลจากทางด้านหน้าให้เริ่มนับจากตำแหน่งที่ 0 ถ้าเข้าถึงตำแหน่งข้อมูลจากทางด้านท้ายสุดจะเริ่มนับจากตำแหน่งที่ -1 มีรูปแบบการเข้าถึงตำแหน่งดังนี้

```
var[start : stop : step]
```

- |              |   |
|--------------|---|
| <b>start</b> | ขอบเขตที่ต้องการแสดงผล มีได้มากกว่า 1 ขอบเขต          |
| <b>stop</b>  | ส่วนที่ใช้แยกแต่ละขอบเขตออกจากกัน ค่าปกติเป็นช่องว่าง |
| <b>step</b>  | การกำหนดแสดงผลขอบเขต ค่าปกติเป็นการขึ้นบรรทัดใหม่     |

#### ตัวอย่าง 5.2

การเขียนคำสั่งโปรแกรมระบุตำแหน่งข้อมูลที่เก็บอยู่ในข้อมูลชนิดลิสต์

```
1 books_lst = ['Python', 'Java', 'C', 'C++', 'C#', 'Scala', 'PHP']
2 print('ตำแหน่ง -5 ใน books_lst คือ', books_lst[-5])
3 print('ตำแหน่ง 2-6 ใน books_lst คือ', books_lst[2:5])
4 print('ตำแหน่ง 0-3 ใน books_lst คือ', books_lst[:4])
5 print('แสดงข้อมูลตำแหน่งที่ 1-5 โดยข้ามครั้งละ 2 ตำแหน่ง= ', books_lst[1:6:2])
6 print('แสดงข้อมูลใน books_lst โดยข้ามครั้งละ 2 ตำแหน่ง= ', books_lst[::-2])
```

ตำแหน่ง -5 ใน books\_lst คือ C  
ตำแหน่ง 2-6 ใน books\_lst คือ ['C', 'C++', 'C#']  
ตำแหน่ง 0-3 ใน books\_lst คือ ['Python', 'Java', 'C', 'C++']  
แสดงข้อมูลตำแหน่งที่ 1-5 โดยข้ามครั้งละ 2 ตำแหน่ง= ['Java', 'C++', 'Scala']  
แสดงข้อมูลใน books\_lst โดยข้ามครั้งละ 2 ตำแหน่ง= ['Python', 'C', 'C#', 'PHP']

จากตัวอย่างโปรแกรมได้แสดงวิธีการเขียนคำสั่งโปรแกรม การเข้าถึงตำแหน่งข้อมูลในข้อมูลชนิดลิสต์ เราไม่จำเป็นต้องระบุตำแหน่งก็ได้ เมื่อต้องการเข้าถึงจากตำแหน่งเริ่มต้นจนถึงตำแหน่งสุดท้าย แต่ต้องคั่นด้วยเครื่องหมาย colon(:)

### 5.1.2 เมธอดและฟังก์ชันที่ใช้กับข้อมูลชนิดลิสต์

ข้อมูลชนิดลิสต์มีเมธอดให้เรียกใช้งานจำนวนมาก หากเราต้องการรายละเอียดเพิ่มเกี่ยวกับการใช้งานสามารถใช้ `help(list)` เพื่อขอตัวอย่างการใช้งานได้

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>append()</code>	เพิ่มข้อมูลต่อท้ายลิสต์	<code>lst.append(x)</code> x คือ ข้อมูลที่ต้องการเพิ่ม
<code>clear()</code>	ลบข้อมูลทั้งหมดออกจากลิสต์	<code>lst.clear()</code>
<code>copy()</code>	คัดลอกข้อมูลชนิดลิสต์	<code>lst_new = lst.copy()</code> <code>lst_new</code> คือ ตัวแปรเก็บข้อมูลชนิดลิสต์ <code>lst</code> คือ ตัวแปรข้อมูลชนิดลิสต์ที่ถูกคัดลอก
<code>count()</code>	นับจำนวนข้อมูลที่ซ้ำกันในลิสต์	<code>lst.count(x)</code> x คือ ข้อมูลที่ต้องการนับ
<code>extend()</code>	เพิ่มลิสต์เข้าไปในลิสต์	<code>lst.extend(x)</code> x คือ ลิสต์ที่ต้องการเพิ่ม
<code>index()</code>	ค้นหาตำแหน่งของข้อมูลที่เก็บอยู่ในลิสต์ ถ้าไม่มีตำแหน่งระบุ จะแจ้งเตือนข้อผิดพลาด	<code>lst.index(x, start, stop)</code> x คือ ข้อมูลที่ต้องการค้นหา <code>start</code> คือ จุดเริ่มที่ต้องการค้นหา <code>stop</code> คือ จุดสุดท้ายที่ต้องการให้ค้นหา
<code>insert()</code>	แทรกข้อมูลเข้าไปในลิสต์โดย การระบุตำแหน่ง	<code>lst.insert(index, x)</code> <code>index</code> คือ ตำแหน่งที่ต้องการแทรก <code>x</code> คือ ข้อมูลที่ต้องการแทรก
<code>pop()</code>	ลบข้อมูลโดยการระบุตำแหน่ง	<code>lst.pop(index)</code> <code>index</code> คือ ตำแหน่งที่ต้องการลบ
<code>remove()</code>	ลบข้อมูลด้วยการระบุชื่อ ถ้าไม่มีข้อมูลที่ระบุจะแจ้งเตือนข้อผิดพลาด	<code>lst.remove(x)</code> x คือ ชื่อข้อมูลที่ต้องการลบ
<code>reverse()</code>	สลับตำแหน่งข้อมูลจากด้านหลังมาด้านหน้า	<code>lst.reverse()</code>

ตาราง 5.1: เมธอดและฟังก์ชันที่ใช้กับข้อมูลชนิดลิสต์ (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>sort()</code>	เรียงลำดับข้อมูลที่อยู่ในลิสต์	<code>lst.sort(reverse=False)</code> <code>reverse=False</code> คือ ค่าเริ่มต้นจะเรียงจากมากไปหาน้อย กำหนดหรือไม่กำหนดก็ได้ ถ้ากำหนดให้เป็น <code>True</code> จะเรียงจากมากไปหาน้อย
<code>len()</code>	ฟังก์ชันที่ใช้แสดงจำนวนข้อมูลที่มีอยู่ในลิสต์	<code>len(lst)</code>

ตาราง 5.1: เมธอดและฟังก์ชันที่ใช้กับข้อมูลชนิดลิสต์

### ตัวอย่าง 5.3

การเขียนคำสั่งโปรแกรมการใช้เมธอดและฟังก์ชันกับข้อมูลชนิดลิสต์

```
1 books = ['Python', 'Java', 'C', 'C++', 'C#']
2 numbers = [1, 50, 4, 51, 4, 6, 10, 15]
3
4 books.append('Julia')
5 print('แสดงรายการหนังสือใน books = ', books)
6
7 in_book = books.index('C', 1, 5)
8 print('ตำแหน่งหนังสือภาษา C ใน books = ', in_book)
9
10 books.pop(1)
11 print('รายชื่อที่ถูกนำออก = ', books)
12
13 books.reverse()
14 print('สลับข้อมูลจากด้านหลังมาด้านหน้า = ', books)
15
16 books.sort()
17 print('เรียงรายชื่อหนังสือใน books = ', books)
18
19 books.sort(reverse=True)
20 print('เรียงรายชื่อหนังสือใน books = ', books)
21
22 print('จำนวนข้อมูลใน numbers = ', len(numbers))
```

แสดงรายการหนังสือใน books = ['Python', 'Java', 'C', 'C++', 'C#', 'Julia']  
ตำแหน่งหนังสือภาษา C ใน books = 2  
รายชื่อที่ถูกนำออก = ['Python', 'C', 'C++', 'C#', 'Julia']  
สลับข้อมูลจากด้านหลังมาด้านหน้า = ['Julia', 'C#', 'C++', 'C', 'Python']  
เรียงรายชื่อหนังสือใน books = ['C', 'C#', 'C++', 'Julia', 'Python']  
เรียงรายชื่อหนังสือใน books = ['Python', 'Julia', 'C++', 'C#', 'C']  
จำนวนข้อมูลใน numbers = 8

### 5.1.3 การดำเนินการกับข้อมูลชนิดลิสต์

เราสามารถดำเนินการต่าง ๆ กับข้อมูลชนิดลิสต์ได้ เช่น การทำซ้ำลิสต์ การเข้ามูลลิสต์เข้าด้วยกัน การเปรียบเทียบลิสต์ เป็นต้น แสดงดังตัวอย่างต่อไปนี้

#### ตัวอย่าง 5.4

การเขียนคำสั่งโปรแกรมการดำเนินการต่าง ๆ กับข้อมูลชนิดลิสต์

```
1 # สร้างตัวแปรข้อมูลชนิดลิสต์เป็นข้อมูลชนิดจำนวนเต็ม
2 nums1 = [4, 5, 7, 15]
3 nums2 = [4, 3, 5, 78]
4
5 # สร้างตัวแปรข้อมูลชนิดลิสต์เป็นข้อมูลชนิดสตริง
6 books = ['Python', 'Java', 'C', 'C++', 'C#', 'Scala']
7
8 # แสดงผลลัพธ์การเข้ามูลลิสต์ที่ต้องค่าตัวแปร nums กับ books
9 print(nums1 + books)
10 # แสดงผลลัพธ์การทำซ้ำค่าตัวแปร nums ทั้งหมด 3 ครั้ง
11 print(nums1 * 3)
12 # แสดงผลลัพธ์การเรียงลำดับของตัวแปร nums กับ num_lst1
13 print('num_lst กับ nums1 เท่ากันหรือไม่ = ', nums1 == nums2)
14
15 # นำค่าตัวแปร nums และ book_lst เก็บไว้ในตัวแปร news_lst
16 news_lst = [nums1, books]
17 # แสดงผลลัพธ์ค่าตัวแปร news_lst
18 print(news_lst)
19 # แสดงผลลัพธ์การตรวจสอบ Python เป็นสมาชิกของ books?
20 print('Python เป็นสมาชิกของ books_list = ', 'Python' in books)
```

```
[4, 5, 7, 15, 'Python', 'Java', 'C', 'C++', 'C#', 'Scala']
[4, 5, 7, 15, 4, 5, 7, 15, 4, 5, 7, 15]
num_lst กับ nums1 เท่ากันหรือไม่ =  False
[[4, 5, 7, 15], ['Python', 'Java', 'C', 'C++', 'C#', 'Scala']]
Python เป็นสมาชิกของ books_list = True
```



เราสามารถใช้ฟังก์ชัน `min()`, `max()` และ `sum()` ดำเนินการกับสมาชิกที่เก็บไว้ในตัวแปร ข้อมูลชนิดลิสต์ที่เป็นข้อมูลชนิดจำนวนเต็มหรือจำนวนทศนิยมได้ โดยที่ฟังก์ชัน `min()` ใช้ค้นหาค่าข้อมูลที่น้อยที่สุดที่เก็บอยู่ในลิสต์ ฟังก์ชัน `max()` ใช้หาค่าที่มากที่สุดที่เก็บอยู่ในลิสต์ และฟังก์ชัน `sum()` ใช้สำหรับรวมค่าข้อมูลทั้งหมดที่มีอยู่ในลิสต์ ดังตัวอย่างต่อไปนี้

### ตัวอย่าง 5.5

การเขียนคำสั่งโปรแกรมการดำเนินการต่าง ๆ กับข้อมูลชนิดลิสต์

```
1 # สร้างตัวแปรข้อมูลชนิดลิสต์เป็นข้อมูลชนิดจำนวนเต็ม
2 numbers = [1, 4, 5, 6, 15]
3 # แสดงผลลัพธ์การหาค่าที่น้อยที่สุดด้วยฟังก์ชัน min()
4 print('ค่าที่น้อยที่สุดใน numbers = ', min(numbers))
5 # แสดงผลลัพธ์การหาค่าที่มากที่สุดด้วยฟังก์ชัน max()
6 print('ค่าที่มากที่สุดใน numbers = ', max(numbers))
7 # แสดงผลลัพธ์การหาค่าผลรวมด้วยฟังก์ชัน sum()
8 print('ผลรวมของ numbers = ', sum(numbers))
```

ค่าที่น้อยที่สุดใน numbers = 1  
ค่าที่มากที่สุดใน numbers = 15  
ผลรวมของ numbers = 31



### ตัวอย่าง 5.6

การเขียนคำสั่งโปรแกรมการดำเนินการต่าง ๆ กับข้อมูลชนิดลิสต์

```
1 # สร้างตัวแปรข้อมูลชนิดลิสต์เป็นข้อมูลชนิดจำนวนเชิงซ้อน
2 numbers = [1-4j, 4j, 5-2j, 6, 15]
3 # แสดงผลลัพธ์การหาค่าผลรวมด้วยฟังก์ชัน sum()
4 print('ผลรวมของ numbers = ', sum(numbers))
```

ผลรวมของ numbers = (27-2j)



## 5.2 รู้จักกับข้อมูลชนิดทูเพลิ

ข้อมูลชนิดทูเพลิ (Tuple) มีลักษณะการจัดเก็บข้อมูลเหมือนข้อมูลชนิดลิสต์คือ จัดเก็บข้อมูลแบบลำดับ และสามารถที่เก็บอยู่ภายในข้อมูลชนิดทูเพลิเป็นแบบไดก์ได แต่ข้อมูลชนิดทูเพลิไม่สามารถแก้ไขได การสร้างข้อมูลชนิดทูเพลิขึ้นมาใช้งาน สามารถทั้งหมดจะอยู่ภายใต้เครื่องหมายวงเล็บ ( . . . ) และจะถูกคั่นด้วยเครื่องหมาย Comma ( , )

## ตัวอย่าง 5.7

การเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิดทูเพล็กซ์ขึ้นมาใช้งาน

```
1 # สร้างตัวแปรข้อมูลชนิดทูเพล็กซ์เป็นค่าว่าง
2 tup = ()
3 # สร้างตัวแปรเป็นข้อมูลชนิดทูเพล็กซ์มีสมาชิกเป็นข้อมูลชนิดสตริง ลิสต์ และทูเพล็กซ์
4 var_tup = ('Name', 'Address', [1, 2, 3], ('ID', 3451))
5 # สร้างตัวแปรเป็นข้อมูลชนิดสตริง
6 book = ('Python')
7 # สร้างตัวแปรเป็นข้อมูลชนิดทูเพล็กซ์ มีเครื่องหมาย ( , )
8 book_tup = ('Python', 'C++')
9
10 # แสดงผลลัพธ์จากค่าตัวแปร var_tup
11 print(var_tup)
12 # แสดงผลลัพธ์ข้อมูลชนิดของค่าตัวแปร var_tup
13 print('ข้อมูลชนิดของ var_tup คือ', type(var_tup))
14 # แสดงผลลัพธ์ข้อมูลชนิดของค่าตัวแปร book
15 print('ข้อมูลชนิดของ book คือ', type(book))
16 # แสดงผลลัพธ์ข้อมูลชนิดของค่าตัวแปร book_tup
17 print('ข้อมูลชนิดของ book_tup คือ', type(book_tup))
```

```
('Name', 'Address', [1, 2, 3], ('ID', 3451))
ข้อมูลชนิดของ var_tup คือ <class 'tuple'>
ข้อมูลชนิดของ book คือ <class 'str'>
ข้อมูลชนิดของ book_tup คือ <class 'tuple'>
```



จากที่ได้กล่าวมาแล้วว่าข้อมูลชนิดทูเพล็กซ์ไม่สามารถดำเนินการแก้ไข หากเราพยายามกระทำการดังกล่าว จะมีการแจ้งเตือนข้อผิดพลาดเกิดขึ้น แสดงดังตัวอย่างต่อไปนี้

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>index()</code>	ใช้สำหรับแสดงตำแหน่งของข้อมูล	<code>tup.index(x)</code>
<code>count()</code>	ใช้สำหรับนับจำนวนข้อมูล	<code>tup.count(x)</code>
<code>len()</code>	ฟังก์ชันที่ใช้แสดงจำนวนข้อมูล ที่มีอยู่ในทูเพิล	<code>len(tup)</code>

ตาราง 5.2: เมธอดและฟังก์ชันในข้อมูลชนิดทูเพิล เมื่อ `tup` คือ ตัวแปรข้อมูลชนิดทูเพิล `x` คือ ค่าข้อมูลใด ๆ

### ตัวอย่าง 5.8

การแจ้งเตือนข้อผิดพลาดเมื่อทำการแก้ไขข้อมูลชนิดทูเพิล

```

1 books = ('Python', 'Java', 'C', 'C++', 'C#')
2 print(books)
3 print(books[0])
4 books[0] = 'PYTHON'      # ให้ค่า books[0] ใหม่เป็น 'PYTHON'
5 print(books)

```

```

('Python', 'Java', 'C', 'C++', 'C#')
Python
Traceback (most recent call last):
  File "src/test.py", line 4, in <module>
    books[0] = 'PYTHON'      # ให้ค่า books[0] ใหม่เป็น 'PYTHON'
TypeError: 'tuple' object does not support item assignment

```



จากตัวอย่าง 5.8 ตัวแปร `books` เป็นตัวแปรชนิดทูเพิลที่มีสมาชิกเป็นสตริงจำนวน 4 ตัว และในบรรทัดที่ 4 เป็นคำสั่งโปรแกรมให้ค่าตัวแปร `books` ในตำแหน่งแรก (`books[0]`) แต่ทำให้เกิดการแจ้งเตือนข้อผิดพลาด เนื่องจากข้อมูลชนิดทูเพิลไม่อนุญาตองรับการให้ค่าสมาชิก (Item Assignment)

### 5.2.1 เมธอดและฟังก์ชันที่ใช้กับข้อมูลชนิดทูเพิล

ข้อมูลชนิดทูเพิลเก็บข้อมูลชนิดได้หลากหลายประเภท เช่น จำนวนเต็ม ทศนิยม สตริง ลิสต์ เป็นต้น ภาษาโปรแกรมไพธอนได้จัดเตรียมเมธอดต่าง ๆ และฟังก์ชันไว้ให้เรียกใช้งานดังต่อไปนี้

### ตัวอย่าง 5.9

การเขียนคำสั่งโปรแกรมการใช้เมธอดและฟังก์ชันกับข้อมูลชนิดทุกเพล

```
1 # สร้างตัวแปรข้อมูลชนิดทุกเพล
2 books = ('Python', 'Java', 'C', 'C++', 'C#', 'PHP')
3 # แสดงผลลัพธ์ตำแหน่งด้วยเมธอด index()
4 print('ตำแหน่งของหนังสือภาษา Python คือ ', books.index('Python'))
5 # นับรายการข้อมูลที่ต้องการด้วยเมธอด count()
6 print('จำนวนหนังสือภาษา Python ทั้งหมด คือ', books.count('Python'), 'เล่ม')
7 # แสดงผลลัพธ์รายการข้อมูลทั้งหมดด้วยฟังก์ชัน len()
8 print('จำนวนหนังสือทั้งหมดใน books คือ', len(books), 'เล่ม')
```

ตำแหน่งของหนังสือภาษา Python คือ 0  
จำนวนหนังสือภาษา Python ทั้งหมด คือ 1 เล่ม  
จำนวนหนังสือทั้งหมดใน books คือ 6 เล่ม



### 5.2.2 การดำเนินการกับข้อมูลชนิดทุกเพล

เราสามารถดำเนินการต่าง ๆ กับข้อมูลชนิดทุกเพลเหมือนกับข้อมูลชนิดลิสต์ เช่น การเปรียบเทียบ การทำซ้ำ การตรวจสอบสมาชิก การหาค่าสูงสุด การหาค่าต่ำสุด การหาค่าผลรวม เป็นต้น และดังตัวอย่างต่อไปนี้

### ตัวอย่าง 5.10

การเขียนคำสั่งโปรแกรมตรวจสอบการเป็นสมาชิกในข้อมูลชนิดทุกเพล

```
1 # สร้างตัวแปรข้อมูลชนิดทุกเพลเป็นข้อมูลชนิดสตริง
2 books_tup = ('Python', 'Java', 'C', 'C++', 'C#')
3 # แสดงผลลัพธ์การตรวจสอบ ".Net" ในค่าตัวแปร books_tup
4 print('มีหนังสือ SQL หรือไม่ = ', 'SQL' in books_tup)
5 # แสดงผลลัพธ์การตรวจสอบ "C#" ในค่าตัวแปร books_tup
6 print('มีหนังสือ C++ หรือไม่ = ', 'C++' in books_tup)
```

มีหนังสือ SQL หรือไม่ = False  
มีหนังสือ C++ หรือไม่ = True



### ตัวอย่าง 5.11

การเขียนคำสั่งโปรแกรมหาค่าต่ำสุด ค่าสูงสุด และผลรวม จากข้อมูลชนิดทูเพิล

```
1 # สร้างตัวแปรข้อมูลชนิดทูเพิลเป็นข้อมูลชนิดจำนวนเต็ม
2 numbers = (4, 5, 4.12, 7, 15)
3 # แสดงผลลัพธ์การหาค่าที่มากที่สุดด้วยฟังก์ชัน max()
4 print ('ค่าที่มากที่สุดใน numbers คือ', max(numbers))
5 # แสดงผลลัพธ์การหาค่าที่ต่ำที่สุดด้วยฟังก์ชัน min()
6 print ('ค่าที่ต่ำที่สุดใน numbers คือ', min(numbers))
7 # แสดงผลลัพธ์การหาค่าผลรวมด้วยฟังก์ชัน sum()
8 print ('ผลรวมของ numbers คือ', sum(numbers))
```

ค่าที่มากที่สุดใน numbers คือ 15

ค่าที่ต่ำที่สุดใน numbers คือ 4

ผลรวมของ numbers คือ 35.120000000000005



### ตัวอย่าง 5.12

การเปรียบเทียบ การเชื่อม การทำซ้ำ ข้อมูลชนิดทูเพิล

```
1 n_1 = (-1, -2, -1, 0, 12)
2 n_2 = (1, 2, 3.14, 10)
3 n_3 = (1, 2, 3.14, 10)
4 books_1 = ('Python', 'Java', 'C', 'C++', 'SQL')
5 books_2 = ('C', 'C++', 'C#')
6 print('ผลเปรียบเทียบ n_1 < n_3 คือ', n_1 < n_3)
7 print('ผลเปรียบเทียบ n_2 == n_3 คือ', n_2 == n_3)
8 print('ผลการรวมระหว่าง books_1 กับ booksle_2 คือ', books_1 + books_2)
9 print('books_2 * 2 คือ', books_2 * 2)
```

ผลเปรียบเทียบ n\_1 < n\_3 คือ True

ผลเปรียบเทียบ n\_2 == n\_3 คือ True

ผลการรวมระหว่าง books\_1 กับ booksle\_2 คือ ('Python', 'Java', 'C', 'C++',  
→ 'SQL', 'C', 'C++', 'C#')

books\_2 \* 2 คือ ('C', 'C++', 'C#', 'C', 'C++', 'C#')



## 5.3 รู้จักกับข้อมูลชนิดดิกชันนารี

ข้อมูลดิกชันนารี (Dictionary) สามารถเปลี่ยนแปลงข้อมูลได้เหมือนกับข้อมูลชนิดลิสต์ คือ สามารถเพิ่มลบ แก้ไขข้อมูล แต่มีความแตกต่างระหว่างข้อมูลชนิด 2 ประเภทนี้คือ ข้อมูลชนิดดิกชันนารีจะเก็บข้อมูลแบบคู่ประกอบด้วย คีย์ (Key) และค่าข้อมูล (Value) การจัดเก็บข้อมูลจะอยู่ในครึ่งของหลายวงเล็บปีกกา { ... } แยกส่วนของ Key และ Value ด้วยเครื่องหมาย Colon (:) และใช้เครื่องหมาย Comma (,) เพื่อแยกแต่ละคู่ Key และ Value ตัวอย่างวิธีการเขียนคำสั่งโปรแกรมสร้างข้อมูลดิกชันนารีมีดังนี้

โครงสร้างข้อมูลชนิดดิกชันนารี

```
var = {key_1: val_1, ..., key_n: val_n}
```

key\_1, ..., key\_n

ข้อมูลแบบเปลี่ยนค่าไม่ได้ (Immutable) เช่นข้อมูลชนิดจำนวน สถิติ หรือทุกเพิล โดยที่ค่าทั้งหมดต้องไม่ซ้ำกัน

val\_1, ..., val\_n

ข้อมูลชนิดได้กีด้วย

### ตัวอย่าง 5.13

การเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิดดิกชันนารี

```
1 # Key เป็นข้อมูลชนิดจำนวนเต็ม และ Value เป็นข้อมูลชนิดสถิติ
2 animals = {1: 'Cat', 2: 'Dog', 3: 'Tiger', 4: 'Bird', 5: 'Lion'}
3
4 # Key และ Value เป็นข้อมูลชนิดสถิติ
5 sports = {'1': 'Football', '2': 'Tennis', '3': '', '4': 'Runnig'}
6
7 print(animals)
8 print(sports)
```

```
{1: 'Cat', 2: 'Dog', 3: 'Tiger', 4: 'Bird', 5: 'Lion'}
{'1': 'Football', '2': 'Tennis', '3': '', '4': 'Runnig'}
```



ในส่วนของ Value สามารถเป็นได้ทั้งข้อมูลชนิดสถิติ จำนวนเต็ม ทศนิยม ลิสต์ ทุกเพิล เชต ได้ แต่ มีข้อแม้ว่า Key ต้องไม่ซ้ำกัน นอกจากนี้ Key หรือ Value ยังสามารถสร้างขึ้นมาใหม่เป็นค่าว่างก่อนได้ แสดงดังตัวอย่างต่อไปนี้

### ตัวอย่าง 5.14

การเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิดดิกชันนามีที่มีค่า Key และ Value เป็นข้อมูลชนิดต่างกัน

```
1 # Key เป็นข้อมูลชนิดจำนวนเต็ม และ Value เป็นข้อมูลชนิดสตริง
2 animals = {1: 'Cat', 2: 'Dog', 3: 'Tiger', 4: 'Bird', 5: 'Lion'}
3
4 # Key และ Value เป็นข้อมูลชนิดสตริง
5 sports = {'1': 'Football', '2': 'Tennis', '3': '', '4': 'Basketball'}
6
7 # Key ข้อมูลชนิดสตริงและ Value เป็นข้อมูลชนิดทูเพิล
8 laptops = {'CPU': ('Core i5', 'Core i7', 'M1'), 'Ram': ('8GB', '16GB',
    ↴ '32GB'), 'SSD': ('512GB', '1TB', '2TB')}
9
10 # Key เป็นข้อมูลชนิดทศนิยม และ Value เป็นข้อมูลชนิดสตริง
11 offices = {1.1: ['เก้าอี้', 'โต๊ะ,'], 1.2: ['ปากกา', 'ดินสอ', 'ยางลบ'], 1.3:
    ↴ ['กรรไกร', 'คัตเตอร์']}
12 print(animals) # แสดงผลลัพธ์จากค่าตัวแปร animals
13 print(sports) # แสดงผลลัพธ์จากค่าตัวแปร sports
14 print(laptops) # แสดงผลลัพธ์จากค่าตัวแปร devices
15 print(offices) # แสดงผลลัพธ์จากค่าตัวแปร offices
```

```
{1: 'Cat', 2: 'Dog', 3: 'Tiger', 4: 'Bird', 5: 'Lion'}
{'1': 'Football', '2': 'Tennis', '3': '', '4': 'Basketball'}
{'CPU': ('Core i5', 'Core i7', 'M1'), 'Ram': ('8GB', '16GB', '32GB'),
    ↴ 'SSD': ('512GB', '1TB', '2TB')}
{1.1: ['เก้าอี้', 'โต๊ะ,'], 1.2: ['ปากกา', 'ดินสอ', 'ยางลบ'], 1.3: ['กรรไกร',
    ↴ 'คัตเตอร์']}
```

#### 5.3.1 เมรอดและฟังก์ชันที่ใช้กับข้อมูลชนิดดิกชันนารี

ข้อมูลชนิดดิกชันนารีมีเมรอดและฟังก์ชันต่าง ๆ ให้เรียกใช้งานจำนวนมาก เพื่อนำมาใช้ดำเนินการกับข้อมูลที่ถูกจัดเก็บไว้ในตัวแปร หากเราต้องการดูรายละเอียดการใช้งานเพิ่มเติม สามารถเรียกดูผ่านคำสั่ง `help(dict)` ได้

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>clear()</code>	ใช้สำหรับลบข้อมูลทั้งหมด	<code>d.clear()</code>
<code>copy()</code>	ใช้สำหรับคัดลอกข้อมูลนิดเด็กชั้น นารี	<code>d.copy()</code>
<code>del()</code>	ใช้สำหรับลบ key หรือตัวแปรติกชั้น นารี นับจำนวนข้อมูลที่ซ้ำกันในลิสต์	<code>del d(key)</code> หรือ <code>del d</code>
<code>fromkeys()</code>	ใช้สำหรับสร้าง Value ให้เหมือนกัน ทุกค่า แต่ key จะไม่ซ้ำกัน	<code>d.fromkeys(key, value=None)</code> key คือ Key ที่ต้องการระบุที่ไม่ซ้ำกัน <code>value=None</code> คือ ข้อมูลของแต่ละ Key จะกำหนดหรือไม่มีก็ได้
<code>get()</code>	ใช้สำหรับแสดงค่า Value ของ Key ถ้าไม่พบค่า Key ที่ได้ระบุ จะคืนค่า เป็น None	<code>d.get(key, value=None)</code> key คือ Key ที่ต้องการแสดง Value <code>value=None</code> คือ Value ของแต่ละ Key จะกำหนดหรือไม่มีก็ได้
<code>items()</code>	ใช้สำหรับแสดงข้อมูลทั้งหมด	<code>d.items()</code>
<code>keys()</code>	ใช้สำหรับแสดงชื่อ Key ทั้งหมด	<code>d.keys()</code>
<code>pop()</code>	ใช้สำหรับลบ Value โดยการระบุ ตำแหน่ง Key โดยจะคืนค่า Value ที่ถูกลบ ถ้าระบุ Key ที่ไม่มีอยู่ในติก ชั้นนารี จะเกิดการแจ้งเตือนข้อผิด พลาดขึ้น	<code>d.pop(key [, default])</code> key คือ ค่า Key ที่ต้องการลบค่า Value default คือ ค่าที่ส่งกลับมา ถ้าพบ Key ที่กำหนด จะคืนค่ากลับมาเป็น Value ถ้า ไม่พบ Key ที่กำหนด จะคืนค่ากลับมา เป็นค่าที่กำหนดในส่วน default ถ้า ไม่ได้กำหนดจะคืนค่ากลับมาเป็นการแจ้ง เตือนข้อผิดพลาด

ตาราง 5.3: เมธอดและฟังก์ชันสำหรับตัวแปรข้อมูลติกชั้นนารี (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>popitem()</code>	ใช้สำหรับลบข้อมูลตัวสุดท้ายออก จากดิกชันนารี จะคืนค่าคู่ Key และ Value กลับมา ถ้าลบค่าข้อมูลในดิกชันนารีที่เก็บค่าว่างจะเกิดการแจ้งเตือนข้อผิดพลาด	<code>d.popitem()</code>
<code>setdefault()</code>	ใช้สำหรับค้นหาและเพิ่มข้อมูลให้กับดิกชันนารี แต่มีข้อแม้ว่า Key ต้องไม่ซ้ำกับ Key ที่มีอยู่ ถ้าพบว่ามี Key อยู่แล้วจะไม่ทำการเพิ่มข้อมูล หรือเปลี่ยนแปลง Value แต่ถ้าไม่พบ Key ถึงจะเพิ่มค่าข้อมูลใหม่ และหากไม่กำหนด Value จะกำหนดเป็นค่า None	<code>d.setdefault(key, [default=None])</code> key คือ Key ที่ต้องการสร้างและเพิ่ม Value <code>default=None</code> คือ Value ที่ต้องการเพิ่ม หากไม่กำหนดจะเป็นค่า None
<code>update()</code>	ใช้สำหรับแก้ไขข้อมูลในดิกชันนารี และจะทำการเรียงทับหากมี Key นั้นอยู่แล้ว	<code>d.update([other])</code> other คือ ตัวแปรชนิดดิกชันนารี เช่น <code>d.update({'cat': 2})</code> หรือทุกเพลที่มีความยาวเป็น 2 เช่น <code>d.update('cat', 2)</code> หรือการให้ค่าในรูปแบบ key/value เช่น <code>d.update(cat=2)</code>
<code>values()</code>	ใช้สำหรับแสดงค่า Value ทั้งหมดในดิกชันนารี	<code>d.values()</code>
<code>len()</code>	เป็นฟังก์ชันที่ใช้แสดงจำนวนสมาชิก ในดิกชันนารี	<code>len(d)</code>

ตาราง 5.3: เมธอดและฟังก์ชันสำหรับตัวแปรข้อมูลดิกชันนารี

### ตัวอย่าง 5.15

การเขียนคำสั่งโปรแกรมการใช้เมธอดและฟังก์ชันกับข้อมูลชนิดดิกชันนารี

```
1 # Key เป็นข้อมูลชนิดจำนวนเต็ม และ Value เป็นข้อมูลชนิดสตริง
2 animals = {1: 'Cat', 2: 'Dog', 3: 'Rat', 4: 'Bird', 5: 'Lion'}
3 # Key และ Value เป็นข้อมูลชนิดสตริง
4 sports = {'1': 'Football', '2': '', '3': 'Golf', '4': 'Basketball'}
5 # ใช้เมธอด get() แสดงค่าข้อมูล Value ของ animals
6 print ('ตำแหน่งที่ 2 ของ animals คือ', animals.get(2))
7 # ใช้เมธอด key() แสดงค่า key ทั้งหมดในค่าตัวแปร sports
8 print ('key ที่มีอยู่ใน sports คือ', sports.keys())
9 # แสดงจำนวนข้อมูลทั้งหมดด้วยฟังก์ชัน len() ของ animals
10 print ('ข้อมูลทั้งหมดใน animals คือ', len(animals))
11 # แสดงค่า Value ทั้งหมดด้วยเมธอด values() ใน sports
12 print ('ค่า value ที่อยู่ใน sports = ', sports.values())
```

ตำแหน่งที่ 2 ของ animals คือ Dog  
key ที่มีอยู่ใน sports คือ dict\_keys(['1', '2', '3', '4'])  
ข้อมูลทั้งหมดใน animals คือ 5  
ค่า value ที่อยู่ใน sports = dict\_values(['Football', '', 'Golf',  
→ 'Basketball'])



### 5.3.2 การดำเนินการกับข้อมูลชนิดดิกชันนารี

นอกจากเมธอดต่าง ๆ ที่ได้กล่าวมาแล้วนั้น เรายังดำเนินการกับข้อมูลชนิดดิกชันนารีได้อีก หากเราต้องการลบข้อมูลออกจากตัวแปรทั้งค่า Key และค่า Value ให้ใช้คำสั่ง del แต่จะมีการแจ้งเตือนเกิดขึ้น เมื่อระบุตำแหน่งหรือระบุตัวแปรที่ต้องการลบไม่ถูกต้อง มีรูปแบบการใช้งานดังนี้

โครงสร้างคำสั่งการลบข้อมูลออกจากตัวแปรชนิดดิกชันนารี

**del dict.[key]**

**dict** ตัวแปรข้อมูลชนิดดิกชันนารีที่ต้องลบ ออบเจ็คที่ต้องการแสดงผล มีได้มากกว่า 1 ออบเจ็ค

**key** Key ที่ต้องการลบ หากไม่ระบุ จะเป็นการลบค่าตัวแปรนั้นทั้งหมด

### ตัวอย่าง 5.16

การเขียนคำสั่งโปรแกรมลบดิกชันนารี โดยการคำสั่ง del

```
1 sports = {'1': 'Football', '2': 'Tennis', '3': 'Basketball',
2   '4': 'Golf'}
3 del sports['4'] # ใช้คำสั่ง del ลบข้อมูล Key '4' ของ sports
4 print('ข้อมูลที่อยู่ใน sports = ', sports)
5 del sports # ใช้คำสั่ง del ตัวแปร sports
6 print(sports)
```

```
ข้อมูลที่อยู่ใน sports = {'1': 'Football', '2': 'Tennis', '3': 'Basketball'}  
Traceback (most recent call last):  
  File "src/test.py", line 6, in <module>  
    print(sports)  
NameError: name 'sports' is not defined
```

จากตัวอย่าง 5.16 จะเห็นว่าหลังจากเราทำการลบข้อมูล sports ในตำแหน่งที่ key '**4**' ในบรรทัดที่ 3 ทำให้ sports มีสมาชิกเหลือเพียง 3 ชุด และในบรรทัดที่ 5 เราทำการลบข้อมูลตัวแปรดิกชันนารี sports ทั้งหมด ทำให้โปรแกรมแสดงข้อผิดพลาดว่าไม่มีตัวแปรชื่อ sports เมื่อเราต้องการแสดงค่า sports ในบรรทัดที่ 6

เมื่อเราต้องการแก้ไขข้อมูลในส่วนของ Value หรือเพิ่มข้อมูลเข้าไปเก็บไว้ในตัวแปรดิกชันนารี เราสามารถใช้คำสั่ง **dict[key]=value** โดยที่ key คือ Key ที่ต้องการแก้ไข หรือเพิ่มข้อมูลใหม่ ถ้า Key มีอยู่แล้วจะเป็นการเพิ่มข้อมูลใหม่ ส่วน value คือ ค่าข้อมูลที่ต้องการแก้ไข

### ตัวอย่าง 5.17

การเขียนคำสั่งโปรแกรมเปลี่ยนข้อมูลของตัวแปรดิกชันนารี

```
1 # Key และ Value เป็นข้อมูลชนิดสตริง
2 sports = {'1': 'Football', '2': 'Tennis', '3': 'Basketball'}
3 # แก้ไข Value ที่ Key 1 จาก Football เป็น Racing
4 sports['1'] = 'Golf'
5 # เพิ่มสมาชิกเข้าไปในค่าตัวแปร sports
6 sports['4'] = 'Rugby'
7 # แสดงผลลัพธ์จากการแก้ไขและเพิ่มข้อมูล
8 print('รายชื่อชนิดกีฬาที่มีอยู่ใน sports = ', sports)
```

```
รายชื่อชนิดกีฬาที่มีอยู่ใน sports = {'1': 'Golf', '2': 'Tennis', '3':  
  'Basketball', '4': 'Rugby'}
```

## 5.4 รู้จักกับข้อมูลชนิดเซต

การสร้างข้อมูลชนิดเซต (Set) ขึ้นมาใช้งาน จะเป็นการจัดเก็บกลุ่มข้อมูลประเภทเดียวกันหรือคล้ายกันให้อยู่ด้วยกัน เช่น

กลุ่มข้อมูลสัตว์ปีก = {'ไก่', 'เป็ด', 'นก', 'ห่าน'}

กลุ่มข้อมูลยี่ห้อรถยนต์ = {'Toyota', 'Honda', 'BMW', 'Porsche'}

กลุ่มข้อมูลกีฬา = {'Football', 'Basketball', 'Golf', 'Tennis'}

เป็นต้น ข้อมูลที่ถูกเก็บในข้อมูลชนิดเซตจะอยู่ในเครื่องหมายเปิดกาง { ... } และถูกคั่นด้วยเครื่องหมาย Comma (,) ข้อมูลชนิดเซตจัดเก็บข้อมูลแบบไม่มีลำดับ (Unordered Collection) แบ่งออกเป็น 2 ประเภทได้แก่ Set และ Frozenset แต่มีความแตกต่างกันคือ Set สามารถเปลี่ยนแปลงแก้ไขสมาชิกได้ แต่สำหรับ Frozenset นั้น ไม่สามารถแก้ไขสมาชิกในเซตได้ กรณีที่สร้างเซตขึ้นมาใช้งานและมีสมาชิกซ้ำกัน และเมื่อสั่งประมวลผลสมาชิกที่ซ้ำกันจะเหลือเพียงแค่ตัวเดียวเท่านั้น

### ตัวอย่าง 5.18

การเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิดเซตและการตรวจสอบข้อมูลชนิดเซต

```
1 # สร้างตัวแปร sport เป็นข้อมูลชนิดเซตที่เป็นเซตว่าง
2 sport = set()
3 # สร้างตัวแปรเป็นข้อมูลชนิดเซตของข้อมูลชนิดสตริง
4 books = {'Python', 'C', 'C++', 'Java', 'C++'}
5 # สร้างตัวแปรเป็นข้อมูลชนิดเซตเป็นข้อมูลชนิดจำนวนเต็ม
6 numbers = {2, 5, 11, 7, 3, 13, 17, 19}
7 # แสดงผลลัพธ์ค่าตัวแปร books
8 print('สมาชิกของ books ประกอบด้วย', books)
9 # แสดงผลลัพธ์ค่าตัวแปร numbers
10 print('สมาชิกของ numbers ประกอบด้วย', numbers)
11 # ตรวจสอบข้อมูลชนิดด้วยฟังก์ชัน type()
12 print('ข้อมูลชนิดของ sport คือ', type(sport))
13 # ตรวจสอบข้อมูลชนิดด้วยฟังก์ชัน type()
14 print('ข้อมูลชนิดของ books คือ', type(books))
```

สมาชิกของ books ประกอบด้วย {'C', 'C++', 'Java', 'Python'}  
สมาชิกของ numbers ประกอบด้วย {2, 3, 5, 7, 11, 13, 17, 19}  
ข้อมูลชนิดของ sport คือ <class 'set'>  
ข้อมูลชนิดของ books คือ <class 'set'>



จากตัวอย่าง 5.18 ในบรรทัดที่ 4 ตัวแปร books มี 'C++' ปรากฏอยู่ 2 ครั้ง เมื่อแสดงผลออกมาจะทำให้ข้อมูลที่ซ้ำกันเหลือเพียงค่าเดียว ในบรรทัดที่ 6 เราสร้างตัวแปร numbers เมื่อทำการแสดงผล

ผ่านคำสั่ง `print()` ผลลัพธ์ที่ได้ในแต่ละครั้งอาจจะมีการเรียงลำดับต่างกันไป ซึ่งเป็นการสุ่มสมาชิกของเซตมาแสดงผลแบบไม่มีลำดับ

### ตัวอย่าง 5.19

ข้อมูลชนิดเซตเป็นข้อมูลแบบไม่มีลำดับ

```
1 # สร้างตัวแปรเป็นข้อมูลชนิดเซตของข้อมูลชนิดสตริง
2 books = {'Python', 'C', 'C++', 'Java', 'C++'}
3 # แสดงข้อมูลตัวแปร books
4 print('ตัวแปร books คือ', books)
5 # การเข้าถึงข้อมูลตำแหน่งแรกของ books
6 print('ข้อมูลตำแหน่งแรกของ books คือ', books[0])
```

```
ตัวแปร books คือ {'Java', 'C', 'C++', 'Python'}
Traceback (most recent call last):
  File "src/test.py", line 6, in <module>
    print('ข้อมูลตำแหน่งแรกของ books คือ', books[0])
TypeError: 'set' object is not subscriptable
```



จากตัวอย่าง 5.19 ในบรรทัดที่ 4 เมื่อพยายามเข้าถึงข้อมูลตำแหน่งแรกของตัวแปร `books` โปรแกรมจะแสดงข้อผิดพลาดดังนี้ เนื่องจากข้อมูลชนิดเซตจัดเก็บข้อมูลแบบไม่มีลำดับ ภาษาโปรแกรม Python ไม่อนุญาตให้ใช้เครื่องหมาย [ . . . ] ในการเข้าถึงได้

จากที่ได้นำเสนอตัวอย่างข้างต้นเป็นการสร้างข้อมูลชนิดเซต ซึ่งเป็นข้อมูลชนิดที่สามารถแก้ไขหรือลบสมาชิกออกจากเซตได้ และมีอีกหนึ่งข้อมูลชนิดคือ `Frozenset` ซึ่งเป็นข้อมูลชนิดที่มีการเก็บสมาชิกเหมือนกับข้อมูลชนิดเซต แต่สมาชิกภายในเซตจะไม่สามารถแก้ไขเปลี่ยนแปลงได้ หากพยายามแก้ไขสมาชิกจะเกิดการแจ้งเตือนข้อผิดพลาดดังนี้ การสร้างข้อมูลชนิด `Frozenset` ขึ้นมาใช้งานจะอยู่ในรูปแบบของ `frozensent()` แสดงดังตัวอย่างต่อไปนี้

### ตัวอย่าง 5.20

การเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิด Frozenset

```
1 # สร้างตัวแปร cities เป็นข้อมูลชนิด frozenset
2 cities = frozenset(['Bangkok', 'Berlin', 'Tokyo', 'New York'])
3 # สร้างตัวแปร colors เป็นข้อมูลชนิด frozenset
4 colors = frozenset(['Red', 'Green', 'Blue'])
5 # แสดง types และ สมาชิกที่เก็บอยู่ในค่าตัวแปร cities
6 print('cities = ', cities, 'which type =', type(cities))
7 # แสดง types และ สมาชิกที่เก็บอยู่ในค่าตัวแปร colors
8 print('colors = ', colors, 'which type =', type(colors))
```

```
cities = frozenset({'New York', 'Bangkok', 'Tokyo', 'Berlin'}) whi▶
↪ type = <class 'frozenset'>
colors = frozenset({'Green', 'Blue', 'Red'}) which type = <class
↪ 'frozenset'>
```

#### 5.4.1 เมธอดและฟังก์ชันที่ใช้กับข้อมูลชนิดเซต

ข้อมูลชนิดเซตมีเมธอดต่าง ๆ ที่นำมาใช้จัดการกับสมาชิกที่อยู่ภายในเซตจำนวนมาก หากเราสามารถใช้คำสั่ง `help(set)` ขอดุข้อมูลการใช้งานเมธอดกับข้อมูลชนิดนี้เพิ่มเติมได้

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>add()</code>	เพิ่มสมาชิกเข้าไปในเซต	<code>s.add()</code>
<code>clear()</code>	ลบสมาชิกทั้งหมดในเซต	<code>s.clear()</code>
<code>copy()</code>	คัดลอกข้อมูลชนิดเซต	<code>s.copy()</code>
<code>difference()</code>	หาสมาชิกที่ต่างกันระหว่างสองเซต หรือใช้ตัวดำเนินการ <code>-</code> แทนได้	<code>s1.difference(s2)</code>
<code>difference_update()</code>	หาสมาชิกที่ต่างกันระหว่างสองเซต แต่เซตที่นำมาเปรียบเทียบ จะเก็บค่าสมาชิกใหม่	<code>s1.difference_update(s2)</code>

ตาราง 5.4: เมธอดและฟังก์ชันของข้อมูลชนิดเซต (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>discard()</code>	ลบสมาชิกในเซต	<code>s.discard(x)</code> x คือ ชื่อสมาชิกที่ต้องการลบ
<code>intersection()</code>	หาสมาชิกที่เหมือนกันระหว่างเซต สามารถใช้เครื่องหมาย <code>&amp;</code> แทนได้	<code>s1.intersection(s2, ..., sn)</code>
<code>intersection_update()</code>	หาสมาชิกที่เหมือนกันระหว่างสองเซต ผลลัพธ์ที่ได้จะถูกนำไปแทนที่สมาชิกของเซตที่นำมา หากสมาชิกที่เหมือนกัน	<code>s1.intersection_update(s2)</code>
<code>isdisjoint()</code>	ตรวจสอบสมาชิกภายในเซตว่า เป็นสมาชิกภายในเซตว่าเป็นสมาชิกของเซตอื่นด้วยหรือไม่ ถ้าเป็นสมาชิกของเซตอื่น ผลลัพธ์เป็น <code>False</code> ถ้าไม่เป็น สมาชิกของเซตอื่นผลลัพธ์เป็น <code>True</code>	<code>s1.isdisjoint(s2)</code>
<code>issubset()</code>	ตรวจสอบการเป็นสับเซต (subset) หรือเซตย่อย ถ้า สมาชิกทุกตัวของ <code>s1</code> เป็น สมาชิกของ <code>s2</code> จะได้ผลลัพธ์ เป็น <code>True</code> ถ้ามีบางตัวไม่เป็น สมาชิกของตัวแปร <code>s2</code> จะได้ ผลลัพธ์เป็น <code>False</code>	<code>s1.issubset(s2)</code>
<code>issuperset()</code>	ตรวจสอบการเป็นซุปเปอร์เซต ของเซต ถ้าเป็นซุปเปอร์เซต ให้ผลลัพธ์เป็น <code>True</code> ถ้าไม่ เป็นซุปเปอร์เซตให้ผลลัพธ์เป็น <code>False</code>	<code>s1.issuperset(s2)</code>

ตาราง 5.4: เมธอดและฟังก์ชันของข้อมูลชนิดเซต (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>pop()</code>	ลบสมาชิกออกจากเซตโดย การสุ่ม จะคืนค่ากลับมาเป็น สมาชิกที่ถูกลบ ถ้าลบสมาชิก ในเซตว่างจะคืนค่ากลับมา เป็นการแจ้งเตือนข้อผิดพลาด	<code>s.pop()</code>
<code>remove()</code>	ลบสมาชิกออกจากเซตเหมือน กับเมธอด <code>discard()</code> แต่ เมื่อนำเมธอดนี้มาใช้งาน ถ้าลบ สมาชิกที่ไม่มีอยู่ในเซตจะเกิด การแจ้งเตือนข้อผิดพลาด	<code>s.remove(x)</code> x คือ ชื่อสมาชิกที่ต้องการลบ
<code>symmetric_difference()</code>	หาสมาชิกที่ไม่มีอยู่ในอีกเซต สามารถใช้เครื่องหมาย (^) แทนได้	<code>s1.symmetric_difference(s2)</code>
<code>symmetric_difference_update()</code>	รวมสมาชิกของเซตเข้าด้วยกัน สามารถใช้เครื่องหมาย   แทน ได้ ถ้ามีสมาชิกที่เหมือนกันจะ เหลือเพียงตัวเดียว	<code>s1.symmetric_difference_update(s2)</code>
<code>union()</code>	update สมาชิกของเซต หรือ เป็นการรวมสมาชิกสองเซต ถ้า สมาชิกมีซ้ำกันจะถูกตัดออกให้ เหลือเพียงตัวเดียว	<code>s1.union(s2)</code>
<code>update()</code>	update สมาชิกของเซต หรือ เป็นการรวมสมาชิกสองเซต ถ้า สมาชิกมีซ้ำกันจะถูกตัดออกให้ เหลือเพียงตัวเดียว	<code>s1.update(s2)</code>
<code>len()</code>	ฟังก์ชันแสดงจำนวนสมาชิก ทั้งหมดในเซต	<code>len(s)</code>

ตาราง 5.4: เมธอดและฟังก์ชันของข้อมูลชนิดเซต

## 5.4.2 การดำเนินการกับข้อมูลนิดเซต

นอกจากเราสามารถนำเมธอดข้างต้นมาใช้กับข้อมูลนิดเซตแล้ว

เรายังสามารถนำฟังก์ชัน

`min()`, `max()` และ `sum()` ตัวดำเนินการเปรียบเทียบ ตัวดำเนินการเป็นสมาชิก หรือตัวดำเนินการเอกลักษณ์ เข้ามาดำเนินการกับข้อมูลนิดเซตได้อีกด้วย แสดงต่อไปนี้

### ตัวอย่าง 5.21

การใช้ฟังก์ชัน `min()`, `max()` และ `sum()` กับข้อมูลนิดเซต

```
1 # สร้างตัวแปรข้อมูลนิดเซตเป็นจำนวนเต็ม
2 nums = {2, 3, 5, 7, 11}
3 # แสดงผลลัพธ์ค่าที่น้อยที่สุดด้วยฟังก์ชัน min()
4 print ('จำนวนที่น้อยที่สุดใน nums คือ', min(nums))
5 # แสดงผลลัพธ์ค่าที่มากที่สุดด้วยฟังก์ชัน max()
6 print ('จำนวนที่มากที่สุดใน nums คือ', max(nums))
7 # แสดงผลลัพธ์ค่าผลรวมด้วยฟังก์ชัน sum()
8 print ('ผลรวมสมาชิกใน nums คือ', sum(nums))
```

จำนวนที่น้อยที่สุดใน `nums` คือ 2

จำนวนที่มากที่สุดใน `nums` คือ 11

ผลรวมสมาชิกใน `nums` คือ 28



สำหรับตัวดำเนินการเปรียบเทียบ `<=` และ `<` สำหรับข้อมูลนิดเซตจะหมายถึงการตรวจสอบว่าเป็นซับเซต (Subset) และซับเซตแท้ (Proper Subset) หรือไม่ตามลำดับ ซึ่งความหมายของซับเซตและซับเซตแท้เนี้ยจะนิยามตามความหมายทางคณิตศาสตร์ ในทำนองเดียวกันกับตัวดำเนินการเปรียบเทียบ `>=` และ `>` สำหรับข้อมูลนิดเซตจะหมายถึงการตรวจสอบว่าเป็นซูปเปอร์เซต (Superset) และซูปเปอร์เซตแท้ (Proper Superset) หรือไม่ตามลำดับ และสำหรับตัวดำเนินการเปรียบเทียบ `==` สำหรับข้อมูลนิดเซตจะหมายถึงการตรวจสอบว่าตัวแปรเซตทั้งสองตัวนั้นมีข้อมูลเหมือนกันใช่หรือไม่

### ตัวอย่าง 5.22

การใช้ตัวดำเนินการเปรียบเทียบกับข้อมูลชนิดเซต

```
1 A = {2, 3, 5, 7, 11}
2 B = {3, 2, 7, 5}
3 C = {7, 5, 3, 2}
4
5 print('A เป็นซับเซตของ B หรือไม่', A < B)
6 print('A เป็นซับเซตของ B หรือไม่', A <= B)
7 print('A เป็นซูปเปอร์เซตของ B หรือไม่', A > B)
8 print('A เป็นซูปเปอร์เซตของ B หรือไม่', A >= B)
9 print('B เป็นซับเซตของ C หรือไม่', B < C)
10 print('B เป็นซับเซตของ C หรือไม่', B <= C)
11 print('4 เป็นสมาชิกของ A หรือไม่', 4 in A)
12 print('0 ไม่เป็นสมาชิกของ A หรือไม่', 0 not in A)
13 print('A เท่ากับ B หรือไม่', A == B)
14 print('B เท่ากับ C หรือไม่', B == C)
15 print('B คือออบเจ็คเดียวกับ C หรือไม่', B is C)
```

A เป็นซับเซตแห่ง B หรือไม่ False  
A เป็นซับเซตของ B หรือไม่ False  
A เป็นซูปเปอร์เซตแห่ง B หรือไม่ True  
A เป็นซูปเปอร์เซตของ B หรือไม่ True  
B เป็นซับเซตแห่ง C หรือไม่ False  
B เป็นซับเซตของ C หรือไม่ True  
4 เป็นสมาชิกของ A หรือไม่ False  
0 ไม่เป็นสมาชิกของ A หรือไม่ True  
A เท่ากับ B หรือไม่ False  
B เท่ากับ C หรือไม่ True  
B คือออบเจ็คเดียวกับ C หรือไม่ False



## 5.5 การแปลงข้อมูลชนิดลิสต์ ทูเพิล ดิกชันนารี และเซต

ในบางครั้งเราอาจจำเป็นต้องทำการแปลง (Casting) ข้อมูลชนิดหนึ่งไปเป็นอีกชนิดหนึ่ง เช่น เมื่อเราต้องการนำจัดข้อมูลที่ซ้ำกันในข้อมูลชนิดลิสต์ เราอาจจะแปลงข้อมูลนั้นไปเป็นข้อมูลชนิดเซตก่อน และค่อยแปลงกลับไปเป็นข้อมูลชนิดลิสต์ดังเดิม ซึ่งเราสามารถใช้ฟังก์ชันต่อไปนี้ในการแปลงข้อมูลชิดต่าง ๆ ได้

- ฟังก์ชัน `list()` ใช้ในการแปลงข้อมูลเป็นข้อมูลชนิดลิสต์

- พังก์ชัน `tuple()` ใช้ในการแปลงข้อมูลเป็นข้อมูลชนิดทูเพิล
- พังก์ชัน `dict()` ใช้ในการแปลงข้อมูลเป็นข้อมูลชนิดดิกชันนารี
- พังก์ชัน `set()` และ `frozenset()` ใช้ในการแปลงข้อมูลเป็นข้อมูลชนิดเซต

### ตัวอย่าง 5.23

การเขียนคำสั่งโปรแกรมแปลงข้อมูลเป็นข้อมูลชนิดลิสต์

```

1  fnums = [3.50, 17.475, 15.899]
2  text = 'Python'
3  snums = {2, 3, 5, 7}
4  sports = {'1': 'Football', '2': 'Tennis', '3': 'Basketball'}
5  print('แปลงลิสต์เป็นลิสต์ ->', list(fnums))
6  print('แปลงสตริงเป็นลิสต์ ->', list(text))
7  print('แปลงเซตเป็นลิสต์ ->', list(snums))
8  print('แปลง Key ในดิกชันนารีเป็นลิสต์ ->', list(sports.keys()))
9  print('แปลง Value ในดิกชันนารีเป็นลิสต์ ->', list(sports.values()))

```

แปลงลิสต์เป็นลิสต์ -> [3.5, 17.475, 15.899]  
 แปลงสตริงเป็นลิสต์ -> ['P', 'y', 't', 'h', 'o', 'n']  
 แปลงเซตเป็นลิสต์ -> [2, 3, 5, 7]  
 แปลง Key ในดิกชันนารีเป็นลิสต์ -> ['1', '2', '3']  
 แปลง Value ในดิกชันนารีเป็นลิสต์ -> ['Football', 'Tennis', 'Basketball']



### ตัวอย่าง 5.24

การเขียนคำสั่งโปรแกรมแปลงข้อมูลเป็นข้อมูลชนิดทูเพิล

```
1 fnums = [3.50, 17.475, 15.899]
2 text = 'Python'
3 snums = {2, 3, 5, 7}
4 sports = {'1': 'Football', '2': 'Tennis', '3': 'Basketball'}
5 print('แปลงลิสต์เป็นทูเพิล ->', tuple(fnums))
6 print('แปลงสตริงเป็นทูเพิล ->', tuple(text))
7 print('แปลงเซตเป็นทูเพิล ->', tuple(snums))
8 print('แปลง Key ในดิกชันนารีเป็นทูเพิล ->', tuple(sports.keys()))
9 print('แปลง Value ในดิกชันนารีเป็นทูเพิล ->', tuple(sports.values()))
```

```
แปลงลิสต์เป็นทูเพิล -> (3.5, 17.475, 15.899)
แปลงสตริงเป็นทูเพิล -> ('P', 'y', 't', 'h', 'o', 'n')
แปลงเซตเป็นทูเพิล -> (2, 3, 5, 7)
แปลง Key ในดิกชันนารีเป็นทูเพิล -> ('1', '2', '3')
แปลง Value ในดิกชันนารีเป็นทูเพิล -> ('Football', 'Tennis', 'Basketball')
```



### ตัวอย่าง 5.25

การเขียนคำสั่งโปรแกรมแปลงข้อมูลเป็นข้อมูลชนิดดิกชันนารี

```
1 animals = [[1, 'Cat'], [2, 'Dog'], [3, 'Bee'], [4, 'Ant']]
2 sports = ((1, 'Football'), (2, 'Basketball'))
3 numbers = {2, 3, 5, 7}
4 print('แปลงลิสต์เป็นดิกชันนารี ->', dict(animals))
5 print('แปลงทูเพิลเป็นดิกชันนารี ->', dict(sports))
6 print('แปลงเซตเป็นดิกชันนารี ->', dict.fromkeys(numbers))
```

```
แปลงลิสต์เป็นดิกชันนารี -> {1: 'Cat', 2: 'Dog', 3: 'Bee', 4: 'Ant'}
แปลงทูเพิลเป็นดิกชันนารี -> {1: 'Football', 2: 'Basketball'}
แปลงเซตเป็นดิกชันนารี -> {2: None, 3: None, 5: None, 7: None}
```



### ตัวอย่าง 5.26

การเปลี่ยนค่าสั่งโปรแกรมแปลงข้อมูลเป็นข้อมูลชนิดเซต

```
1  fnums = (3.14, 16.0, 3.14)
2  books = ['Python', 'Java', 'C', 'C++']
3  text = 'Python'
4  numbers = '112'
5  print('แปลงทูเพิลเป็นเซต ->', set(fnums))
6  print('แปลงลิสต์เป็นเซต ->', set(books))
7  print('แปลงสตริงเป็นเซต ->', set(text))
8  print('แปลงสตริงเป็นเซต ->', set(numbers))
9  print('แปลงทูเพิลเป็นเซต ->', frozenset(fnums))
10 print('แปลงลิสต์เป็นเซต ->', frozenset(books))
11 print('แปลงสตริงเป็นเซต ->', frozenset(text))
12 print('แปลงสตริงเป็นเซต ->', frozenset(numbers))
```

```
แปลงทูเพิลเป็นเซต -> {16.0, 3.14}
แปลงลิสต์เป็นเซต -> {'C', 'C++', 'Python', 'Java'}
แปลงสตริงเป็นเซต -> {'n', 't', 'o', 'P', 'y', 'h'}
แปลงสตริงเป็นเซต -> {'1', '2'}
แปลงทูเพิลเป็นเซต -> frozenset({16.0, 3.14})
แปลงลิสต์เป็นเซต -> frozenset({'C', 'C++', 'Python', 'Java'})
แปลงสตริงเป็นเซต -> frozenset({'n', 't', 'o', 'P', 'y', 'h'})
แปลงสตริงเป็นเซต -> frozenset({'1', '2'})
```



## สรุปก่อนจบบท

เนื้อหาในบทนี้เราได้รู้จักกับโครงสร้างข้อมูลภาษาไพธอน ได้แก่ ข้อมูลชนิดลิสต์ ข้อมูลชนิดทูเพิล ข้อมูลชนิดดิกชันนารี และข้อมูลชนิดเซต ซึ่งเป็นข้อมูลชนิดที่มีการจัดเก็บข้อมูลแบบลำดับและสามารถจัดเก็บข้อมูลได้หลายชนิด อย่างไรก็ตามข้อมูลชนิดเหล่านี้มีความแตกต่างกันคือ การแก้ไขหรือเปลี่ยนแปลงข้อมูล ดังนั้นเราจะต้องระมัดระวังหรือเลือกข้อมูลชนิดเพื่อใช้งานให้ถูกต้อง

## แบบฝึกหัด

1. กำหนดให้

```
months=['Jan', 'May', 'Jul', 'Aug', 'Oct', 'Dec']
```

- (a) ให้เขียนคำสั่งโปรแกรมเรียกใช้เมธอดแทรกรดเดือนที่ขาดหายไป
- (b) ให้เขียนคำสั่งโปรแกรมเรียกใช้เมธอดลบชื่อเดือนตำแหน่งที่ 2, 5, 9 ออกจากลิสต์
- (c) ให้เขียนคำสั่งโปรแกรมเรียกใช้เมธอดแสดงชื่อเดือนที่เหลืออยู่ในลิสต์

2. กำหนดให้

```
days = ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri' , 'Sat']
```

- (a) ให้เขียนคำสั่งโปรแกรมเรียกใช้เมธอดเรียงชื่อวันจากท้ายสุด
- (b) ให้เขียนคำสั่งโปรแกรมเรียกใช้เมธอดเรียงลำดับชื่อวันตามตัวอักษร
- (c) ให้เขียนคำสั่งโปรแกรมแสดงชื่อวันในตำแหน่งที่ 0, 5 และ 6

3. กำหนดให้

```
brand_cars = ('Toyota', 'Honda', 'Benz', 'BMW', 'Tesla',  
              'Ford' , 'KIA', 'Volvo')
```

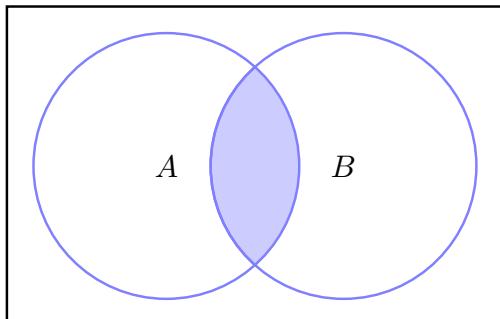
- (a) ให้เขียนคำสั่งโปรแกรมแสดงตำแหน่งของ 'Benz', 'Ford' และ 'Volvo'
- (b) ให้เขียนคำสั่งโปรแกรมแสดงจำนวนข้อมูลทั้งหมดในทุกเพล
- (c) ให้เขียนคำสั่งโปรแกรมตรวจสอบมี耶ห้อรถ 'Suzuki', 'Ferrary', 'Ford' อยู่ใน  
 brand\_cars หรือไม่

4. จงเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิดติกซันนารี เก็บรายชื่อเดือนทั้งหมดพร้อมทั้งแสดงตัวอย่าง การเรียกใช้งานเมธอดดังนี้

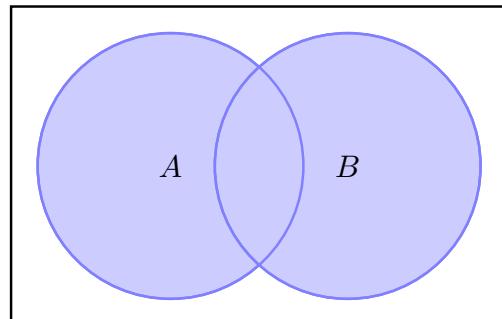
- (a) เรียกใช้งานเมธอดแสดงค่าข้อมูลของ Key ในเดือนมกราคม, ตุลาคม และ ธันวาคม
- (b) เรียกใช้งานเมธอดลบค่า Key ในเดือนมีนาคม และ พฤศจิกายน
- (c) เรียกใช้งานเมธอดแสดงค่า Value ทั้งหมดในติกซันนารี

5. จงเขียนคำสั่งโปรแกรมสร้างข้อมูลชนิดติกซันนารีเก็บรายชื่อเพื่อนที่สนใจจำนวน 3 คน กำหนดให้ ค่าข้อมูล Value ต้องประกอบด้วยชื่อ – นามสกุล ชื่อเล่น และเบอร์โทรศัพท์ และให้แสดงตัวอย่าง การเรียกใช้งานเมธอดดังนี้

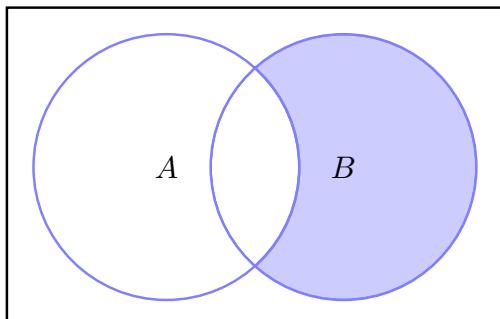
- (a) เรียกใช้งานเมธอดเพิ่มชื่อเพื่อนเข้าไปในดิกชันนารีอีก 2 คน และแสดงผลข้อมูลทั้งหมดที่มีอยู่ในดิกชันนารี
- (b) ให้เขียนคำสั่งโปรแกรมแสดงชื่อเพื่อนทั้งหมดที่มีอยู่ในดิกชันนารี
- (c) ให้เขียนคำสั่งโปรแกรมแสดงชื่อเพื่อนและเบอร์โทรศัพท์ที่มีอยู่ในดิกชันนารี
6. กำหนดให้ A และ B เป็นตัวแปรของข้อมูลชนิดเซต จงเขียนนิพจน์ในภาษาโปรแกรมไพรอนที่ สอดคล้องกับแผนภาพเวนน์-อยเลอร์ที่กำหนดให้ต่อไปนี้



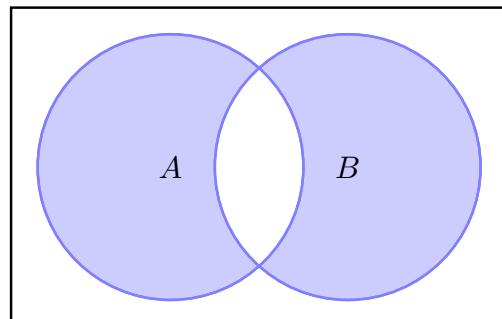
(a)



(b)



(c)



(d)

7. กำหนดให้

```
x = {'cat', 'dog', 'fish', 'bird', 'bee'}
```

```
y = {'snake', 'lion', 'pig', 'dog', 'cat'}
```

- (a) จงเขียนคำสั่งโปรแกรมหาผลลัพธ์สมาชิกที่เหมือนกันของเซต x และ y
- (b) จงเขียนคำสั่งโปรแกรมหาสมาชิกที่อยู่ในเซต x แต่ไม่อยู่ในเซต y
- (c) จงเขียนคำสั่งโปรแกรมหาสมาชิกที่อยู่ในเซต x แต่ไม่อยู่ในเซต y และอยู่ในเซต y แต่ไม่อยู่ในเซต x



## บทที่ 6

# คำสั่งควบคุมทิศทางการทำงานโปรแกรม

ในบทนี้เราจะได้เรียนรู้การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงาน (Control Statement) ซึ่งแบ่งออกได้ทั้งหมด 3 รูปแบบ ได้แก่ การเขียนคำสั่งโปรแกรมควบคุมทิศทางแบบลำดับ (Sequence Control Statement), แบบมีเงื่อนไข (Conditions Control Statement) และแบบทำซ้ำ (Iteration Control Statement)

### 6.1 รู้จักกับผังงาน

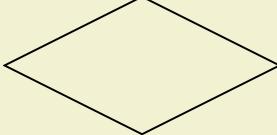
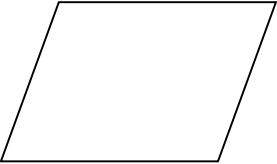
โดยพื้นฐานแล้วความสามารถใช้ให้ผังงาน (Flowchart) ในการเรียนรู้การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงาน ซึ่งผังงานก็คือแผนภาพสัญลักษณ์แสดงขั้นตอนจากเริ่มต้นจนสิ้นสุดในกระบวนการแก้ปัญหา (โดยทั่วไปแผนภาพจะเรียกลำดับขั้นตอนจากด้านบนลงด้านล่าง) ช่วยให้เราลำดับขั้นตอนตรวจสอบ และแก้ไขกระบวนการแก้ปัญหาได้ง่าย ผังงานประกอบไปด้วยสัญลักษณ์ที่ใช้แสดงถึงแต่ขั้นตอน ทั้งนี้มาตรฐาน ANSI<sup>1</sup> และ ISO<sup>2</sup> 5807:1985 ได้กำหนดสัญลักษณ์ในการสร้างผังงาน ดังตาราง 6.1

สัญลักษณ์	ชื่อเรียก	คำอธิบาย
	ศรีทิศทาง (Flowline)	ใช้ระบุทิศทางของกระบวนการ อาจระบุคำอธิบายในสัญลักษณ์นี้เพื่ออธิบายกระบวนการเพิ่มเติมได้ตามจำเป็น
	เทอร์มินัล (Terminal)	ใช้ระบุจุดเริ่มต้นหรือจุดสิ้นสุดของการกระบวนการ โดยต้องระบุชื่อย่างเช่น START, BEGIN, TERMINATE หรือ STOP ในสัญลักษณ์นี้

ตาราง 6.1: สัญลักษณ์ของผังงานตามมาตรฐาน ANSI/ISO (มีต่อ)

<sup>1</sup>American National Standards Institute: ANSI

<sup>2</sup>The International Organization for Standardization: ISO

สัญลักษณ์	ชื่อเรียก	คำอธิบาย
	กระบวนการ (Process)	ใช้แทนคำสั่ง หรือการทำงานพร้อมระบุคำอธิบายอย่างสั้นในสัญลักษณ์นี้
	การตัดสินใจ (Decision)	ใช้แทนการทำงานที่มีเงื่อนไข โดยต้องระบุทิศทางของกระบวนการต่อไปสองกระบวนการที่ต่างกัน และระบุเงื่อนไขที่ให้ค่าจริงหรือเท็จในสัญลักษณ์นี้
	การรับเข้า/ส่งออก (Input/Output)	ใช้แทนการทำงานที่มีการรับเข้าหรือส่งออกข้อมูล โดยระบุว่าแสดงหรือนำเข้าข้อมูลใดในสัญลักษณ์นี้
	กระบวนการที่มีการนิยามแล้ว (Pre-defined Process)	ใช้แทนหนึ่งกระบวนการที่มีการบอกชื่อกระบวนการไว้ก่อนแล้ว ณ ตำแหน่งอื่น
	ตัวเชื่อมในหน้า (On-Page Connector)	ระบุชื่อในสัญลักษณ์นี้ โดยใช้เป็นคู่เพื่อเชื่อมกระบวนการ เมื่อการเขียนผังงานเกิดครบทิศทางทั่วซ้อน หรืออาจทำให้เกิดความสับสนในผังงาน ซึ่งตัวเชื่อมคุณนี้ต้องอยู่ในหน้าเดียวกัน
	ตัวเชื่อมนอกหน้า (Off-Page Connector)	ระบุชื่อในสัญลักษณ์นี้ โดยใช้เป็นจุดเชื่อมกระบวนการ เมื่อเขียนผังงานที่ยาวเกินกว่าหนึ่งหน้า

ตาราง 6.1: สัญลักษณ์ของผังงานตามมาตรฐาน ANSI/ISO

## 6.2 การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบลำดับ

การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบลำดับ เป็นลักษณะการเขียนคำสั่งให้โปรแกรมทำงานจากบนลงล่าง โดยให้ทำงานตามคำสั่งที่ 1, คำสั่งที่ 2, คำสั่งที่ 3 ไปจนถึงครบทุกคำสั่ง ไม่มีคำสั่งการตัดสินใจหรือมีคำสั่งการทำซ้ำเข้ามาเกี่ยวข้อง และไม่มีคำสั่งให้กระโดดข้ามไปทำงานในคำสั่งอื่นแล้วกลับมาทำคำสั่งเดิมอีกรอบ

รูปแบบการเขียนผังงานของการเขียนโปรแกรมควบคุมทิศทางการทำงานแบบลำดับแสดง ดังรูป 6.1 และเราได้ให้ตัวอย่างการเขียนแผนภาพผังงานคำนวนหาพื้นที่สี่เหลี่ยมผืนผ้า และการเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบลำดับคำนวนรายได้สุทธิ ดังต่อไปนี้

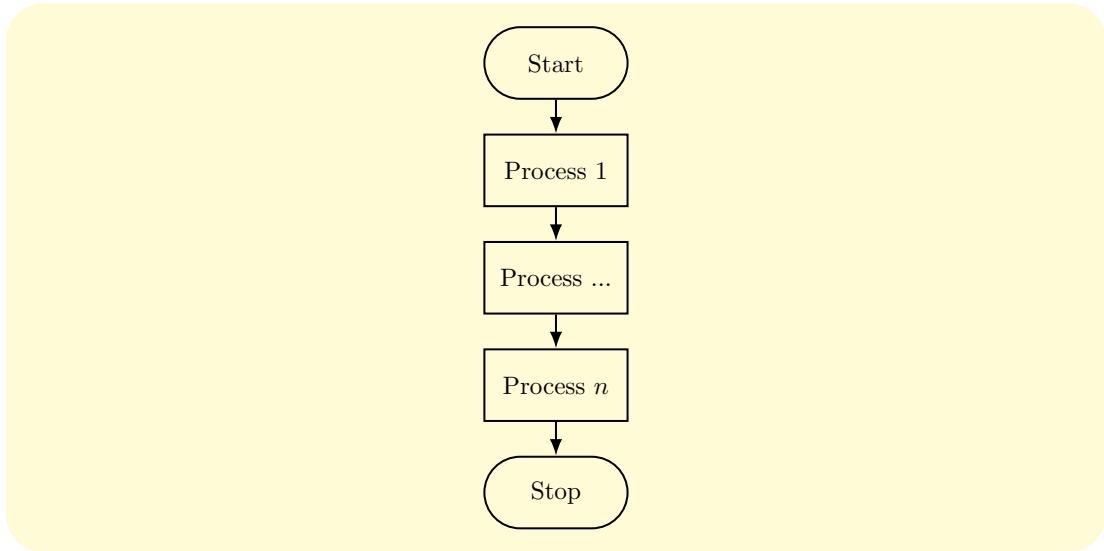
### ตัวอย่าง 6.1

การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบลำดับคำนวนหาพื้นที่สี่เหลี่ยมผืนผ้า

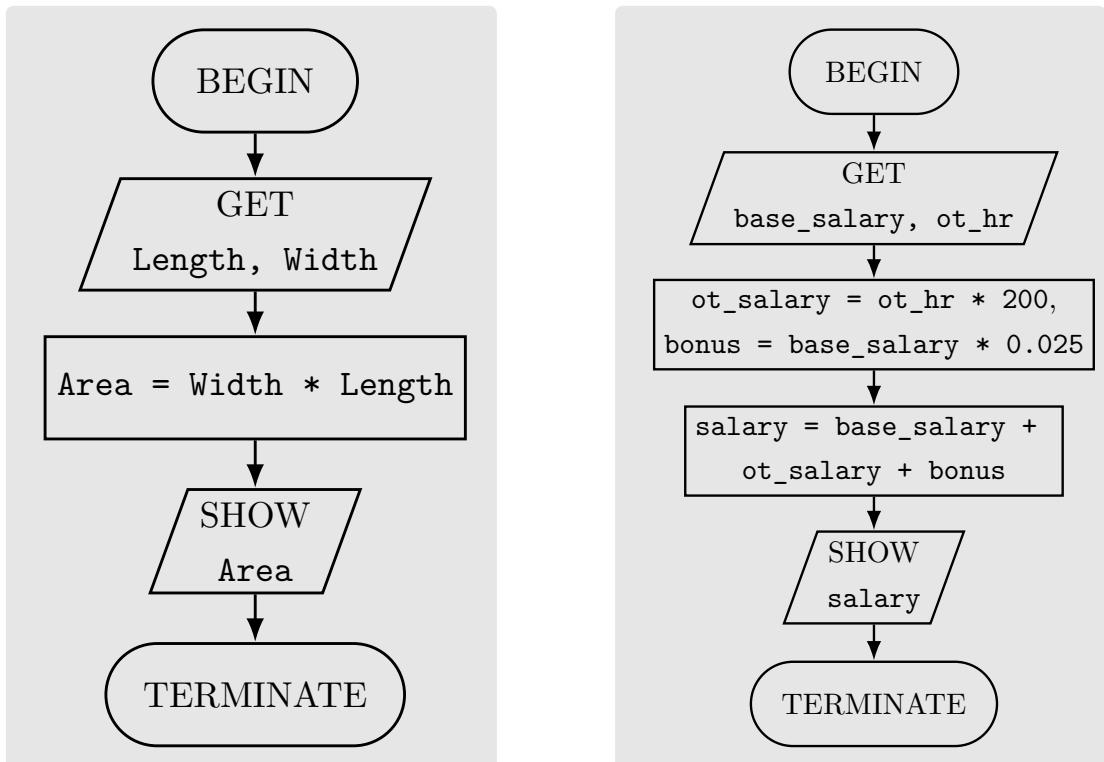
```
1 # รับค่าการป้อนข้อมูลด้านความยาวแล้วแปลงเป็นจำนวนทศนิยม
2 x = float(input('กรุณาป้อนความยาวของสี่เหลี่ยมผืนผ้า: '))
3
4 # รับค่าการป้อนข้อมูลด้านความกว้างแล้วแปลงเป็นจำนวนทศนิยม
5 y = float(input('กรุณาป้อนความกว้างของสี่เหลี่ยมผืนผ้า: '))
6
7 # คำนวณค่าตัวแปร x กับ y ผลลัพธ์เก็บไว้ในตัวแปร area
8 area = x * y
9
10 # แสดงผลลัพธ์ตัวแปร area
11 print('พื้นที่สี่เหลี่ยมผืนผ้า = ', area, 'ตารางหน่วย')
```

กรุณาป้อนความยาวของสี่เหลี่ยมผืนผ้า: 12  
กรุณาป้อนความกว้างของสี่เหลี่ยมผืนผ้า: 5.5  
พื้นที่สี่เหลี่ยมผืนผ้า = 66.0 ตารางหน่วย





รูป 6.1: ผังงานคำสั่งควบคุมทิศทางแบบลำดับแบบทั่วไป



รูป 6.2: ตัวอย่างผังงานคำสั่งควบคุมทิศทางแบบลำดับ

## ตัวอย่าง 6.2

การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบลำดับคำนวณรายได้สุทธิ

```
1 base_salary = float(input('กรอกฐานเงินเดือน: '))
2 ot_hr = float(input('กรอกจำนวนชั่วโมงทำงานล่วงเวลา: '))
3 ot_salary = ot_hr * 200
4 bonus = base_salary * 0.025
5 salary = base_salary + ot_salary + bonus
6 print(f'ค่าทำงานล่วงเวลา {ot_hr} * 200 = {ot_salary:.2f} บาท')
7 print(f'โบนัส {base_salary} * 0.025 = {bonus:.2f} บาท')
8 print(f'เงินเดือนสุทธิ = {salary:.2f} + {ot_salary:.2f} + {bonus:.2f} =
   {salary:.2f} บาท')
```

```
กรอกฐานเงินเดือน: 15000
กรอกจำนวนชั่วโมงทำงานล่วงเวลา: 20
ค่าทำงานล่วงเวลา 20.0 * 200 = 4000.00 บาท
โบนัส 15000.0 * 0.025 = 375.00 บาท
เงินเดือนสุทธิ = 19375.00 + 4000.00 + 375.00 = 19375.00 บาท
```



## 6.3 การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบมีเงื่อนไข

การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบมีเงื่อนไข (Condition Control Statement) ใช้เพื่อควบคุมทิศทางการทำงานหรือกระบวนการการทำงานต่าง ๆ ของโปรแกรมให้เป็นไปตามเงื่อนไข โดยคำสั่งโปรแกรมจะให้มีการตัดสินใจเลือกทิศทางการทำงานอย่างใดอย่างหนึ่งตามเงื่อนไขที่กำหนดก่อนทำงานอย่างอื่นต่อไป สำหรับในภาษาไพธอนจะใช้คำสั่ง **if** ที่มีการใช้งานอยู่ 4 แบบ ได้แก่

1. **if**,
2. **if ... else**,
3. **if ... elif ... else** และ
4. **if** ซ้อนกัน (Nested If)

โดยรายละเอียดการใช้งานคำสั่งแต่ละแบบ จะถูกกล่าวในหัวข้ออยู่ถัดไป ดังนี้

### 6.3.1 คำสั่ง `if`

คำสั่ง `if` ใช้ตรวจสอบเงื่อนไขจากนิพจน์ (condition) ที่สร้างขึ้นมาว่าเป็นจริงหรือเท็จ ถ้าผลการตรวจสอบเงื่อนไขนิพจน์เป็น `True` คำสั่งโปรแกรมที่อยู่ในขอบเขตคำสั่ง `if` จะทำงาน ถ้าผลการตรวจสอบเงื่อนไขนิพจน์เป็นเท็จ `False` คำสั่งโปรแกรมที่อยู่นอกขอบเขตคำสั่ง `if` จะทำงาน และอย่าลืมใส่เครื่องหมาย `colon(:)` ปิดท้ายคำสั่งเงื่อนไขคำสั่ง `if` เสมอ และต้อง ย่อหน้า เพื่อเป็นการกำหนดขอบเขตการทำงานของคำสั่งโปรแกรมด้วย

ในการกำหนดขอบเขตคำสั่ง รูปแบบการเขียนคำสั่ง `if` มีลักษณะโครงสร้างโปรแกรม และผังงานดังต่อไปนี้

โครงสร้างโปรแกรมควบคุมการทำงานแบบมีเงื่อนไข `if`

```
if condition:          # ตรวจสอบเงื่อนไขเป็นจริงหรือเท็จ
    statement_1        # ชุดคำสั่งที่ 1 เมื่อเงื่อนไขเป็นจริง
    ...
    statement_n        # ชุดคำสั่งที่ n เมื่อเงื่อนไขเป็นจริง
```

จากแผนภาพผังงานในรูป 6.4a และ 6.4b นำมาเขียนเป็นคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบมีเงื่อนไข `if` ในตัวอย่าง 6.3 และ 6.4 ได้ดังนี้

#### ตัวอย่าง 6.3

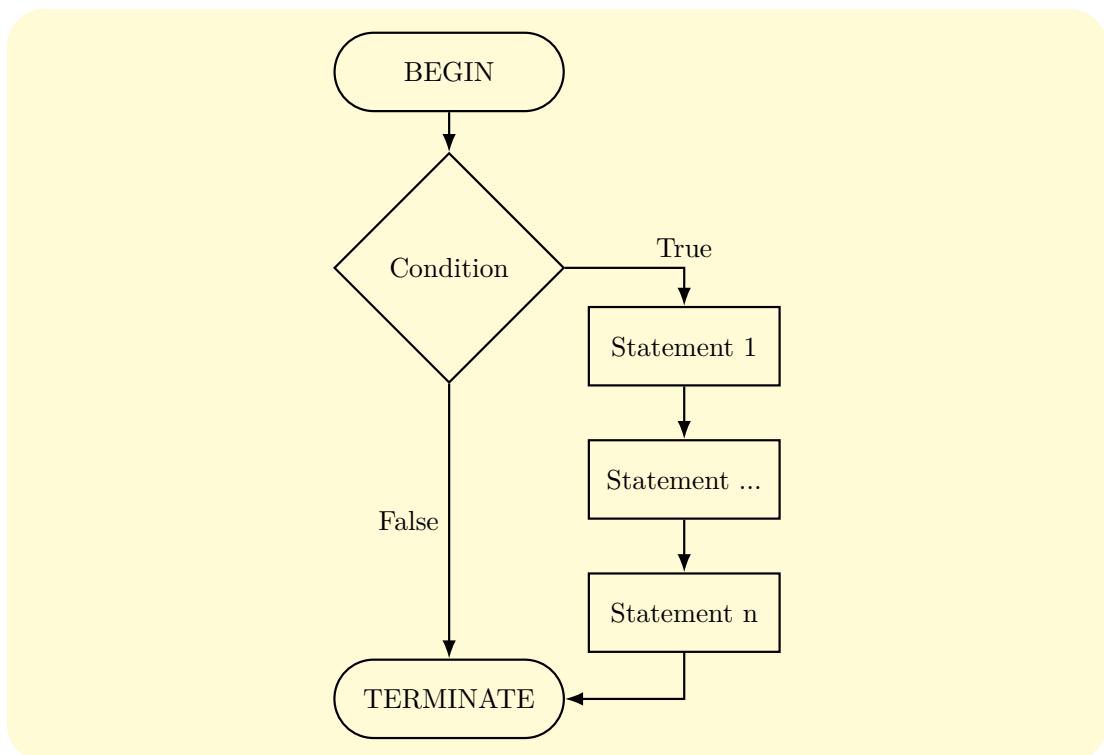
การเขียนคำสั่งโปรแกรมแบบมีเงื่อนไขด้วย `if` ตรวจสอบอายุ

```
1 age = int(input('กรุณาป้อนอายุของคุณ = ')) # รอรับค่าการป้อนข้อมูลเป็นจำนวนเต็ม
2 if age > 35: # ตรวจสอบค่าตัวแปร age มากกว่า 35 หรือไม่
3     print('You are old.') # แสดงผลลัพธ์
4 print('The program is done !!') # แสดงผลหลังจากโปรแกรมทำงานทุกครั้ง
```

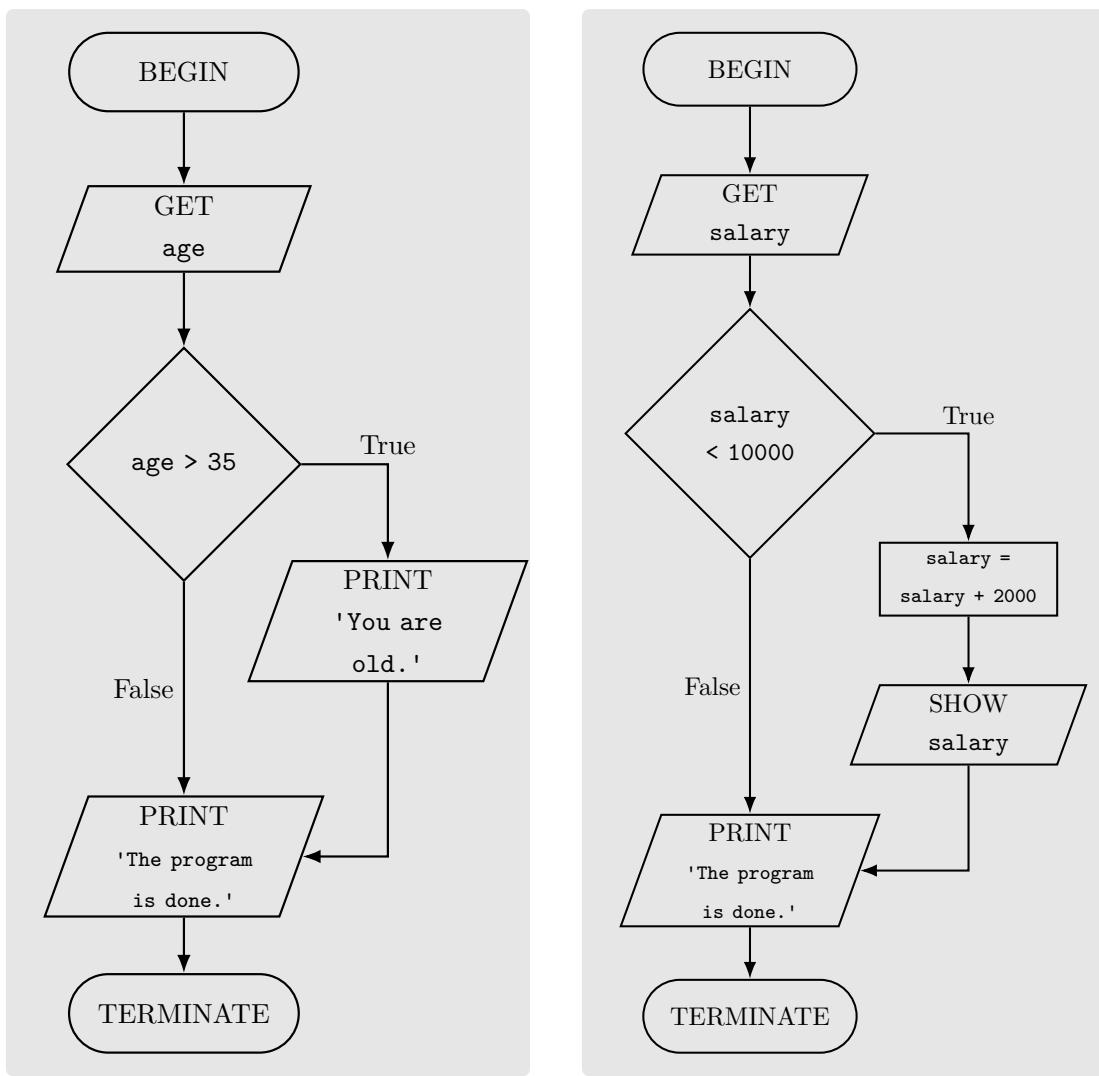
กรุณาป้อนอายุของคุณ = 45  
You are old.  
The program is done !!

กรุณาป้อนอายุของคุณ = 15  
The program is done !!

จากตัวอย่าง 6.3 เมื่อผู้ใช้งานป้อนอายุไม่เกินกว่า **35** ทำให้เงื่อนไขเป็นเท็จ โปรแกรมจะแสดงผลคำว่า '`The program is done`' เท่านั้น



รูป 6.3: ผังงานการทำงานแบบมีเงื่อนไข



รูป 6.4: ตัวอย่างผังงานคำสั่งควบคุมทิศทางแบบมีเงื่อนไข

#### ตัวอย่าง 6.4

การเขียนคำสั่งโปรแกรมแบบมีเงื่อนไขด้วย **if** ตรวจสอบเงินเดือน

```
1 salary = int(input('กรุณาป้อนเงินเดือน = '))
2 if salary < 10000:
3     salary = salary + 2000
4     print('รายได้สุทธิ = ', salary)
5 print('The program is done !!')
```

กรุณาป้อนเงินเดือน = 9500  
รายได้สุทธิ = 11500  
The program is done !!

กรุณาป้อนเงินเดือน = 14500  
The program is done !!



จากตัวอย่าง 6.4 เมื่อป้อนเงินเดือนน้อยกว่า **10000** ส่งผลให้เงื่อนไขเป็นจริง ซึ่งจะนำจำนวนเงินเดือนที่ป้อนมาบวกกับค่าโบนัส **2000** ถ้าป้อนเงินเดือนมากกว่าหรือเท่ากับ **10000** ส่งผลให้เงื่อนไขเป็นเท็จ ดังนั้นจึงไม่นำเงินเดือนที่ป้อนมาบวกกับค่าโบนัส และโปรแกรมจะแสดงผลคำว่า '**The program is done**' เท่านั้น

#### ตัวอย่าง 6.5

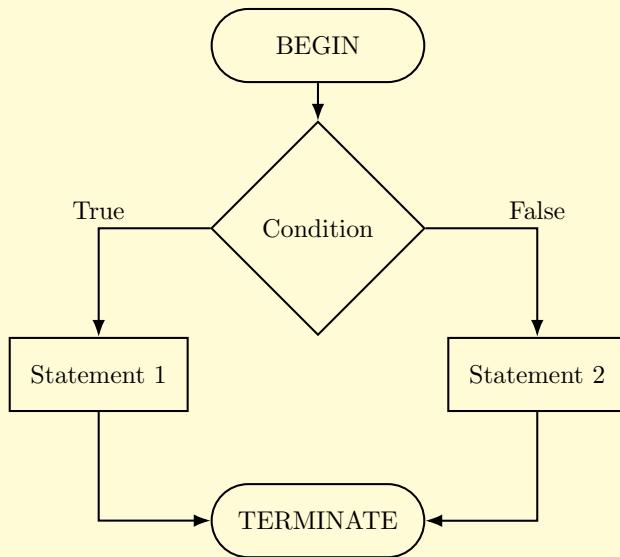
การเขียนคำสั่งโปรแกรมแบบมีเงื่อนไขด้วย **if** ตรวจสอบว่าจำนวนเต็มอยู่ในช่วงปิดที่กำหนดไว้

```
1 n = int(input('Enter an integer: ')) # รับค่าจำนวนเต็ม n
2 my_interval = [1, 100] # สร้างตัวแปรชนิดลิสต์ แทนช่วงปิดที่กำหนด
3 # ตรวจสอบเงื่อนไข n มีค่าอยู่ในช่วงปิด [1, 100]
4 if n >= my_interval[0] and n <= my_interval[1]:
5     print(f'{n} is a member of {my_interval}') # การกระทำเมื่อเงื่อนไขเป็นจริง
6 print('Program terminated')
```

Enter an integer: 55  
55 is a member of [1, 100]  
Program terminated

Enter an integer: 555  
Program terminated





รูป 6.5: ผังงานของการทำงานแบบมีเงื่อนไข

### 6.3.2 คำสั่ง `if ... else`

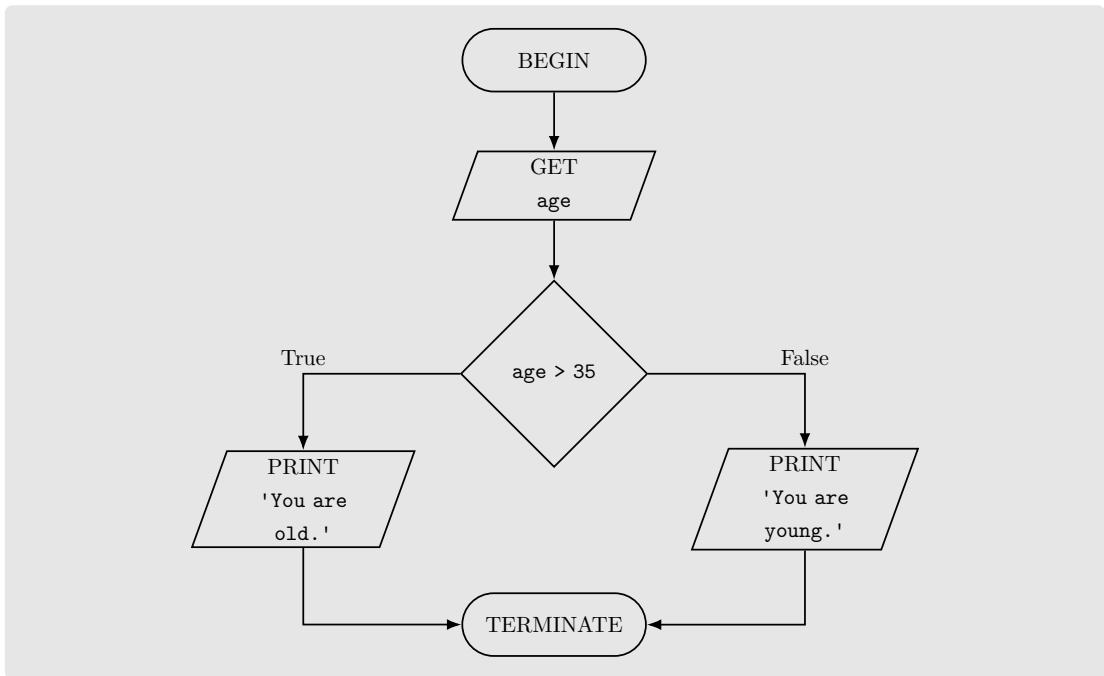
คำสั่ง `if ... else` ควบคุมการทำงานแบบมีเงื่อนไข 2 ทิศทาง ซึ่งจะมีการตรวจสอบเงื่อนไขที่ถูกกำหนดขึ้นในนิพจน์เหมือนกันกับคำสั่ง `if` ว่าเป็นจริง (`True`) หรือเท็จ (`False`) ถ้าผลเป็นจริงจะทำงานตามคำสั่งที่อยู่หลัง `if` ถ้าผลเป็นเท็จจะทำงานตามคำสั่งที่อยู่หลัง `if` รูปแบบการเขียนคำสั่งโปรแกรมแบบ `if ... else` มีลักษณะดังต่อไปนี้ (ต้องใส่เครื่องหมาย Colon (:)) หลังเงื่อนไข `if` และคำสั่ง `else` ด้วย)

โครงสร้างโปรแกรมควบคุมการทำงานแบบมีเงื่อนไข `if...else`

```

if condition:          # ตรวจสอบเงื่อนไขเป็นจริงหรือเท็จ
    statement_1        # คำสั่งที่ให้ทำงานเมื่อเงื่อนไขเป็นจริง
else:
    statement_2        # คำสั่งที่ให้ทำงานเมื่อเงื่อนไขเป็นเท็จ
  
```

จากโครงสร้างของคำสั่งแบบมีเงื่อนไข `if...else` นำมาเขียนเป็นแผนภาพผังงานได้ดังรูป 6.5 และแสดงตัวอย่างการเขียนผังงานตรวจสอบอายุ ดังรูป 6.6 กำหนดเงื่อนไขคือ ถ้าอายุน้อยกว่า 35 ให้แสดงคำว่า 'You are young' ถ้าอายุมากกว่า 35 ให้แสดงคำว่า 'You are old'



รูป 6.6: ผังงานของตัวอย่าง 6.7

เมื่อนำมาเขียนเป็นคำสั่งโปรแกรมแบบ **if...else** โดยนำโปรแกรมจากตัวอย่าง 6.3 มาปรับปรุงเพิ่มเติม แสดงได้ดังตัวอย่าง 6.7

### ตัวอย่าง 6.6

การเขียนคำสั่งโปรแกรมตรวจสอบเงื่อนไขด้วยคำสั่ง **if...else** ตรวจสอบอายุ

```

1 age = int(input('กรุณาป้อนอายุของคุณ = ')) # รอรับการป้อนข้อมูลเป็นจำนวนเต็ม
2 if age > 35: # ตรวจสอบค่าตัวแปร age มากกว่า 35 หรือไม่
3     print('You are old.') # แสดงผลลัพธ์
4 else: # ตรวจสอบแล้วเป็นเท็จคือค่าตัวแปร age น้อยกว่า 35
5     print('You are young.')# แสดงผลลัพธ์
  
```

กรุณาป้อนอายุของคุณ = 57  
You are old.

กรุณาป้อนอายุของคุณ = 25  
You are young.



### ตัวอย่าง 6.7

การเขียนคำสั่งโปรแกรมตรวจสอบเงื่อนไขด้วยคำสั่ง `if...else` ตรวจสอบชนิดของจำนวนเต็ม

```
1 n = int(input('Enter an integer: ')) # รับค่าจำนวนเต็ม n
2 m = n % 2 # คำนวณเศษจากการหาร n ด้วย 2
3 if m == 0: # ตรวจสอบเงื่อนไขการหารลงตัว
4     print(f'{n} is even') # การกระทำเมื่อเงื่อนไขเป็นจริง
5 else:
6     print(f'{n} is odd') # การกระทำเมื่อเงื่อนไขเป็นเท็จ
```

```
Enter an integer: 30
30 is even
```



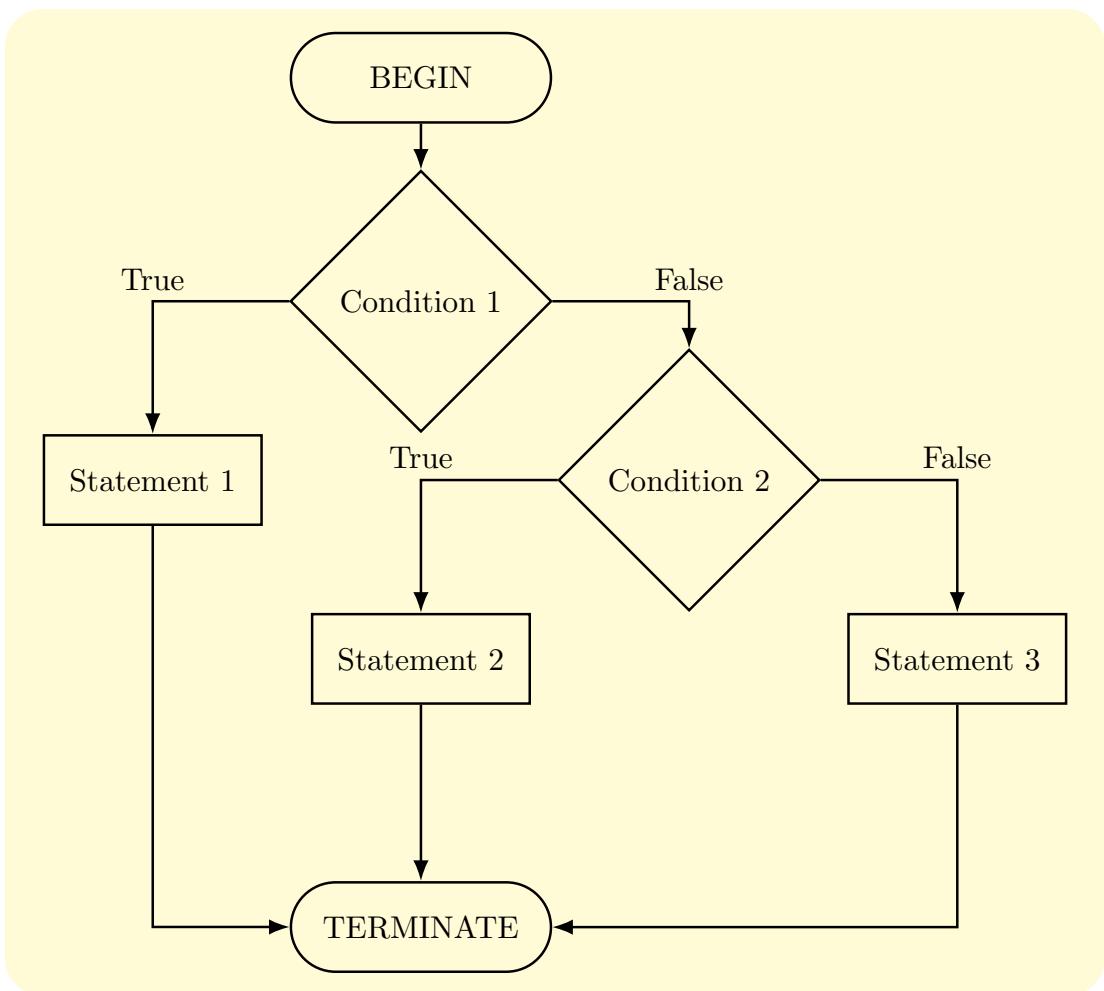
```
Enter an integer: 447
447 is odd
```

### 6.3.3 คำสั่ง `if ... elif ... else`

คำสั่ง `if ... elif ... else` ควบคุมทิศทางการทำงานของโปรแกรมแบบมีหลายเงื่อนไข โดยมีการสร้างนิพจน์เป็นตัวตรวจสอบเงื่อนไขเหมือนกับคำสั่ง `if` และ `if...else` แต่คำสั่ง `if ... elif ... else` สามารถตรวจสอบได้หลายเงื่อนไขมากกว่า โดยมีลักษณะการตรวจสอบเงื่อนไขเรื่อยๆ จนกว่าจะพบเงื่อนไขที่เป็นจริงแล้วจึงแสดงผลตามคำสั่งโปรแกรมที่กำหนด ถ้าตรวจสอบเงื่อนไขนิพจน์แล้วเป็นเท็จ จะทำงานตามคำสั่งโปรแกรมที่อยู่หลัง `else` มีโครงสร้างรูปแบบการเขียนคำสั่งโปรแกรมแสดงดังต่อไปนี้

โครงสร้างรูปแบบการเขียนคำสั่ง `if ... elif ... else`

```
if condition_1: # ตรวจสอบเงื่อนไข condition_1
    statement_1 # คำสั่งเมื่อ condition_1 เป็นจริง
elif condition_2: # ตรวจสอบเงื่อนไข condition_2
    statement_2 # คำสั่งเมื่อ condition_2 เป็นจริง
else:
    statement_3 # คำสั่งเมื่อเงื่อนไขด้านบนทั้งหมดเป็นเท็จ
```



รูป 6.7: ผังงานของการทำงานแบบมีเงื่อนไข `if ... elif ... else`

จากโครงสร้างรูปแบบการเขียนคำสั่งควบคุมทิศทางการทำงานแบบมีเงื่อนไข **if ... elif ... else** เมื่อนำมาใช้ยนเป็นผังงานจะได้ดังรูป 6.7 และตัวอย่างต่อไปนี้เป็นคำสั่งโปรแกรมแสดงการทำงานแบบมีเงื่อนไข **if ... elif ... else** ซึ่งรูป 6.8 แสดงผังงานของการเขียนคำสั่งโปรแกรมควบคุมทิศทางแบบมีเงื่อนไข **if ... elif ... else** ตรวจสอบขนาดไส้เลื่อยในตัวอย่าง 6.11

### ตัวอย่าง 6.8

การเขียนคำสั่งโปรแกรมควบคุมทิศทางแบบมีเงื่อนไข **if ... elif ... else**

```
1 a = 200
2 b = 33
3 if b > a:
4     print('b is greater than a')
5 elif a == b:
6     print('a and b are equal')
7 else:
8     print('a is greater than b')
```

a is greater than b



### ตัวอย่าง 6.9

การเขียนคำสั่งโปรแกรมควบคุมทิศทางแบบมีเงื่อนไข **if ... elif**

```
1 a = 33
2 b = 33
3 if b > a:
4     print('b is greater than a')
5 elif a == b:
6     print('a and b are equal')
```

a and b are equal



### ตัวอย่าง 6.10

การเขียนคำสั่งโปรแกรมควบคุมทิศทางแบบมีเงื่อนไข `if ... elif ... else` ในการตรวจสอบชนิดของจำนวนเต็ม

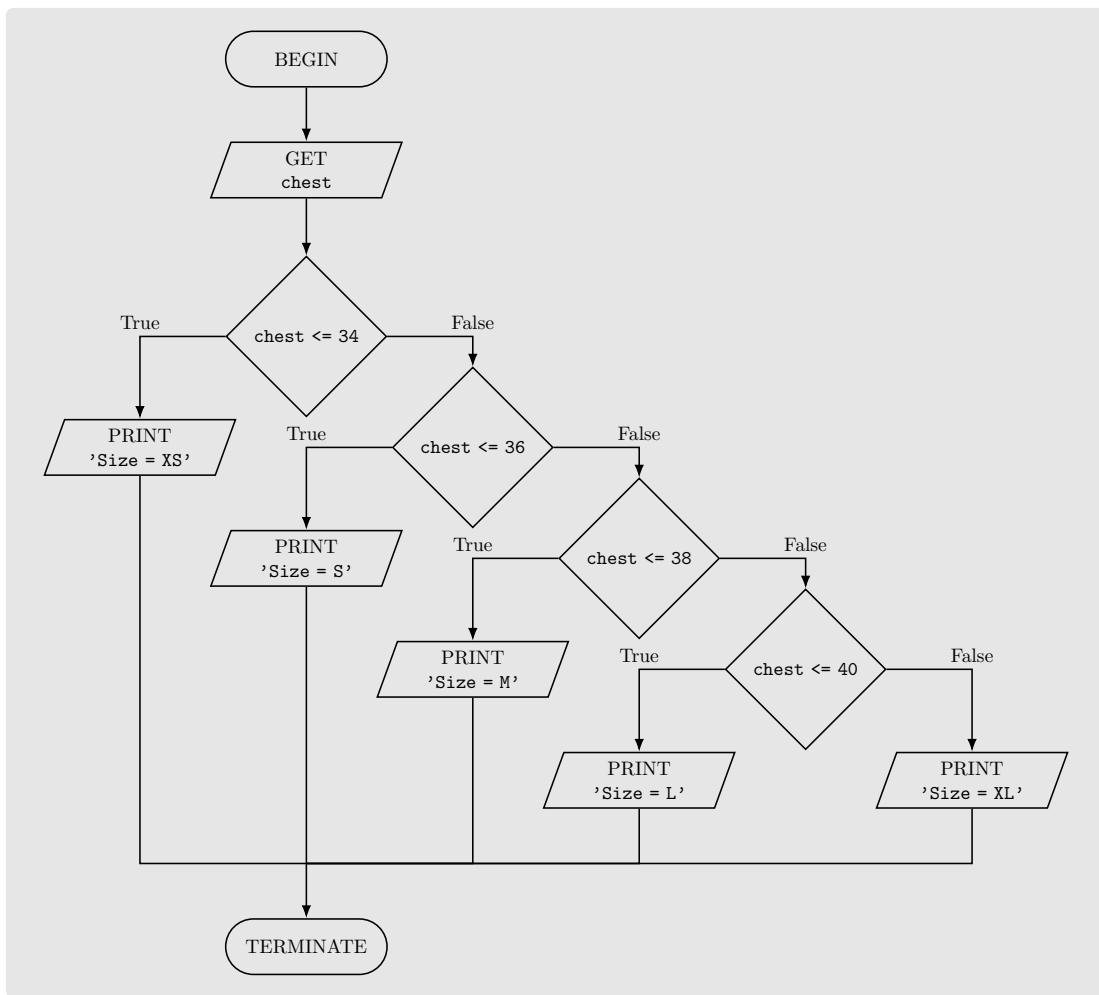
```
1 n = int(input('Enter an integer: '))
2 if n > 0:
3     print(f'{n} is positive')
4 elif n < 0:
5     print(f'{n} is negative')
6 else:
7     print(f'{n} is Zero')
```

```
Enter an integer: -345
-345 is negative
```



```
Enter an integer: 222
222 is positive
```

```
Enter an integer: 0
0 is Zero
```



ສູບ 6.8: ພັນຍານຂອງຕົວຢ່າງ 6.11

### ตัวอย่าง 6.11

การเขียนคำสั่งโปรแกรมควบคุมทิศทางแบบมีเงื่อนไข `if ... elif ... else` ตรวจสอบขนาดไซส์เสื้อ

```
1 chest = int(input('กรุณาป้อนความกว้างอก: ')) # รอรับการป้อนข้อมูลเป็นจำนวนเต็ม
2 if chest <= 34: # ตรวจสอบค่าตัวแปร chest "ไม่เกินกว่า 34 หรือไม่
3     print('Size = XS') # แสดงผลลัพธ์ ถ้าเงื่อนไขบรรทัดที่ 2 เป็นจริง
4 elif chest <= 36: # ตรวจสอบค่าตัวแปร chest "ไม่เกินกว่า 36 หรือไม่
5     print('Size = S') # แสดงผลลัพธ์ ถ้าเงื่อนไขบรรทัดที่ 4 เป็นจริง
6 elif chest <= 38: # ตรวจสอบค่าตัวแปร chest "ไม่เกินกว่า 38 หรือไม่
7     print('Size = M') # แสดงผลลัพธ์ ถ้าเงื่อนไขบรรทัดที่ 6 เป็นจริง
8 elif chest <= 40: # ตรวจสอบค่าตัวแปร chest "ไม่เกินกว่า 40 หรือไม่
9     print('Size = L') # แสดงผลลัพธ์ ถ้าเงื่อนไขบรรทัดที่ 8 เป็นจริง
10 else: # ถ้าค่าตัวแปร chest มากกว่า 40
11     print('Size = XL') # แสดงผลลัพธ์ ถ้าไม่ตรงกับเงื่อนไขใดๆ
```

กรุณาป้อนความกว้างอก: 50  
Size = XL

กรุณาป้อนความกว้างอก: 37  
Size = M

กรุณาป้อนความกว้างอก: 35  
Size = S

กรุณาป้อนความกว้างอก: 40  
Size = L

กรุณาป้อนความกว้างอก: 30  
Size = XS

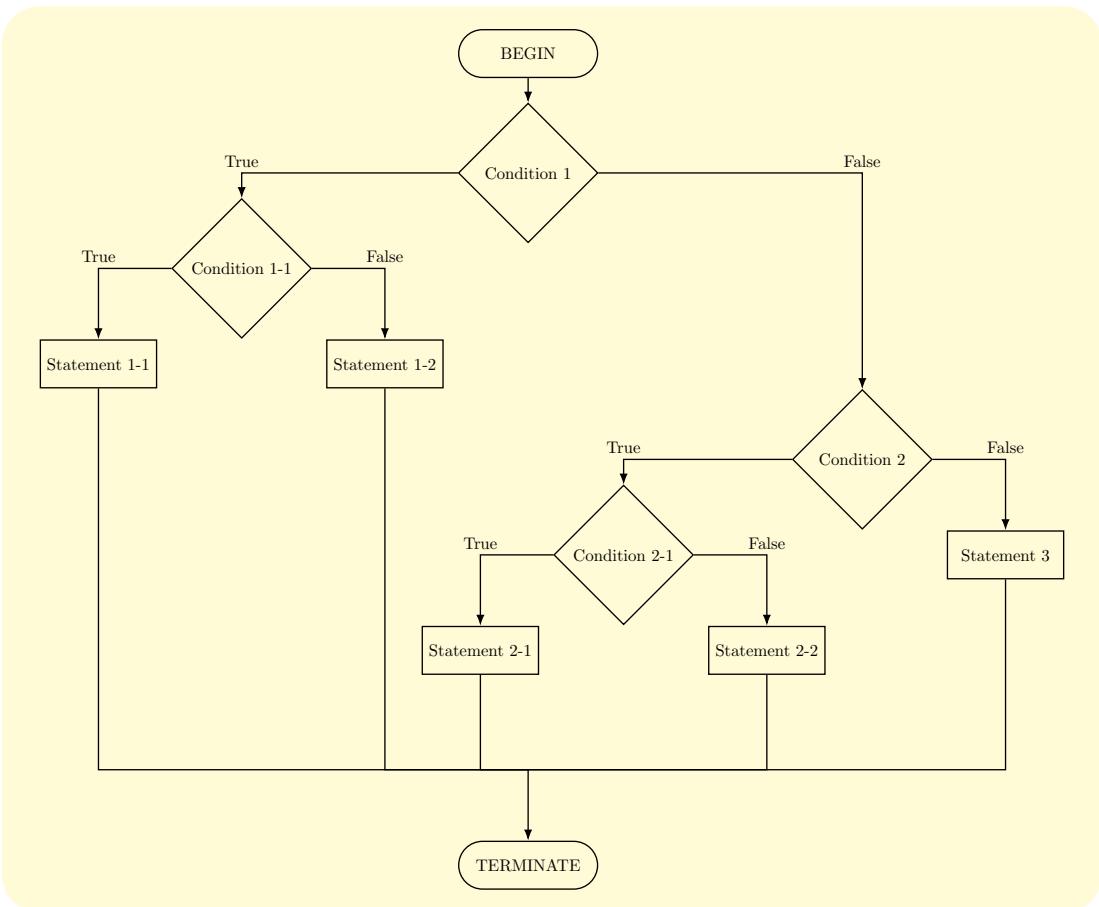
#### 6.3.4 คำสั่ง if ซ้อนกัน

คำสั่ง **if** ซ้อนกัน (Nested If) ใช้ควบคุมทิศทางการทำงานของโปรแกรมแบบเงื่อนไขซ้อนเงื่อนไข หรือ **if** ซ้อน **if** นั่นเอง ซึ่งจะช่วยให้เราเขียนโปรแกรมสำหรับตรวจสอบเงื่อนไขที่มีความซับซ้อนได้มากขึ้น ลักษณะการทำงานของคำสั่ง Nested If ถ้าเงื่อนไขขอบเขตนอกเป็นจริงแล้วจะเข้าไปตรวจสอบเงื่อนไขที่อยู่ภายในได้โดยอัตโนมัติ จนกว่าจะพบเงื่อนไขที่เป็นจริงหรือเป็นตามที่กำหนดถึงจะหยุดการทำงาน โครงสร้างรูปแบบการเขียนคำสั่งโปรแกรมแบบ Nested If แสดงดังนี้

โครงสร้างโปรแกรมควบคุมการทำงานแบบมีเงื่อนไข Nested If

```
if condition_1:  
    if condition_1_1:  
        statement_1_1  
    else:  
        statement_1_2  
elif condition_2:  
    if condition_2_1:  
        statement_2_1  
    else:  
        statement_2_2  
else:  
    statement_3
```

จากโครงสร้างโปรแกรมควบคุมทิศทางการทำงานแบบมีเงื่อนไข Nested If เมื่อนำมาเขียนเป็นผังงานแสดงขั้นตอนการทำงานจะได้ดังรูป 6.9 และในตัวอย่าง 6.12 แสดงการเขียนคำสั่งโปรแกรมการตัดสินใจซื้อสมาร์ทโฟนภายใต้เงื่อนไขของจำนวนเงินที่มีอยู่



รูป 6.9: ผังงานการทำงานแบบมีเงื่อนไข Nested If

## ตัวอย่าง 6.12

การเขียนคำสั่งโปรแกรมควบคุมทิศทางแบบมีเงื่อนไข Nested If ในการตัดสินใจซื้อสมาร์ทโฟน ภายใต้เงื่อนไขของจำนวนเงินที่มีอยู่

```
1 money = float(input('กรุณาป้อนจำนวนเงิน = ')) # กรอกจำนวนเงินที่มีอยู่
2 if money >= 27000:          # ตรวจสอบค่า money มากกว่า 27000 หรือไม่
3     if money >= 35000:      # ตรวจสอบค่า money มากกว่า 35000 หรือไม่
4         print('Buy iPhone 12 Pro Max') # แสดงผลถ้าบรรทัดที่ 3 เป็นจริง
5     else:
6         print('Buy iPhone 12 Pro')   # แสดงผลถ้าบรรทัดที่ 3 เป็นเท็จ
7 elif money >= 20000:          # ตรวจสอบค่า money มากกว่า 20000 หรือไม่
8     if money >= 25000:      # ตรวจสอบค่า money มากกว่า 25000 หรือไม่
9         print('Buy iPhone 12')    # แสดงผลถ้าบรรทัดที่ 8 เป็นจริง
10    else:
11        print('Buy iPhone SE')   # แสดงผลถ้าบรรทัดที่ 8 เป็นเท็จ
12 else:
13     print('Can not buy a new iPhone.') # แสดงผลเมื่อเงื่อนไขด้านบนทั้งหมดเป็นเท็จ
```

กรุณาป้อนจำนวนเงิน = 40000  
Buy iPhone 12 Pro Max



กรุณาป้อนจำนวนเงิน = 22000  
Buy iPhone SE

กรุณาป้อนจำนวนเงิน = 4000  
Can not buy a new iPhone

## 6.4 คำสั่งโปรแกรมควบคุมการทำงานแบบทำซ้ำ

การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบทำซ้ำ (Iteration Control Statement) ในภาษาไพธอน มีอยู่ด้วยกัน 2 คำสั่ง ได้แก่ คำสั่ง **while** และ คำสั่ง **for** สำหรับคำสั่ง **while** ต้องตรวจสอบเงื่อนไข ก่อนการทำงาน ถ้าเป็นเท็จจะไม่ทำงาน ถ้าเป็นจริงจะทำงานไปจนกว่าเงื่อนไขจะเป็นเท็จถึงจะหยุดการทำงาน กรณีคำสั่ง **for** ในภาษาไพธอนนิมามาใช้อ่านค่าข้อมูลหรือชนิดข้อมูลแบบลำดับ ซึ่งจะอ่านข้อมูลที่ถูกเก็บไว้ในตัวแปรจนกว่าจะหมดถึงหยุดการทำงาน

### 6.4.1 คำสั่งวนซ้ำด้วย **while**

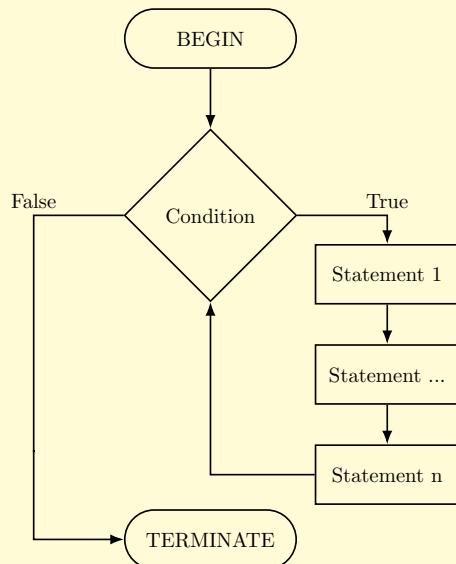
คำสั่ง **while** เป็นคำสั่งที่กำหนดให้โปรแกรมทำซ้ำตามเงื่อนไขในนิพจน์ ก่อนคำสั่งโปรแกรมที่อยู่หลัง คำสั่ง **while** จะทำงานได้จะมีการตรวจสอบเงื่อนไขก่อนทุกครั้ง เมื่อผลตรวจสอบเงื่อนไขเป็นจริงคำสั่ง หลัง **while** ถึงจะทำงาน ถ้าผลการตรวจสอบเงื่อนไขเป็นเท็จ โปรแกรมจะออกจาก การทำซ้ำและไปทำงานตามคำสั่งอื่น ๆ ตามที่ได้กำหนดไว้ คำสั่ง **while** เหมาะกับงานที่มีการทำซ้ำที่ไม่แน่นอน เมื่อเรา นำคำสั่ง **while** ไปใช้งานต้องปิดท้ายคำสั่งด้วยเครื่องหมาย Colon (:) เช่น เหมือนกับคำสั่ง **if** มี โครงสร้างรูปแบบการเขียนคำสั่ง **while** แสดงดังรูปต่อไปนี้

โครงสร้างการเขียนคำสั่งโปรแกรมควบคุมที่ศึกษาการการทำงานแบบทำซ้ำด้วย **while**

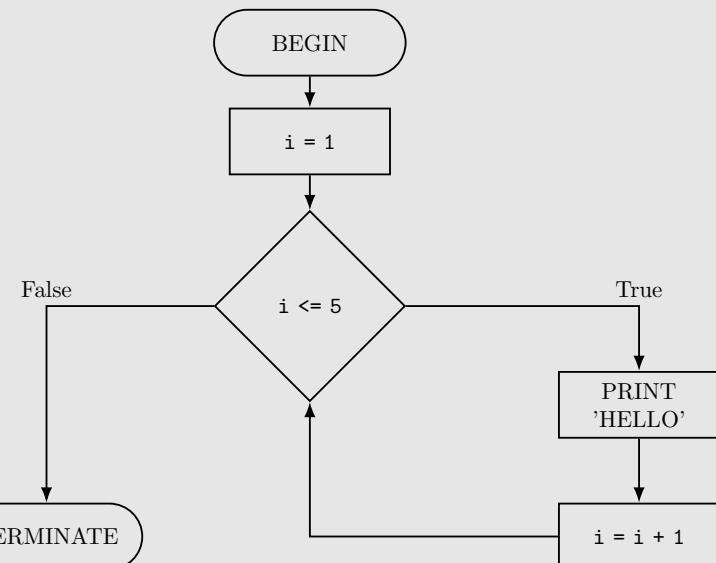
```
while condition:  
    statement w_1      # คำสั่งโปรแกรมให้ทำงานเมื่อเงื่อนไขเป็นจริง  
    statement ...       # คำสั่งโปรแกรมให้ทำงานเมื่อเงื่อนไขเป็นจริง  
    statement w_n       # คำสั่งโปรแกรมให้ทำงานเมื่อเงื่อนไขเป็นจริง
```

จากโครงสร้างโปรแกรมควบคุมการทำงานแบบทำซ้ำด้วย **while** เมื่อนำมาเขียนเป็นผังงานจะได้ ดังรูป 6.10

ตัวอย่างโปรแกรมต่อไปนี้ทำการควบคุมการทำงานแบบวนซ้ำด้วย **while** ให้แสดงผลคำว่า 'Hello' จำนวน 5 ครั้ง ออกทางหน้าจอ จะได้อย่างผังงานดังรูป 6.11



รูป 6.10: ผังงานของโปรแกรมควบคุมการทำงานแบบวนซ้ำด้วย **while**



รูป 6.11: ผังงานของตัวอย่างโปรแกรม 6.13

### ตัวอย่าง 6.13

การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบทำซ้ำด้วยคำสั่ง `while`

```
1 i = 1 # สร้างตัวแปรและกำหนดค่าเริ่มต้นในการตรวจสอบเงื่อนไข
2 while i <= 5: # ตรวจสอบค่าตัวแปร i น้อยกว่าหรือเท่ากับ 5 หรือไม่
3     print('Hello') # ถ้าเงื่อนไขเป็นจริง ให้แสดงผลลัพธ์
4     i = i + 1 # เพิ่มค่าตัวแปร i ด้วยการบวก 1
```

```
Hello
Hello
Hello
Hello
Hello
```



เริ่มต้นตัวแปร `i` ถูกกำหนดค่าเป็น `1` ในแต่ละรอบโปรแกรมจะตรวจสอบค่าตัวแปร `i` ว่าน้อยกว่า หรือเท่ากับ `5` หรือไม่ ถ้ามีค่าน้อยกว่าหรือเท่ากับ `5` จะแสดงผลคำว่า '`Hello`' ออกทางหน้าจอ และ เพิ่มค่าตัวแปร `i` ด้วยการนำตัวแปร `i` บวก `1` โปรแกรมจะทำซ้ำจนกว่าตัวแปร `i` มีค่ามากกว่า `5` ถึงจะ หยุดการทำงาน

### ตัวอย่าง 6.14

การเขียนคำสั่งโปรแกรมสำหรับคำนวณอนุกรม

$$1 + 2 + 3 + \dots + 100000$$

ด้วยคำสั่ง `while`

```
1 mysum = 0
2 k = 1
3 N = 100000
4 while k < N:
5     mysum = mysum + k
6     k = k + 1
7 print('Summation is', mysum)
```

```
Summation is 4999950000
```



### ตัวอย่าง 6.15

การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบทำซ้ำด้วยคำสั่ง **while** และแสดงตัวเลขที่ผู้ใช้งานป้อนจำนวน 5 ครั้ง และแสดงค่าตัวเลขน้อยที่สุด ค่าตัวเลขมากที่สุด และผลรวมที่ป้อนทั้งหมด

```
1 i = 1 # สร้างตัวแปรและกำหนดค่าเริ่มต้นในการตรวจสอบเงื่อนไข
2 num = [] # สร้างตัวแปรเป็นชนิดข้อมูลลิสต์
3 while i <= 5: # ตรวจสอบค่าตัวแปร i น้อยกว่าหรือเท่ากับ 5 หรือไม่
4     n = int(input(f'ป้อนตัวเลขจำนวนเต็ม ครั้งที่ {i} = ')) # ถ้าเป็นจริง จะให้ป้อน
        → จำนวนเต็ม
5     num.append(n) # เพิ่มค่าตัวแปร n เก็บไว้ในลิสต์ num
6     i = i + 1 # เพิ่มค่าตัวแปร i ด้วยการบวก 1
7 print(f'ตัวเลขที่คุณป้อน = {num}') # แสดงผลจากค่าลิสต์ num
8 print(f'ตัวเลขค่าที่น้อยที่สุด = {min(num)}') # แสดงผลค่าที่น้อยที่สุดจากลิสต์ num
9 print(f'ตัวเลขค่าที่มากที่สุด = {max(num)}') # แสดงผลค่าที่มากที่สุดจากลิสต์ num
10 print(f'ผลรวมตัวเลขทั้งหมด = {sum(num)}') # แสดงผลผลรวมจากลิสต์ num
```

```
ป้อนตัวเลขจำนวนเต็ม ครั้งที่ 1 = 100
ป้อนตัวเลขจำนวนเต็ม ครั้งที่ 2 = 50
ป้อนตัวเลขจำนวนเต็ม ครั้งที่ 3 = -50
ป้อนตัวเลขจำนวนเต็ม ครั้งที่ 4 = 0
ป้อนตัวเลขจำนวนเต็ม ครั้งที่ 5 = 99
ตัวเลขที่คุณป้อน = [100, 50, -50, 0, 99]
ตัวเลขค่าที่น้อยที่สุด = -50
ตัวเลขค่าที่มากที่สุด = 100
ผลรวมตัวเลขทั้งหมด = 199
```



เมื่อโปรแกรมทำงานรอบที่ 1 เสร็จ ค่าตัวแปร i ถูกเพิ่มค่าเป็น 2 จากนั้นมีเริ่มทำงานในรอบที่ 2 ค่าตัวแปร i จะด้วยคำสั่ง **while** อีกครั้ง โปรแกรมจะทำซ้ำจนกว่าค่าตัวแปร i มีค่ามากกว่า 5 ถึงจะหยุดให้ผู้ใช้งานป้อนตัวเลข

ในบางครั้งเมื่อนำคำสั่ง **while** มาใช้งานจะเกิดเหตุการณ์หนึ่งเกิดขึ้นเรียกว่า การทำซ้ำไม่รู้จบ (Infinite Loop) ซึ่งเกิดขึ้นจากการกำหนดเงื่อนไขที่ผิดพลาดของผู้เขียนโปรแกรมเอง ทำให้โปรแกรมตรวจสอบไม่พบเงื่อนไขที่เป็นเท็จ ดังนั้นผู้เขียนโปรแกรมควรตรวจสอบการกำหนดเงื่อนไขให้ถูกต้องก่อนสังให้โปรแกรมทำงานเมื่อกำหนดเงื่อนไขไม่ถูกต้องจะทำให้เกิดการทำซ้ำไม่รู้จบ ดังตัวอย่างต่อไปนี้

### ตัวอย่าง 6.16

โปรแกรมที่มีการทำซ้ำไม่รู้จบ

```
1 i = 1
2 num = []
3 while i > 0:
4     n = float(input('กรุณาป้อนตัวเลข : '))
5     num.append(n)
6     i = i + 1
7 print(f'จำนวนตัวเลขที่คุณป้อน : {num}')
8
```

```
กรุณาป้อนตัวเลข : 55
กรุณาป้อนตัวเลข : 34
กรุณาป้อนตัวเลข : 44.5
กรุณาป้อนตัวเลข : 90
กรุณาป้อนตัวเลข : 12
...
กรุณาป้อนตัวเลข :
```



การทำซ้ำของคำสั่ง `while` แบบไม่รู้จบ (Infinite loop) จากตัวอย่างโปรแกรมคำสั่งในบรรทัดที่ 1 กำหนดค่าตัวแปร `i` เท่ากับ **1** และเมื่อนำค่าตัวแปร `i` ไปตรวจสอบเงื่อนไขด้วยคำสั่ง `while i > 0` ในบรรทัดที่ 3 จะเห็นว่าเงื่อนไขนี้เป็นจริงเสมอ เมื่อทำงานรอบแรกเสร็จตัวแปร `i` ถูกเพิ่มค่าอีก **1** ตามคำสั่งในบรรทัดที่ 6 เมื่อนำตัวแปร `i` ไปตรวจสอบเงื่อนไขอีกครั้งในบรรทัดที่ 3 เงื่อนไขนี้ก็เป็นจริงอีก และจะเป็นจริงอย่างนี้ตลอดไป ทำให้ผู้ใช้งานต้องป้อนจำนวนตัวเลขไปอย่างไม่มีที่สิ้นสุด ผู้ใช้ต้องป้อนคำสั่ง `KeyboardInterrupt` โดยการกดคีย์ `ctrl + C` หรือ `ctrl + D` หากต้องการให้โปรแกรมหยุดการทำงาน

#### 6.4.2 การใช้คำสั่ง `else` ร่วมกับคำสั่งวนซ้ำ `while`

ในภาษาไพธอนได้นำคำสั่ง `else` เข้ามาร่วมทำงานกับคำสั่งวนซ้ำ `while` ด้วย โดยคำสั่ง `else` จะได้รับการประมวลผลเมื่อมีการตรวจสอบเงื่อนไขคำสั่ง `while` เป็นเท็จ และไม่มีการนำเข้าคำสั่ง `break` เข้ามาใช้ ถ้านำเข้าคำสั่ง `break` มาใช้งานจะทำให้คำสั่ง `else` ไม่ถูกประมวลผล สำหรับการใช้คำสั่ง `break` ผู้เขียนจะได้อธิบายการใช้ในหัวข้อถัดไป การใช้คำสั่ง `else` ร่วมกับคำสั่ง `while` แสดงดังตัวอย่างต่อไปนี้

### ตัวอย่าง 6.17

การใช้คำสั่ง `else` ร่วมทำงานกับคำสั่ง `while`

```
1 i = 1
2 while i <= 10:
3     print(i)
4     i = i + 2
5 else:
6     print('Done !!') # แสดงผลเมื่อบรรทัดที่ 2 เป็นเท็จ คือ i มากกว่าหรือเท่า 10
```

```
1
3
5
7
9
Done !!
```



### 6.4.3 คำสั่งวนซ้ำด้วย `for`

คำสั่ง `for` เป็นคำสั่งควบคุมการทำงานแบบวนซ้ำด้วยจำนวนรอบที่แน่นอน ถ้าผลการตรวจสอบเงื่อนไข เป็นจริงจะแสดงผลคำสั่งโปรแกรมที่อยู่หลัง `for` หากเงื่อนไขเป็นเท็จจะออกจาก การวนซ้ำ และทำงานในคำสั่งอื่นถัดไป คำสั่ง `for` ในภาษาไพธอนถูกนำมาใช้สำหรับอ่านค่าข้อมูลที่เก็บแบบเรียงลำดับหรือชนิดข้อมูลแบบเรียงลำดับ เช่น ลิสต์ ทูเพิล เชต อักขระหรือสตริง เป็นต้น หรือนำมาใช้อ่านค่าข้อมูลที่ถูกสร้างขึ้นมาได้ด้วยฟังก์ชัน `range()` มีรูปแบบดังต่อไปนี้

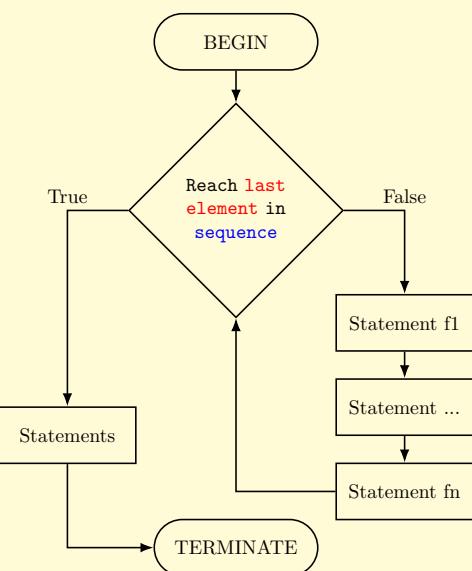
```
for seq_var in sequence:
    statement
```

`seq_var` ตัวแปรที่ค่อยรับค่าจาก `sequence`

`sequence` ข้อมูลหรือชนิดข้อมูลแบบเรียงลำดับ

`statement` คำสั่งโปรแกรมที่ต้องการให้ทำงาน อาจจะมีมากกว่า 1 คำสั่ง

จากโครงสร้างโปรแกรมควบคุมการทำงานแบบวนซ้ำ `for` นำมาเขียนเป็นผังงานได้ดังรูป 6.12 ซึ่งจะเขียนเป็นโปรแกรมควบคุมการทำงานแบบวนซ้ำด้วย `for` กับชนิดข้อมูลต่าง ๆ ทั้งลิสต์ สตริง เชต และผลลัพธ์จากฟังก์ชัน `range()` ได้ดังตัวอย่างต่อไปนี้



รูป 6.12: ผังงานของโปรแกรมควบคุมการทำงานแบบวนซ้ำด้วย **for**

### ตัวอย่าง 6.18

การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบวนซ้ำ for กับข้อมูลแบบลำดับ

```

1 # ทำซ้ำนิดข้อมูลลิสต์
2 my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
3 for num in my_list:
4     print(num, end = ', ')
5 print('\n' + '-' * 20)
6 # ทำซ้ำนิดข้อมูลสตริง
7 my_string = 'Python Programming'
8 for char in my_string:
9     print(char, end = ', ')
10 print('\n' + '-' * 20)
11 # ทำซ้ำนิดข้อมูลเซต
12 my_set = {2, 5, 7, 5, 8, 7, 9}
13 for m in my_set:
14     print(m, end = ', ')
15 print('\n' + '-' * 20)
16 # ทำซ้ำจากฟังก์ชัน range()
17 for n in range(1, 10):
18     print(n, end = ', ')
19 print('\n' + '-' * 20)

```

```

1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
-----
P, y, t, h, o, n, , P, r, o, g, r, a, m, m, i, n, g,
-----
2, 5, 7, 8, 9,
-----
1, 2, 3, 4, 5, 6, 7, 8, 9,
-----
```



คำสั่ง **for** อ่านค่าตัวแปรที่ลະค่าในแต่ละรอบ จากนั้นจึงจะนำค่าตัวแปรมาแสดงผล ซึ่งจะทำช้าอย่างนี้ไปเรื่อย ๆ จนกว่าอ่านค่าจากตัวแปรครบทุกค่าในลิสต์ โปรแกรมถึงจะหยุดการทำงาน สำหรับฟังก์ชัน **range(x, y)** จะแจกแจงค่าจำนวนเต็มในช่วงที่กำหนดตั้งแต่ **x** จนถึง **y-1** อกมาเป็นตัว ๆ จนครบ

### ตัวอย่าง 6.19

การเขียนโปรแกรมแม่สูตรคูณด้วยคำสั่งทำช้า **for** และฟังก์ชัน **range()** สร้างช่วงตัวเลข 1-12

```
1 i = int(input('Enter an integer: '))
2 print(f'Multiplication Table of {i}')
3 for j in range(1, 13):
4     # สร้างช่วงตัวเลขด้วยฟังก์ชัน range() จากค่าเริ่มต้น โดยค่าสุดท้ายจะลบด้วย 1
5     num = i * j
6     # ค่าตัวแปร j เริ่มที่ 1 และเพิ่มครั้ง 1 จนกว่าจะถึง 12
7     print(f'{i} * {j} = {num}')
8     # แสดงผลค่าตัวแปร i, j และ num ในแต่ละรอบ
```

```
Enter an integer: 9
Multiplication Table of 9
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
9 * 10 = 90
9 * 11 = 99
9 * 12 = 108
```



นอกจากนี้เรายังนำคำสั่งการทำช้ามาช่วยในการจัดการกับชนิดข้อมูลได้ ซึ่งในตัวอย่างต่อไปเป็นการแปลงชนิดข้อมูลดิจิทัลนารีเป็นชนิดข้อมูลลิสต์

### ตัวอย่าง 6.20

การเขียนคำสั่งโปรแกรมแปลงชนิดข้อมูลดิจิทัลนารีเป็นชนิดข้อมูลลิสต์

```
1  dct = {1:'A', 2:'E', 3:'I', 4:'O', 5:'U'}
2  dct_lst = []
3  for i, j in dct.items():
4      dct_lst.append((i, j))
5  print(dct_lst)
```

```
[(1, 'A'), (2, 'E'), (3, 'I'), (4, 'O'), (5, 'U')]
```



### 6.4.4 การใช้คำสั่ง `else` ร่วมกับคำสั่งทำซ้ำ `for`

คำสั่ง `else` นอกจากจะนำมาใช้งานร่วมกับคำสั่ง `while` แล้ว ยังนำมาใช้งานร่วมกับคำสั่ง `for` ได้อีกด้วย โดยคำสั่ง `else` จะทำงานก็ต่อเมื่อกระบวนการทำงานของคำสั่ง `for` อ่านค่าข้อมูลแบบเรียงลำดับครบถ้วนตำแหน่ง และไม่มีการนำคำสั่ง `break` เข้ามาใช้งานภายในคำสั่ง `for` (สำหรับคำสั่ง `break` จะได้อธิบายการใช้งานในหัวข้อถัดไป)

### ตัวอย่าง 6.21

การใช้คำสั่ง `else` ร่วมกับคำสั่งทำซ้ำ `for`

```
1  sports = ['Running', 'Swimming', 'Tennis', 'Racing', 'Football']
2  i = 0
3  for s in range(len(sports)):
4      print(f'ชื่อชนิดกีฬาตำแหน่งที่ {i} คือ {sports[s]}')
5      i = i + 1
6  else:
7      print('Done !!') # แสดงผลหลังคำสั่ง for แสดงผลตัวแปร sports ครบถ้วนตำแหน่ง
```

```
ชื่อชนิดกีฬาตำแหน่งที่ 0 คือ Running
ชื่อชนิดกีฬาตำแหน่งที่ 1 คือ Swimming
ชื่อชนิดกีฬาตำแหน่งที่ 2 คือ Tennis
ชื่อชนิดกีฬาตำแหน่งที่ 3 คือ Racing
ชื่อชนิดกีฬาตำแหน่งที่ 4 คือ Football
Done !!
```



## 6.5 การใช้คำสั่งทำซ้ำ `while` หรือ `for` ซ้อนกัน

คำสั่ง `while` หรือ `for` สามารถถูกวางซ้อนกันได้ เมื่อฉันกับการใช้คำสั่งแบบมีเงื่อนไข `if` ซ้อน `if` เพื่อช่วยแก้ไขปัญหางานที่มีความซับซ้อนของการทำซ้ำ โดยทำได้ทั้ง `while` ซ้อน `while`, `for` ซ้อน `for` หรือ `for` ซ้อน `while` ก็ได้ ดังตัวอย่างต่อไปนี้

### ตัวอย่าง 6.22

การเขียนโปรแกรมแม่สูตรคูณด้วยคำสั่งทำซ้ำ `while` ซ้อน `while` สร้างช่วงตัวเลข 1-12

```
1 i = 1
2 while i <= 5: # คำสั่ง while ด้านนอกตรวจสอบ i น้อยกว่าหรือเท่ากับ 5 หรือไม่
3     print(f'Round {i}: ', end = '') # แสดงผลลัพธ์ i น้อยกว่าหรือเท่ากับ 5
4     j = 1
5     while j <= 12: # คำสั่ง while ด้านในตรวจสอบ j น้อยกว่าหรือเท่ากับ 12 หรือไม่
6         print(i * j, end = ' ') # แสดงผลลัพธ์ j น้อยกว่าหรือเท่ากับ 12
7         j += 1 # เพิ่มค่า j อีก 1
8     print(' ') # จัดบรรทัดใหม่
9     i = i + 1 # เพิ่มค่า i อีก 1
10 print('Done')
```

```
Round 1: 1 2 3 4 5 6 7 8 9 10 11 12
Round 2: 2 4 6 8 10 12 14 16 18 20 22 24
Round 3: 3 6 9 12 15 18 21 24 27 30 33 36
Round 4: 4 8 12 16 20 24 28 32 36 40 44 48
Round 5: 5 10 15 20 25 30 35 40 45 50 55 60
Done
```



ผลลัพธ์จากการทำงานในแต่ละรอบซึ่งมีทั้งหมด 5 รอบ ค่าตัวเลขจะเพิ่มตามจำนวนรอบ เช่น รอบที่ 1 ตัวเลขเพิ่มขึ้นครั้งละ 1, รอบที่ 2 เพิ่มขึ้นครั้งละ 2 สุดท้ายเมื่อถึงรอบที่ 5 ตัวเลขจะเพิ่มขึ้นครั้งละ 5

### ตัวอย่าง 6.23

การเขียนคำสั่งโปรแกรมแม่สูตรคูณโดยใช้คำสั่ง **for** ซ้อน **for**

```
1 for i in range(1, 13):
2     for j in range(2, 7):
3         print(f'{j}*{i}={j * i}', end='\t')
4     print()
5 print('Done')
```

```
2*1=2    3*1=3    4*1=4    5*1=5    6*1=6
2*2=4    3*2=6    4*2=8    5*2=10   6*2=12
2*3=6    3*3=9    4*3=12   5*3=15   6*3=18
2*4=8    3*4=12   4*4=16   5*4=20   6*4=24
2*5=10   3*5=15   4*5=20   5*5=25   6*5=30
2*6=12   3*6=18   4*6=24   5*6=30   6*6=36
2*7=14   3*7=21   4*7=28   5*7=35   6*7=42
2*8=16   3*8=24   4*8=32   5*8=40   6*8=48
2*9=18   3*9=27   4*9=36   5*9=45   6*9=54
2*10=20  3*10=30  4*10=40  5*10=50  6*10=60
2*11=22  3*11=33  4*11=44  5*11=55  6*11=66
2*12=24  3*12=36  4*12=48  5*12=60  6*12=72
Done
```



จากการทำงานของโปรแกรมคำสั่ง **for** ซ้อน **for** เราจะพบว่าคำสั่ง **for** ด้านนอก จะทำงานในรอบถัดไปได้ก็ต่อเมื่อคำสั่ง **for** ด้านในอ่านค่าข้อมูลและแสดงผลครบทั้งหมดก่อน จานวนคำสั่ง **for** ด้านนอกค่อยกลับมาอ่านค่าข้อมูลจนกว่าจะหมด โปรแกรมถึงจะหยุดการทำงาน

## ตัวอย่าง 6.24

การเขียนคำสั่งโปรแกรมคำสั่งวนซ้ำ **for** ซ้อน **while** หากคะแนนรวมของนักเรียนแต่ละคน

```
1 # ป้อนจำนวนนักเรียน
2 stu = int(input('ป้อนจำนวนนักเรียน = '))
3 # ทำซ้ำตามจำนวนค่าตัวแปร stu
4 for i in range(1, stu + 1):
5     j = 1 # กำหนดค่าเริ่มต้นเพื่อตรวจสอบคำสั่ง while
6     lst = [] # ตัวแปรลิสต์สำหรับเก็บคะแนน
7     print(f'นักเรียนคนที่ {i}') # จะแสดงผลตามรอบที่ i
8     while j <= 3: # เงื่อนไขทำซ้ำ เมื่อ j น้อยกว่าหรือเท่ากับ 3
9         # ถ้ามากกว่าจะออกจากการทำซ้ำ
10        n = float(input(f'กรอกคะแนนครั้งที่ {j}: '))
11        lst.append(n) # นำค่าตัวแปรเก็บไว้ที่ตัวแปร lst
12        j += 1 # เพิ่มค่าตัวแปร j ด้วยการบวก 1
13    # เส่งผลคะแนนทั้งหมดและคะแนนรวม
14    print(f'นักเรียนคนที่ {i} ได้คะแนน = {lst} \t คะแนนรวม = {sum(lst)}')
```

```
ป้อนจำนวนนักเรียน = 3
▶
นักเรียนคนที่ 1
กรอกคะแนนครั้งที่ 1:  30
กรอกคะแนนครั้งที่ 2:  23
กรอกคะแนนครั้งที่ 3:  26.5
นักเรียนคนที่ 1 ได้คะแนน = [30.0, 23.0, 26.5]      คะแนนรวม = 79.5
นักเรียนคนที่ 2
กรอกคะแนนครั้งที่ 1:  12
กรอกคะแนนครั้งที่ 2:  9.5
กรอกคะแนนครั้งที่ 3:  24
นักเรียนคนที่ 2 ได้คะแนน = [12.0, 9.5, 24.0]      คะแนนรวม = 45.5
นักเรียนคนที่ 3
กรอกคะแนนครั้งที่ 1:  33
กรอกคะแนนครั้งที่ 2:  25
กรอกคะแนนครั้งที่ 3:  19.5
นักเรียนคนที่ 3 ได้คะแนน = [33.0, 25.0, 19.5]      คะแนนรวม = 77.5
```

จากตัวอย่างที่ผ่านมาผู้เราได้เรียนรู้ถึงการนำคำสั่งการทำซ้ำมาซ้อนกัน นอกจากนี้เรายังสามารถนำเอาราคำสั่งแบบมีเงื่อนไข **if** มาซ้อนในคำสั่งการทำซ้ำ **for** และ **while** ได้อีกด้วย เพื่อใช้เป็นเงื่อนไขในการตัดสินใจการทำงานของโปรแกรม ดังตัวอย่างต่อไปนี้

### ตัวอย่าง 6.25

การใช้คำสั่งแบบมีเงื่อนไข **if** ซ้อนคำสั่ง **for** และคำสั่ง **while** ตัดเกรด

```
1 stu = int(input('ป้อนจำนวนนักเรียน = '))
2 for i in range(1, stu + 1):
3     j = 1
4     lst = []
5     print('นักเรียนคนที่ ', i)
6     while j <= 3:
7         n = float(input(f'กรอกคะแนนครั้งที่ {j} = '))
8         lst.append(n)
9         j += 1
10    if sum(lst) <= 49:
11        grade = 'F'
12    elif sum(lst) <= 59:
13        grade = 'D'
14    elif sum(lst) <= 69:
15        grade = 'C'
16    elif sum(lst) <= 79:
17        grade = 'B'
18    else:
19        grade = 'A'
20    print(f'นักเรียนคนที่ {i} ได้คะแนน = {lst} คะแนนรวม = {sum(lst)} ได้เกรด
→ {grade}' )
```

ป้อนจำนวนนักเรียน = 2  
นักเรียนคนที่ 1  
กรอกคะแนนครั้งที่ 1 = 23  
กรอกคะแนนครั้งที่ 2 = 12  
กรอกคะแนนครั้งที่ 3 = 23  
นักเรียนคนที่ 1 ได้คะแนน = [23.0, 12.0, 23.0] คะแนนรวม = 58.0 ได้เกรด D  
นักเรียนคนที่ 2  
กรอกคะแนนครั้งที่ 1 = 33  
กรอกคะแนนครั้งที่ 2 = 23  
กรอกคะแนนครั้งที่ 3 = 25  
นักเรียนคนที่ 2 ได้คะแนน = [33.0, 23.0, 25.0] คะแนนรวม = 81.0 ได้เกรด A



จากตัวอย่างโปรแกรมการพัฒนาต่อจากโปรแกรมการใช้คำสั่ง **for** ซ้อน **while** ในบรรทัดที่ 10-19 ใช้คำสั่ง **if** เข้ามาช่วยตัดสินใจการตัดเกรด A-F หลังจากผู้ใช้งานป้อนคะแนนของนักเรียนแต่ละคนครบ

จำนวน 3 ครั้ง จากการตรวจสอบการทำซ้ำด้วยคำสั่ง **while** ในบรรทัดที่ 6 และโปรแกรมทำงานตามจำนวนรอบที่กำหนดไว้ในคำสั่ง **for** ที่อ่านค่าข้อมูลจากที่สร้างด้วยฟังก์ชัน **range()** ในบรรทัดที่ 2

## 6.6 การควบคุมการทำซ้ำด้วยคำสั่ง **break**, **continue** และ **pass**

คำสั่ง **break** นำมาใช้ในกรณีที่ต้องการให้ออกจากการทำซ้ำก่อนถึงรอบที่กำหนด หรือใช้ในกรณีที่ต้องการตรวจสอบความถูกต้อง ซึ่งมีผลทำให้คำสั่งที่เหลือหลัง **break** ในขอบเขตคำสั่งทำซ้ำเดียวกันไม่ถูกประมวลผล แต่จะข้ามไปทำงานในคำสั่งที่เหลือของโปรแกรมเลยทันที ดังตัวอย่างต่อไปนี้

### ตัวอย่าง 6.26

การใช้คำสั่ง **break** หยุดการทำงานทันที เมื่อป้อนข้อมูลไม่ถูกต้อง

```
1 for i in range(1, 6):
2     n = int(input('กรุณาป้อนตัวเลข 1-10 : '))
3     if n >= 11:
4         print('คุณป้อนตัวเลขไม่ถูกต้อง !!!')
5         break
6     else:
7         j = n * i
8         print(f'ผลคูณระหว่าง {i} * {n} = {j}')
9 print('สิ้นสุดการทำงานของโปรแกรม')
```

กรุณาป้อนตัวเลข 1-10 : 3  
ผลคูณระหว่าง 1 \* 3 = 3  
กรุณาป้อนตัวเลข 1-10 : 5  
ผลคูณระหว่าง 2 \* 5 = 10  
กรุณาป้อนตัวเลข 1-10 : 11  
คุณป้อนตัวเลขไม่ถูกต้อง !!  
สิ้นสุดการทำงานของโปรแกรม



โปรแกรมจะแสดงผลครบถ้วนของการทำงานเมื่อผู้ใช้ป้อนตัวเลขได้ถูกต้องตามเงื่อนไขในบรรทัดที่ 6 คือป้อนตัวเลขไม่เกิน 10 ถ้าในระหว่างการทำงานเมื่อผู้ใช้งานป้อนตัวเลขไม่ถูกต้องคือ ป้อนตัวเลขมากกว่า 10 โปรแกรมจะออกจากการทำงานทันทีด้วยคำสั่ง **break** ในบรรทัดที่ 5

### ตัวอย่าง 6.27

การใช้คำสั่ง break ร่วมกับคำสั่ง while ออกจากทำซ้ำ

```
1 while True:
2     i = input('ต้องการป้อนคะแนนให้กด Enter, ออกจากโปรแกรมให้กด N: ')
3     print ('-'*30)
4     if i == 'N':
5         break
6     else:
7         name = input('ชื่อนักเรียน: ')
8         j = 1
9         lst = []
10        while j <= 3:
11            score = float(input(f'กรอกคะแนนครั้งที่ {j} = '))
12            lst.append(score)
13            j += 1
14        print(f'นักเรียน: {name} ได้คะแนน = {lst} คะแนนรวม {sum(lst)}')
15    print ('สิ้นสุดการทำงานของโปรแกรม')
```

ต้องการป้อนคะแนนให้กด Enter, ออกจากโปรแกรมให้กด N:



-----  
ชื่อนักเรียน: Sally

กรอกคะแนนครั้งที่ 1 = 33

กรอกคะแนนครั้งที่ 2 = 23

กรอกคะแนนครั้งที่ 3 = 25

นักเรียน: Sally ได้คะแนน = [33.0, 23.0, 25.0] คะแนนรวม 81.0

ต้องการป้อนคะแนนให้กด Enter, ออกจากโปรแกรมให้กด N:

-----  
ชื่อนักเรียน: John

กรอกคะแนนครั้งที่ 1 = 23

กรอกคะแนนครั้งที่ 2 = 22

กรอกคะแนนครั้งที่ 3 = 12

นักเรียน: John ได้คะแนน = [23.0, 22.0, 12.0] คะแนนรวม 57.0

ต้องการป้อนคะแนนให้กด Enter, ออกจากโปรแกรมให้กด N:

-----  
สิ้นสุดการทำงานของโปรแกรม

คำสั่ง **continue** นำมาใช้หยุดการทำงานของคำสั่งโปรแกรมที่เหลือในขอบเขตคำสั่งทำซ้ำเดียวกัน แล้วให้กลับไปทำงานในรอบทำซ้ำถัดไป หลังจากทำซ้ำเสร็จแล้วจึงออกไปประมวลคำสั่งโปรแกรมอีก ๆ ดังตัวอย่างต่อไปนี้

## ตัวอย่าง 6.28

การใช้คำสั่ง `continue` ร่วมกับคำสั่ง `while`

```
1 i = 1
2 while i <= 5:
3     n = int(input('กรุณาป้อนตัวเลข 1-10 : '))
4     if n > 10:
5         print('คุณป้อนตัวเลขไม่ถูกต้อง !!!')
6         continue
7     else:
8         j = n * i
9     print(f'ผลคูณระหว่าง {i} * {n} = {j}')
10    i += 1
```

```
กรุณาป้อนตัวเลข 1-10 : 4
ผลคูณระหว่าง 1 * 4 = 4
กรุณาป้อนตัวเลข 1-10 : 5
ผลคูณระหว่าง 2 * 5 = 10
กรุณาป้อนตัวเลข 1-10 : 6
ผลคูณระหว่าง 3 * 6 = 18
กรุณาป้อนตัวเลข 1-10 : 112
คุณป้อนตัวเลขไม่ถูกต้อง !!
กรุณาป้อนตัวเลข 1-10 : 3
ผลคูณระหว่าง 4 * 3 = 12
กรุณาป้อนตัวเลข 1-10 : 8
ผลคูณระหว่าง 5 * 8 = 40
```



จากตัวอย่างโปรแกรม เมื่อผู้ใช้งานป้อนตัวเลขได้ถูกต้องตามเงื่อนไขคือ **1-10** โปรแกรมถึงจะแสดงผลลัพธ์ แต่เมื่อผู้ใช้งานป้อนตัวเลขมากกว่า **10** ส่งผลให้คำสั่ง `continue` ทำงาน แล้วกลับไปทำงานในรอบใหม่และคำสั่งที่เหลือในขอบเขตของคำสั่งทำขึ้นไม่ถูกประมวลผล แต่โปรแกรมจะทำงานจนครบรอบตามเงื่อนไขที่กำหนดคือ **5** รอบ ตามเงื่อนไขที่ได้กำหนดไว้ในคำสั่ง `while`

คำสั่ง `pass` นำมายังสำหรับการลงทะเบียนการประมวลผลคำสั่งโปรแกรม ในกรณีที่เรายังไม่ทราบว่าจะเขียนคำสั่งโปรแกรมให้ทำงานอะไรซึ่งช่วยให้โปรแกรมประมวลผลคำสั่งได้ตามปกติ หลังจากที่ได้คำสั่งการทำงานที่ต้องการแล้วจึงนำมาแทนที่คำสั่ง `pass` ในภายหลัง ซึ่งมีประโยชน์ในการทดสอบโปรแกรม ดังตัวอย่างต่อไปนี้

## ตัวอย่าง 6.29

การเขียนคำสั่งโปรแกรมการใช้คำสั่ง **pass** ทดสอบการทำงานของโปรแกรม

```
1 salary = int(input('ป้อนเงินเดือนพนักงาน = '))
2 if salary <= 30000:
3     pass
4 elif salary <= 35000:
5     pass
6 elif salary <= 40000:
7     bonus = salary * 3
8     print('เงินโบนัส = ', bonus, 'บาท')
9 else:
10    bonus = salary * 2
11    print('เงินโบนัส = ', bonus, 'บาท')
12 print('สิ้นสุดการทำงานของโปรแกรม')
```

ป้อนเงินเดือนพนักงาน = 25000

สิ้นสุดการทำงานของโปรแกรม

ป้อนเงินเดือนพนักงาน = 45000

เงินโบนัส = 90000 บาท

สิ้นสุดการทำงานของโปรแกรม



จากผลลัพธ์รูปบนเป็นการทำงานของคำสั่ง **pass** ในบรรทัดที่ 3 โดยที่ผู้ใช้งานป้อนเงินเดือนเท่ากับ **25000** แต่เรายังไม่ได้ตัดสินใจกำหนดค่าโบนัสให้กับพนักงาน จึงใช้คำสั่ง **pass** ให้ทำงานแทนไปก่อน ทำให้บรรทัดที่ 12 แสดงผลเท่านั้น ส่วนรูปถัดมีผู้ใช้งานป้อนเงินเดือนเท่ากับ **45000** ทำให้เข้าเงื่อนไข บรรทัดที่ 6 และคำนวนค่าโบนัสให้กับพนักงานในบรรทัดที่ 7 แล้วแสดงผลลัพธ์ค่าโบนัสที่ได้ในบรรทัดที่ 8 สำหรับบรรทัดที่ 12 จะแสดงผลทุกครั้งหลังโปรแกรมทำงานเสร็จ

## สรุปก่อนจบบท

การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบ่งออกเป็น 3 รูปแบบ ได้แก่ คำสั่งการทำงานแบบ ลำดับ คำสั่งการทำงานแบบมีเงื่อนไข **if, if ... else** และ **if ... elif** และคำสั่งการทำงานแบบทำซ้ำ **while** และ **for** นอกจากนี้เรายังได้เรียนรู้การนำคำสั่ง **if** และคำสั่ง **while** หรือ **for** มาเขียนเป็นคำสั่งโปรแกรมช้อนกัน เพื่อนำไปประยุกต์ใช้ในงานที่มีความซับซ้อนมากขึ้น ใน การเขียนคำสั่ง **if, while** และ **for** ด้วยภาษาไพธอน ผู้เขียนโปรแกรมต้องไม่ลืมใส่เครื่องหมาย **colon(:)** ท้ายประโยคเสมอ และเมื่อขึ้นบรรทัดใหม่ในขอบเขตคำสั่งทำซ้ำเดียวกัน ควรใช้กดปุ่ม Tab แทนการกดปุ่ม Space bar

## แบบฝึกหัด

1. จงเขียนโปรแกรมเบรี่ยบเทียบตัวเลขที่รับเข้ามาผ่านทางคีย์บอร์ดจำนวน 2 ตัว ถ้าตัวเลขที่ 1 มีค่ามากกว่าตัวเลขที่ 2 ให้แสดงคำว่า 'ตัวเลขที่ 1 มีค่ามากกว่าตัวเลขที่ 2' ถ้าตัวเลขที่ 1 มีค่าน้อยกว่าตัวเลขที่ 2 ให้แสดงคำว่า 'ตัวเลขที่ 1 มีค่าน้อยกว่าตัวเลขที่ 2'
2. จงเขียนโปรแกรมคำนวนการซื้อสินค้าทั้งหมด 3 อย่าง โดยรับราคาสินค้าผ่านทางคีย์บอร์ด พร้อมทั้งแสดงราคาสินค้าแต่ละชิ้น สรุปรวมของสินค้าที่ซื้อ สุดท้ายให้แสดงผลคำว่า 'Good Bye !!'
3. จงเขียนโปรแกรมคำนวนค่าวงวดรายนั้นๆ กำหนดให้ป้อนรายรับรายจ่ายและจำนวนเดือนที่ต้องการ ผ่อนผ่านทางคีย์บอร์ด โดยมีเงื่อนไขดังนี้
  - ถ้าต้องการผ่อน 12 เดือน คิดดอกเบี้ย 1% ต่อเดือน
  - ถ้าต้องการผ่อน 24 เดือน คิดดอกเบี้ย 1.5% ต่อเดือน
  - ถ้าต้องการผ่อน 36 เดือน คิดดอกเบี้ย 2.5% ต่อเดือน
  - ถ้าต้องการผ่อน 48 เดือน คิดดอกเบี้ย 3.5% ต่อเดือน
  - ถ้าต้องการผ่อน 60 เดือน คิดดอกเบี้ย 4.5% ต่อเดือน
4. จงเขียนโปรแกรมเลือกเครือข่ายเติมเงินโทรศัพท์มือถือ โดยมีเงื่อนไขดังต่อไปนี้
  - ก่อนป้อนจำนวนเงินให้แสดงเมนูเลือกเครือข่ายเสมอ
  - มีเครือข่ายให้เลือกเติมเงิน 3 เครือข่ายได้แก่ DTAC, AIS และ True
  - มีเมนูแสดงจำนวนเงินที่สามารถเติมเงินคือ 50, 100, 300, 500 และ 1000 บาท ทุกเครือข่าย
  - โปรแกรมสามารถเติมเงินได้ครั้งละหลาย ๆ คน แล้วเลือกว่าจะเติมให้คนไหน ในขั้นตอนไหน โดยไม่ต้องสั่งให้โปรแกรมทำงานใหม่
  - มีเมนูแสดงแจ้งผู้ใช้งานให้กดปุ่มใด เพื่ออกจากโปรแกรม
  - สรุปยอดจากการเติมเงินจากลูกค้าที่มีการเติมเงินแต่ละคนและยอดรวมทั้งหมด
5. จงเขียนโปรแกรมเพื่อคำนวนอนุกรม

$$1^4 + 2^4 + 3^4 + \dots + n^4$$

เมื่อ  $n$  เป็นจำนวนเต็มบวกใด ๆ

## บทที่ 7

# ฟังก์ชัน

เมื่อเขียนโปรแกรมไปสักระยะหนึ่ง เรามักจะได้เขียนโปรแกรมที่มีขนาดใหญ่ และหลายครั้งเรามักจะพบร่วมคำสั่งโปรแกรมบางส่วนทำงานเหมือนกัน และเราเก็บเสียเวลาในการเขียนคำสั่งโปรแกรมช้า ๆ กันนี้ นอกจานี้หากต้องทำการแก้ไขชุดคำสั่งเหล่านี้ ยังทำให้เราต้องเสียเวลามาก เพราะต้องแก้ไขคำสั่งโปรแกรมทุก ๆ จุด

โดยทั่วไปคำสั่งโปรแกรมที่เขียนช้ากันบ่อย ๆ ราคารวบรวมแยกอุปกรณ์สร้างเป็นโปรแกรมย่อยหรือที่เรียกว่า ฟังก์ชัน (Functions) ทำให้คำสั่งโปรแกรมของเรามีความเป็นระเบียบและแก้ไขได้ง่าย โดยฟังก์ชันแบ่งออกเป็น 2 ชนิด ได้แก่ ฟังก์ชันที่ผู้เขียนโปรแกรมสร้างขึ้นมาใช้งานเอง (User Defined Functions) และฟังก์ชันที่สร้างขึ้นมาจากผู้เขียนโปรแกรมท่านอื่นถูกเก็บไว้เป็นไลบรารี (Library) มีทั้งนำมาใช้ได้พร้อมและแบบมีค่าใช้จ่าย

ที่ผ่านมาเราได้เรียนใช้งานฟังก์ชันภายในภาษาไพธอน (Built-in Functions) ที่ภาษาไพธอนได้จัดเตรียมไว้ให้ใช้งาน เมื่อสั่งให้โปรแกรมทำงานภาษาไพธอนจะโหลดฟังก์ชันเหล่านี้เข้าสู่หน่วยความจำ เราสามารถตรวจสอบชื่อฟังก์ชันภายในได้โดยการใช้คำสั่ง `dir('__builtin__')` และเรียกใช้งานโดยไม่จำเป็นต้องใช้คำสั่ง `import`

### ตัวอย่าง 7.1

การตรวจสอบชื่อฟังก์ชันภายในภาษาไพธอน (Built-in Functions)

```
1 print(dir('__builtins__'))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', 
__doc__, '__eq__', '__format__', '__ge__', '__getattribute__', 
__getitem__, '__getnewargs__', '__gt__', '__hash__', '__init__', 
__init_subclass__, '__iter__', '__le__', '__len__', '__lt__', 
__mod__, '__mul__', '__ne__', '__new__', '__reduce__', 
__reduce_ex__, '__repr__', '__rmod__', '__rmul__', 
__setattr__, '__sizeof__', '__str__', '__subclasshook__', 
'capitalize', 'casifold', 'center', 'count', ..., 'rpartition', 
'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 
'swapcase', 'title', 'translate', 'upper', 'zfill']
```

## 7.1 การสร้างฟังก์ชันขึ้นมาใช้งาน

การสร้างฟังก์ชันขึ้นมาใช้งาน (Creating Function) เป็นการกำหนดหน้าที่ให้กับคำสั่งโปรแกรมทำงานเฉพาะอย่าง เพื่อให้ได้ผลลัพธ์ตามที่ผู้เขียนโปรแกรมต้องการ มีรูปแบบการเขียนคำสั่งโปรแกรมสร้างฟังก์ชันดังต่อไปนี้

```
def function_name ([parameters]) :  
    statements  
    return [var, expression]
```

<b>def</b>	คำสั่งที่ใช้สร้างฟังก์ชัน
<b>function_name</b>	ชื่อฟังก์ชัน ห้ามตั้งชื่อซ้ำกับคำส่วน
<b>parameters</b>	ตัวแปรที่ใช้สำหรับรับค่าข้อมูล กำหนดหรือไม่ก็ได้ ถ้ากำหนดต้องมีเท่ากับค่าตัวแปรอาร์กิวเม้นต์ที่ส่งมา
<b>statements</b>	ชุดคำสั่งการทำงานภายในฟังก์ชัน
<b>return</b>	คำสั่งคืนค่าผลลัพธ์กลับ ถ้าไม่มีการคืนค่ากลับไปยังโปรแกรมที่เรียกใช้งาน จะแสดงผลลัพธ์ด้วยคำว่า <b>None</b>
<b>var, expression</b>	ตัวแปรหรือนิพจน์ที่เก็บผลลัพธ์คืนค่ากลับ

จากรูปแบบการเขียนคำสั่งโปรแกรมสร้างฟังก์ชัน สามารถแบ่งวิธีการสร้างและวิธีการทำงานของฟังก์ชันออกเป็น 4 รูปแบบดังต่อไปนี้

### 7.1.1 การสร้างฟังก์ชันที่ไม่มีการส่งค่าและรับค่า

เป็นการสร้างฟังก์ชันที่ไม่มีพารามิเตอร์ค้อยรับอาร์กิวเม้นต์ หลังจากฟังก์ชันทำงานเสร็จก็จะไม่มีการส่งค่ากลับคืนไปยังโปรแกรมที่เรียกใช้งาน มีรูปแบบการเขียนคำสั่งโปรแกรมดังต่อไปนี้

## ตัวอย่าง 7.2

การเขียนคำสั่งโปรแกรมแบบไม่มีการส่งค่าและรับค่า

```
1 def show_greeting1(): # ประกาศฟังก์ชันโดยไม่มีพารามิเตอร์อยู่รับค่า
2     print('Hello') # แสดงผลลัพธ์
3
4 show_greeting1() # เรียกใช้งานฟังก์ชัน show_greeting1()
```

Hello



จากตัวอย่างเมื่อสั่งให้โปรแกรมทำงาน บรรทัดที่ 3 จะเรียกฟังก์ชันให้แสดงผลคำว่า 'Hello'

### 7.1.2 การสร้างฟังก์ชันที่มีการรับค่า แต่ไม่มีการส่งค่ากลับ

เป็นการสร้างฟังก์ชันที่มีพารามิเตอร์อยู่รับค่า อาร์กิวเมนต์ หลังจากฟังก์ชันทำงานเสร็จจะไม่มีการส่งค่ากลับไปยังโปรแกรมที่เรียกใช้งาน มีรูปแบบการเขียนคำสั่งโปรแกรมดังตัวอย่างต่อไปนี้

## ตัวอย่าง 7.3

การเขียนคำสั่งโปรแกรมแบบสร้างฟังก์ชัน โดยมีการส่งค่าและรับค่า แต่ไม่มีการส่งค่ากลับ

```
1 def show_greeting2(name): # ประกาศฟังก์ชันโดยมีพารามิเตอร์อยู่รับค่า
2     print('Hello', name) # แสดงผลลัพธ์จากค่าพารามิเตอร์
3
4 show_greeting2('Doraemon') # เรียกใช้งานฟังก์ชันพร้อมส่งอาร์กิวเมนต์
```

Hello Doraemon



ฟังก์ชัน `show_greeting2()` แสดงผลการทักทายผู้ใช้งานคล้ายกับตัวอย่างที่ผ่านมา แต่เมื่อเรียกใช้งานฟังก์ชันในบรรทัดที่ 3 มีการส่งค่าอาร์กิวเมนต์คือคำว่า '`Doraemon`' ไปให้กับฟังก์ชันที่มีค่าพารามิเตอร์ชื่อ `name` อยู่รับค่า จากนั้นฟังก์ชันจะนำพารามิเตอร์แสดงผลร่วมกับคำว่า '`Hello`' ในบรรทัดที่ 2

### 7.1.3 การสร้างฟังก์ชันที่ไม่มีการรับค่า แต่มีการส่งค่ากลับ

มีวิธีการสร้างฟังก์ชันขึ้นมาใช้งานเหมือนกับวิธีแรก แต่เมื่อฟังก์ชันทำงานเสร็จจะมีการส่งค่ากลับด้วยคำสั่ง **return** มีรูปแบบการเขียนคำสั่งโปรแกรมดังตัวอย่างต่อไปนี้

#### ตัวอย่าง 7.4

การเขียนคำสั่งโปรแกรมแบบไม่มีการส่งค่าและรับค่า แต่ส่งค่ากลับด้วยคำสั่ง **return**

```
1 def show_greeting3(): # ประกาศฟังก์ชันโดยไม่มีพารามิเตอร์คoyerรับค่า
2     greeting = 'Hello, how are you?' # สร้างตัวแปรพร้อมกำหนดค่า
3     return greeting # ส่งค่าตัวแปร greeting กลับ เมื่อการเรียกใช้งานฟังก์ชัน
4
5 print(show_greeting3()) # แสดงผลลัพธ์จากการเรียกใช้งานฟังก์ชัน
```

Hello, how are you?



จากตัวอย่างโปรแกรมเป็นการเรียกใช้งานฟังก์ชัน **show\_greeting3()** แต่ไม่มีการส่งค่า าร์กิวเมนต์ให้กับฟังก์ชัน และฟังก์ชันก็ไม่มีพารามิเตอร์คoyerรับค่า แต่ฟังก์ชันสามารถส่งค่ากลับมายัง โปรแกรมที่เรียกใช้งานได้ โดยการใช้คำสั่ง **return** จากตัวอย่างจะคืนค่าตัวแปร **greeting** กลับมา แสดงผล

### 7.1.4 การสร้างฟังก์ชันที่มีการรับค่า พิริอุ่มทั้งมีการส่งค่ากลับ

เป็นการสร้างฟังก์ชันที่มีพารามิเตอร์คoyerรับอาร์กิวเมนต์ หลังจากฟังก์ชันทำงานเสร็จก็จะคืนค่ากลับ มา�ังโปรแกรมที่เรียกใช้งานด้วยคำสั่ง **return** มีรูปแบบการเขียนคำสั่งโปรแกรมดังตัวอย่างต่อไปนี้

#### ตัวอย่าง 7.5

การเขียนคำสั่งโปรแกรมที่มีการส่งค่า-รับค่า และส่งค่ากลับด้วยคำสั่ง **return**

```
1 def show_greeting4(name): # ประกาศฟังก์ชันโดยมีพารามิเตอร์คoyerรับค่า
2     # สร้างตัวแปรพร้อมกำหนดค่า
3     greeting_name = 'Hello, ' + name + '. ' + 'Where are you?'
4     # ส่งค่าตัวแปร greeting กลับ เมื่อการเรียกใช้งานฟังก์ชัน
5     return greeting_name
6
7 # แสดงผลลัพธ์จากการเรียกใช้งานฟังก์ชันพร้อมมีการส่งค่า
8 print(show_greeting4('Doraemon'))
```

Hello, Doraemon. Where are you?



จากตัวอย่างโปรแกรมมีการส่งอาร์กิวเมนต์ไปให้กับพารามิเตอร์ของฟังก์ชัน `show_greeting4()` โดยฟังก์ชันจะเก็บข้อมูลรวมกันไว้ที่ตัวแปร `greeting_name` จากนั้นใช้คำสั่ง `return` ส่งกลับมายังคำสั่งโปรแกรมที่เรียกใช้งานฟังก์ชัน

## 7.2 อาร์กิวเมนต์และพารามิเตอร์

ในการเขียนโปรแกรมแบบฟังก์ชัน เราจะพบคำพิพธ์ที่น่าสนใจอีก 2 คำคือ อาร์กิวเมนต์ และพารามิเตอร์

- อาร์กิวเมนต์ (Argument) คือ ค่าตัวแปรหรือค่าคงที่ซึ่งอ้างถึงในตอนเรียกใช้สำหรับส่งค่าไปให้กับฟังก์ชันที่มีการเรียกใช้งาน ค่าอาร์กิวเมนต์จะถูกแนบสองไปพร้อมกับชื่อฟังก์ชันอยู่ในเครื่องหมายวงเล็บ หากมีอาร์กิวเมนต์หลายตัวจะคั่นด้วยเครื่องหมาย Comma ( , )
- พารามิเตอร์ (Parameter) คือ ตัวแปรที่ประกาศไว้ในฟังก์ชันเพื่อรับค่าอาร์กิวเมนต์ อยู่ในเครื่องหมายวงเล็บหลังชื่อฟังก์ชัน และคั่นด้วยเครื่องหมาย Comma ( , ) ถ้ามีพารามิเตอร์มากกว่าหนึ่งตัว

เมื่อมีการเรียกใช้งานฟังก์ชันและส่งค่าอาร์กิวเมนต์ไปให้พารามิเตอร์ที่ประกาศไว้รับค่า จำนวนอาร์กิวเมนต์และจำนวนพารามิเตอร์จะต้องเท่ากัน ใน การส่งค่าอาร์กิวเมนต์มีอยู่ด้วย 2 วิธีด้วยกันคือ การส่งค่าข้อมูล (Call by Value หรือ Pass by Value) และการส่งค่าแบบอ้างอิง (Call by Reference หรือ Pass by Reference)

- Call by Value หรือ Pass by Value ค่าอาร์กิวเมนต์จะถูกทำการ copy แล้วส่งให้กับพารามิเตอร์ของฟังก์ชันที่เราเรียกใช้งาน จำนวนพารามิเตอร์ต้องเท่ากับจำนวนอาร์กิวเมนต์ที่ส่งมาให้ การเปลี่ยนแปลงค่าพารามิเตอร์ภายในฟังก์ชัน จะไม่มีผลกระทบต่อค่าอาร์กิวเมนต์เมื่อมีการส่งค่ากลับ การส่งค่าอาร์กิวเมนต์แบบนี้ทำให้สิ้นเปลืองเนื้อที่ในหน่วยความจำ
- Call by Reference หรือ Pass by Reference ใช้กับข้อมูลชนิดออบเจ็ค โดยส่งค่าตำแหน่งอาร์กิวเมนต์ในหน่วยความจำ ไปให้กับพารามิเตอร์ในฟังก์ชันที่เรียกใช้งาน เมื่อมีการเปลี่ยนแปลงข้อมูลของค่าพารามิเตอร์จะทำให้ค่าอาร์กิวเมนต์ที่ส่งไปเปลี่ยนตามไปด้วย การส่งอาร์กิวเมนต์แบบนี้มีข้อดีคือ ประหยัดเนื้อที่ในหน่วยความจำ เพราะค่าอาร์กิวเมนต์ไม่ต้องมีการ copy อีกชุดหนึ่งเหมือนกับวิธีแรก และลดเวลาในการประมวลผลข้อมูล

นอกจากนี้ในภาษาไพธอนยังมีกลไกการส่งอาร์กิวเมนต์ที่เรียกว่า Call by Object บางครั้งเรียกว่า Call by Object Reference หรือ Call by Sharing ถ้าหากเราส่งค่าอาร์กิวเมนต์เป็นชนิดข้อมูลที่ไม่สามารถเปลี่ยนรูปได้ (Immutable) เช่น ชนิดข้อมูลจำนวนเต็ม ตริง ทูเพิล เป็นต้น ไปยังฟังก์ชันผ่านค่าพารามิเตอร์ หากมีการเปลี่ยนค่าพารามิเตอร์ภายในฟังก์ชัน จะไม่มีผลกระทบต่อค่าอาร์กิวเมนต์ที่ต้นทาง คือ มีลักษณะการทำงานเหมือนกับ Call by Value แต่หากเราส่งค่าอาร์กิวเมนต์ที่เป็นชนิดข้อมูลลิสต์ ซึ่งเป็นชนิดข้อมูลที่เปลี่ยนแปลงได้ (Mutable) จะส่งผลให้มีการเปลี่ยนแปลงค่าพารามิเตอร์ภายในฟังก์ชัน ค่าอาร์กิวเมนต์ต้นทางก็จะถูกเปลี่ยนค่าตามไปด้วย ดังแสดงในตัวอย่างต่อไปนี้

## ตัวอย่าง 7.6

การเขียนคำสั่งโปรแกรมที่มีการส่งค่า-รับค่าแบบ Call by Value

```
1 def msg(f_str): # ประกาศฟังก์ชันโดยมีพารามิเตอร์อยู่รับค่า
2     # แสดงผลลัพธ์ค่าพารามิเตอร์
3     print('ค่าพารามิเตอร์ที่รับมา = ', f_str)
4     # เปลี่ยนค่าข้อมูลพารามิเตอร์ f_str
5     f_str = 'EASY'
6     # แสดงผลลัพธ์หลังจากเปลี่ยนค่าพารามิเตอร์
7     print('เปลี่ยนค่าพารามิเตอร์ = ', f_str)
8
9
10 # กำหนดค่าให้กับตัวแปร
11 arg_str = 'PYTHON'
12 # แสดงผลลัพธ์ค่าตัวแปร arg_str ก่อนส่งค่าให้กับฟังก์ชัน
13 print('ค่าอาร์กิวเม้นต์ที่ส่งให้ค่าพารามิเตอร์ = ', arg_str)
14 # ส่งค่าอาร์กิวเม้นต์ให้กับฟังก์ชัน
15 msg(arg_str)
16 # แสดงผลลัพธ์ค่าตัวแปร arg_str หลังจากที่ฟังก์ชันส่งกลับ
17 print ('ค่าอาร์กิวเม้นต์หลังจากเปลี่ยนค่าพารามิเตอร์ส่งกลับ = ', arg_str)
```

ค่าอาร์กิวเม้นต์ที่ส่งให้ค่าพารามิเตอร์ = PYTHON  
ค่าพารามิเตอร์ที่รับมา = PYTHON  
เปลี่ยนค่าพารามิเตอร์ = EASY  
ค่าอาร์กิวเม้นต์หลังจากเปลี่ยนค่าพารามิเตอร์ส่งกลับ = PYTHON



จากตัวอย่างโปรแกรมภายในฟังก์ชันมีการเปลี่ยนค่าพารามิเตอร์ `f_str` ในบรรทัดที่ 3 จะมีผลทำให้เปลี่ยนแปลงเฉพาะภายในฟังก์ชันเท่านั้น ไม่กระทบถึงตัวแปร `arg_str`

## ตัวอย่าง 7.7

การเขียนคำสั่งโปรแกรมส่งค่า-รับค่า และมีการเปลี่ยนแปลงข้อมูล

```
1 def msg(f_lst):
2     print('ค่าพารามิเตอร์ที่รับมา = ', f_lst)
3     f_lst.append('EASY')
4     print('เปลี่ยนค่าพารามิเตอร์ = ', f_lst)
5
6 arg_lst = ['PYTHON IS']
7 print('ค่าอาร์กิวเม้นต์ที่ส่งให้ค่าพารามิเตอร์ = ', arg_lst)
8 msg(arg_lst)
9 print('ค่าอาร์กิวเม้นต์หลังจากเปลี่ยนค่าพารามิเตอร์ = ', arg_lst)
```

```
ค่าอาร์กิวเม้นต์ที่ส่งให้ค่าพารามิเตอร์ = ['PYTHON IS']
ค่าพารามิเตอร์ที่รับมา = ['PYTHON IS']
เปลี่ยนค่าพารามิเตอร์ = ['PYTHON IS', 'EASY']
ค่าอาร์กิวเม้นต์หลังจากเปลี่ยนค่าพารามิเตอร์ = ['PYTHON IS', 'EASY']
```



จากตัวอย่างโปรแกรมเราจะพบว่า เมื่อมีการส่งค่าอาร์กิวเม้นต์ที่เป็นชนิดข้อมูลลิสต์ซึ่งเป็นชนิดข้อมูลที่สามารถแก้ไขข้อมูลได้ ส่งไปให้กับฟังก์ชันที่มีพารามิเตอร์คoyerรับค่าอยู่ และภายในฟังก์ชัน มีเปลี่ยนแปลงค่าพารามิเตอร์ จากตัวอย่างในบรรทัดที่ 3 ใช้เมธอด `append()` เพิ่มข้อมูลต่อท้ายลิสต์ หลังจากสั่งให้โปรแกรมแสดงผลในบรรทัดที่ 4 ค่าพารามิเตอร์จะแสดงผลค่าข้อมูลที่เพิ่มเข้าไปในฟังก์ชันด้วย และเมื่อแสดงผลค่าอาร์กิวเม้นต์ในบรรทัดที่ 8 จะเห็นว่าค่าอาร์กิวเม้นต์ก็จะเปลี่ยนตามค่าพารามิเตอร์ที่ถูกเปลี่ยนแปลง

### 7.3 รูปแบบการส่งค่าอาร์กิวเม้นต์ให้กับค่าพารามิเตอร์

การสร้างฟังก์ชันขึ้นมาใช้งานเพื่อทำหน้าที่อย่างใดอย่างหนึ่งนั้น มีรูปแบบการส่งค่าอาร์กิวเม้นต์และการกำหนดค่าพารามิเตอร์ของฟังก์ชันที่ถูกเรียกใช้งานได้หลากหลายวิธี ซึ่งเราจะได้ศึกษาและนำไปประยุกต์ใช้งานในการพัฒนาโปรแกรมในหัวข้อนี้

### 7.3.1 การส่งค่าอาร์กิวเมนต์แบบ Required arguments

คือวิธีการส่งค่าอาร์กิวเมนต์ไปให้กับค่าพารามิเตอร์ในฟังก์ชันที่ถูกเรียกใช้งาน โดยจำนวนของค่าอาร์กิวเมนต์ต้องเท่ากับจำนวนพารามิเตอร์ที่มีอยู่ ซึ่งค่าพารามิเตอร์จะแสดงผลตามชนิดข้อมูลที่ค่าอาร์กิวเมนต์ส่งให้ หากจำนวนอาร์กิวเมนต์และพารามิเตอร์ไม่เท่ากันจะทำให้เกิดการแจ้งเตือนข้อผิดพลาด ผู้เขียนโปรแกรมอาจจะตั้งชื่ออาร์กิวเมนต์และพารามิเตอร์เหมือนหรือต่างกันก็ได้ แต่ขอแนะนำให้ตั้งชื่อเหมือนกันและให้สื่อความหมายจะดีกว่า ดังแสดงในตัวอย่างต่อไปนี้

#### ตัวอย่าง 7.8

การส่งค่าอาร์กิวเมนต์แบบ Required arguments ไปให้กับค่าพารามิเตอร์

```
1 def req_arg(numlst, str_, num):
2     print('แสดงค่าข้อมูลในลิสต์ = ', numlst)
3     print('แสดงค่าข้อมูลในตัวแปร num = ', num)
4     print('แสดงค่าข้อมูลในตัวแปร str_ = ', str_)
5
6 lst = [1,2,3]
7 msg = 'Python'
8 req_arg(lst, msg, 50)
```

```
แสดงค่าข้อมูลในลิสต์ = [1, 2, 3]
แสดงค่าข้อมูลในตัวแปร num = 50
แสดงค่าข้อมูลในตัวแปร str_ = Python
```



จากตัวอย่างเราสามารถส่งค่าอาร์กิวเมนต์ไปให้กับค่าพารามิเตอร์ได้หลายชนิดข้อมูล แต่ต้องมีจำนวนเท่ากันและต้องเรียงลำดับให้ถูกต้องด้วย

### 7.3.2 การส่งอาร์กิวเมนต์แบบ Keyword arguments

เป็นวิธีการส่งค่าอาร์กิวเมนต์ไปให้กับค่าพารามิเตอร์ที่ฟังก์ชันถูกเรียกใช้งาน ที่ไม่จำเป็นต้องเรียงลำดับชื่อของค่าพารามิเตอร์ตามชนิดข้อมูลที่ค่าอาร์กิวเมนต์ส่งให้ แต่มีข้อแม้ว่าชื่อของค่าอาร์กิวเมนต์กับค่าพารามิเตอร์ต้องมีชื่อเหมือนกัน และค่าทั้งสองต้องมีจำนวนเท่ากันด้วย

### ตัวอย่าง 7.9

การส่งค่าอาร์กิวเมนต์แบบ Keyword arguments

```
1 def key_arg(lst, msg, num):
2     print('แสดงค่าข้อมูลในลิสต์ = ', lst)
3     print('แสดงค่าข้อมูลในตัวแปร num = ', num)
4     print('แสดงค่าข้อมูลในตัวแปร msg = ', msg)
5
6 key_arg(lst=[1, 2, 3], num=10, msg='Python is easy')
```

```
แสดงค่าข้อมูลในลิสต์ = [1, 2, 3]
แสดงค่าข้อมูลในตัวแปร num = 10
แสดงค่าข้อมูลในตัวแปร msg = Python is easy
```



จากตัวอย่างจะเห็นว่าค่าอาร์กิวเมนต์และค่าพารามิเตอร์มีชื่อเหมือนกัน แต่มีการสลับตำแหน่งชื่อค่าพารามิเตอร์ในฟังก์ชัน เมื่อโปรแกรมทำงานและส่งค่าอาร์กิวเมนต์ไปให้กับค่าพารามิเตอร์ก็จะอ้างอิงตามชื่อ

### 7.3.3 การส่งอาร์กิวเมนต์แบบ Default arguments

มีค่าพารามิเตอร์บางตัวได้ถูกกำหนดค่าไว้ล่วงหน้า โดยเมื่อเรียกใช้ก็จะส่งค่าอาร์กิวเมนต์เฉพาะจำนวนที่เหลือไปให้กับค่าพารามิเตอร์กับฟังก์ชันที่ถูกเรียกใช้งานเท่านั้น

### ตัวอย่าง 7.10

การหาค่าเส้นรอบวงกลมโดยมีการกำหนดค่าคงที่ให้กับค่าพารามิเตอร์ไว้

```
1 def area_circle(r, pi=3.14):
2     result = 2 * pi * r
3     print('ความยาวเส้นรอบวงกลม = ', result)
4
5 area_circle(7)
6 area_circle(7, 3.1415926)
7 area_circle(7, pi=3.1415926)
```

```
ความยาวเส้นรอบวงกลม = 43.96
ความยาวเส้นรอบวงกลม = 43.9822964
ความยาวเส้นรอบวงกลม = 43.9822964
```



จากตัวอย่างโปรแกรมได้กำหนดพารามิเตอร์ชื่อ `pi=3.14` และมีพารามิเตอร์ชื่อ `r` ซึ่งเป็นค่ารัศมีที่ค่อยรับค่าจากอาร์กิวเมนต์ เมื่อเรียกใช้งานฟังก์ชัน `def_arg()` ก็ส่งอาร์กิวเมนต์เพียงค่าเดียวไปให้พารามิเตอร์ `r` ผลลัพธ์ที่แสดงออกมาก็คือขนาดของเส้นรอบวงกลม

#### 7.3.4 การส่งอาร์กิวเมนต์แบบ Variable-length arguments

เป็นการสร้างพารามิเตอร์ไว้ค่อยรับอาร์กิวเมนต์แบบไม่จำกัดจำนวน โดยชื่อของพารามิเตอร์ประเภทนี้จะมีเครื่องหมาย (`*`) นำหน้า เมื่อต้องการนำข้อมูลในพารามิเตอร์มาแสดงผลหรือนำไปประมวลผล ให้ระบุตำแหน่งด้วยเครื่องหมาย `[ ... ]` เพราะเป็นพารามิเตอร์ชนิดข้อมูลทุกเพิล

ตัวอย่าง 7.11

การกำหนดพารามิเตอร์แบบ Variable-length arguments

```
1 def varleng_arg(*num):
2     print('จำนวนข้อมูลในตัวแปร num :')
3     for var in num:
4         print(var, end=' ')
5     print(' ')
6     result = num[2] * 2
7     print(f'ผลคูณพารามิเตอร์ num[2] * 2 = {result}')
8
9 print('-' * 50)
10 varleng_arg(10, 20, 30, 40)
11 print('-' * 50)
12 varleng_arg(2, 3, 5, 7, 11, 13, 17, 19)
13 print('-' * 50)
```

```
-----  
จำนวนข้อมูลในตัวแปร num :  
10 20 30 40  
ผลคูณพารามิเตอร์ num[2] * 2 = 60  
-----
```

```
-----  
จำนวนข้อมูลในตัวแปร num :  
2 3 5 7 11 13 17 19  
ผลคูณพารามิเตอร์ num[2] * 2 = 10  
-----
```

เมื่อโปรแกรมทำงานจะเรียกใช้งานฟังก์ชัน `varleng_arg()` ที่มีพารามิเตอร์ `*num` รองรับอาร์กิวเมนต์ที่ส่งมาให้แบบไม่จำกัด จากตัวอย่างโปรแกรมมีการส่งอาร์กิวเมนต์ไปให้พารามิเตอร์ทั้งหมด 4 ค่า เราสามารถแสดงผลค่าข้อมูลในพารามิเตอร์ด้วยคำสั่ง `for` และนำข้อมูลในพารามิเตอร์มาประมวลด้วยการระบุตำแหน่งด้วยเครื่องหมาย `[ ... ]`

## 7.4 การสร้างฟังก์ชันด้วยคำสั่ง `lambda`

โดยปกติการสร้างฟังก์ชันขึ้นมาใช้งานจะใช้คำสั่ง `def` และมีการตั้งชื่อให้แต่ละฟังก์ชันนั้น ๆ แต่เราสามารถที่จะสร้างฟังก์ชันขึ้นมาใช้งานโดยไม่ต้องกำหนดชื่อ ด้วยการเรียกใช้คำสั่ง `lambda` (แลมบด้า) โดยมีรูปแบบการใช้งานดังนี้

```
var = lambda argument_lists ; expression
```

`var` ตัวแปรที่ค่อยรับค่าจากการดำเนินของฟังก์ชัน `lambda`

`argument_lists` อาร์กิวเมนต์ส่งค่าไปประมวลผล

`expression` นิพจน์ที่ใช้สำหรับการประมวลผล

### ตัวอย่าง 7.12

การสร้างฟังก์ชัน `lambda` หาปริมาตรทรงกระบอก

```
1 volume = lambda r, h: 3.14 * (r * r) * h
2 print(f'ปริมาตรทรงกระบอกที่ 1 = {volume(2, 3)}')
3 print(f'ปริมาตรทรงกระบอกที่ 2 = {volume(3, 5)}')
4 print(f'ปริมาตรทรงกระบอกที่ 3 = {volume(4, 12)}')
```

ปริมาตรทรงกระบอกที่ 1 = 37.68

ปริมาตรทรงกระบอกที่ 2 = 141.3

ปริมาตรทรงกระบอกที่ 3 = 602.88



จากตัวอย่างเป็นการสร้างฟังก์ชันด้วยคำสั่ง `lambda` เมื่อเปรียบเทียบกับวิธีการสร้างฟังก์ชันโดยใช้คำสั่ง `def` จะมีขนาดคำสั่งที่กระทำด้วยกันน้อยกว่า และอาร์กิวเมนต์หลังคำสั่ง `lambda` จะมีกี่ตัวก็ได้ ซึ่งถูกคั่นด้วยเครื่องหมาย Comma (,) และหลังอาร์กิวเมนต์ตัวสุดท้ายจะถูกปิดท้ายด้วยเครื่องหมาย Colon (:) ก่อนเขียนนิพจน์การคำนวณค่า

การสร้างฟังก์ชันด้วยคำสั่ง `lambda` ไม่จำเป็นต้องส่งค่าอาร์กิวเมนต์ก็ได้ โดยจะมีเฉพาะนิพจน์สำหรับประมวลผลข้อมูลเท่านั้น ดังตัวอย่างต่อไปนี้

### ตัวอย่าง 7.13

การสร้างฟังก์ชันด้วยคำสั่ง lambda ที่ไม่มีการส่งค่าอาร์กิวเมนต์

```
1 volume = lambda: 3.14 * 5 * 5 * 10
2 print(f'ปริมาตรทรงกระบอก = {volume()}' )
```

ปริมาตรทรงกระบอก = 785.0



จากตัวอย่างโปรแกรมเป็นการสร้างปริมาตรทรงกระบอกด้วยฟังก์ชัน **lambda** แต่ไม่มีการส่งค่าอาร์กิวเมนต์ให้กับฟังก์ชัน **lambda** เราแสดงผลลัพธ์จากการคำนวณของฟังก์ชันในค่าตัวแปร **volume()** ได้เลย

นอกจากนี้คำสั่ง **lambda** ยังมีความสามารถสร้างฟังก์ชันขึ้นมาใช้งานได้ครั้งละหลาย ๆ ฟังก์ชัน โดยเป็นการสร้างฟังก์ชัน **lambda** ซ้อนเข้าไปในชนิดข้อมูลลิสต์อีกครั้ง และหากต้องการให้ฟังก์ชันได้ทำการประมวลผล ให้ระบุตำแหน่ง (index) ของฟังก์ชันนั้น ๆ ดังตัวอย่างต่อไปนี้

### ตัวอย่าง 7.14

การสร้างฟังก์ชัน lambda ครั้งละหลาย ๆ ฟังก์ชันที่อยู่ภายในชนิดข้อมูลลิสต์

```
1 area_volume = [
2     lambda r, h: 3.14 * r * r * h,
3     lambda r: 3.14 * r * r,
4     lambda b, h: 0.5 * b * h,
5     lambda w, h: w * h
6 ]
7
8 print('ปริมาตรทรงกระบอก = ', area_volume[0](5, 6), 'ลูกบาศก์เมตร')
9 print('พื้นที่วงกลม = ', area_volume[1](2), 'ตารางเมตร')
10 print('พื้นที่สามเหลี่ยม = ', area_volume[2](5, 7), 'ตารางเมตร')
11 print('พื้นที่สี่เหลี่ยมผืนผ้า = ', area_volume[3](10, 12), 'ตารางเมตร')
```

ปริมาตรทรงกระบอก = 471.0 ลูกบาศก์เมตร  
พื้นที่วงกลม = 12.56 ตารางเมตร  
พื้นที่สามเหลี่ยม = 17.5 ตารางเมตร  
พื้นที่สี่เหลี่ยมผืนผ้า = 120 ตารางเมตร



จากตัวอย่างโปรแกรมได้แสดงให้เห็นถึงการสร้างฟังก์ชัน **lambda** ขึ้นมาใช้งาน ที่มีหลายฟังก์ชันในชนิดข้อมูลลิสต์ ซึ่งมีความสะดวกและประหยัดเวลามากกว่าการใช้คำสั่ง **def** สร้างฟังก์ชัน

### ตัวอย่าง 7.15

การสร้างฟังก์ชัน lambda ครั้งละหลาย ๆ ฟังก์ชันที่อยู่ภายในชนิดข้อมูลดิคชันนารี

```
1 geometry = {  
2     'cylinderVol': lambda r, h: 3.14 * r * r * h,  
3     'circleArea': lambda r: 3.14 * r * r,  
4     'triangleArea': lambda b, h: 0.5 * b * h,  
5     'rectangleArea': lambda w, h: w * h  
6 }  
7  
8 print('ปริมาตรทรงกระบอก = ', geometry['cylinderVol'](5, 6), 'ลูกบาศก์เมตร')  
9 print('พื้นที่วงกลม = ', geometry['circleArea'](2), 'ตารางเมตร')  
10 print('พื้นที่สามเหลี่ยม = ', geometry['triangleArea'](5, 7), 'ตารางเมตร')  
11 print('พื้นที่สี่เหลี่ยมผืนผ้า = ', geometry['rectangleArea'](10, 12), 'ตารางเมตร')
```

ปริมาตรทรงกระบอก = 471.0 ลูกบาศก์เมตร  
พื้นที่วงกลม = 12.56 ตารางเมตร  
พื้นที่สามเหลี่ยม = 17.5 ตารางเมตร  
พื้นที่สี่เหลี่ยมผืนผ้า = 120 ตารางเมตร



จากตัวอย่างโปรแกรมได้แสดงให้เห็นถึงการสร้างฟังก์ชัน **lambda** ขึ้นมาใช้งาน ที่มีหลายฟังก์ชัน ในชนิดข้อมูลดิคชันนารี ซึ่งมีความสะดวกกว่าการใช้ข้อมูลลิสต์ในกรณีที่เราไม่สะดวกในการอ้างอิงด้วย Index

## 7.5 ขอบเขตการเรียกใช้งานตัวแปร

ในหัวข้อนี้เราจะได้เรียนรู้ขอบเขตการเรียกใช้งานตัวแปร (Variable Scope) ในภาษาไพธอน โดยทั่วไป แล้วตัวแปรจะถูกร่างขึ้นมาสำหรับเก็บข้อมูลใดชนิดหนึ่ง จากนั้นนำไปประมวลผลหาผลลัพธ์ตามที่ผู้พัฒนาโปรแกรมต้องการ ตัวแปรแบ่งออกเป็น 2 ชนิดได้แก่

1. ตัวแปรโกลบอล (Global) คือ ตัวแปรที่ไว้สำหรับรับเก็บข้อมูลใดชนิดหนึ่ง จำกันไว้ไปประมวลผลหาผลลัพธ์ตามที่ผู้พัฒนาโปรแกรมต้องการ ตัวแปรแบ่งออกเป็น 2 ชนิดได้แก่
2. ตัวแปรโลคอล (Local) คือตัวแปรที่สร้างขึ้นมาแล้วเรียกใช้งานได้เฉพาะภายในฟังก์ชันเท่านั้น

เมื่อต้องการสร้างตัวแปรภายในฟังก์ชันแล้วให้ฟังก์ชันอื่น ๆ เรียกใช้งานได้ ให้ใช้คีย์เวิร์ด **global** นำหน้าชื่อตัวแปรนั้น เช่น **global area, global weight** เป็นต้น

### ตัวอย่าง 7.16

การสร้างตัวแปรโกลบออลและการเรียกใช้งานโดยฟังก์ชัน

```
1 def cylinder():
2     volume = 3.14 * r * r * h # volume เป็นตัวแปรชนิดข้อมูลโกลบออล
3     return volume
4
5 r = int(input('ป้อนค่ารัศมี : '))
6 h = int(input('ป้อนค่าความสูง : '))
7 print('ปริมาตรทรงกระบอก = ', cylinder(), 'ลูกบาศก์เมตร')
```

ป้อนค่ารัศมี : 2  
ป้อนค่าความสูง : 3  
ปริมาตรทรงกระบอก = 37.68 ลูกบาศก์เมตร



จากตัวอย่างโปรแกรมตัวแปร `r` และ `h` ถูกประกาศไว้เป็นชนิดข้อมูลโกลบออล ทำให้ฟังก์ชันสามารถเรียกใช้งานได้ แต่สำหรับตัวแปร `volume` เป็นตัวแปรชนิดโอล寇ล ซึ่งจะทำงานได้เฉพาะภายในฟังก์ชันเท่านั้น

### ตัวอย่าง 7.17

การสร้างตัวแปรโคลออลขึ้นมาใช้งานภายในฟังก์ชัน และการเรียกใช้งานตัวแปรข้ามฟังก์ชัน

```
1 global pi # ตัวแปรชนิดข้อมูลโกลบออล
2 pi = 3.14
3
4 def circle_area():
5     global r # ตัวแปรชนิดข้อมูลโกลบออล
6     r = 4
7     result = pi * (r * r) # ตัวแปรชนิดข้อมูลโคลออล
8     return result
9
10 def circumference():
11     result = 2 * pi * r # ตัวแปรชนิดข้อมูลโคลออล
12     return result
13
14 print(f'พื้นที่วงกลม = {circle_area()} ตารางเมตร')
15 print(f'เส้นรอบวงกลม = {circumference()} เมตร')
```

พื้นที่วงกลม = 50.24 ตารางเมตร  
เส้นรอบวงกลม = 25.12 เมตร



จากตัวอย่างโปรแกรมได้ประกาศตัวแปร `r` และ `pi` ให้เป็นตัวแปรโกลบออล จึงมีผลทำให้ฟังก์ชัน `circumference()` สามารถเรียกใช้งานและนำค่าตัวแปรไปคำนวณหาเส้นรอบวงกลมได้ทั้ง ๆ ที่ไม่ได้กำหนดค่าในโปรแกรมหลัก

ทั้งนี้การประกาศตัวแปรโกลบออลจะทำภายใต้ขอบเขตใดของโปรแกรมก็ได้ ในตัวอย่างนี้เราประกาศ `pi` ให้เป็นตัวแปรโกลบออลนอกขอบเขตของฟังก์ชัน และได้ประกาศ `r` ให้เป็นตัวแปรโกลบออลในขอบเขตของฟังก์ชัน แต่เราถึงสามารถเรียกใช้งานตัวแปรโกลบออลทั้งสองตัวนี้ใน `circumference()` ได้

### ตัวอย่าง 7.18

การสร้างตัวแปรโกลบออลและโลคอลที่มีสืบท่อตัวแปรเหมือนกัน

```
1 def calculate(cash):
2     total = cash - (cash*vat) # ตัวแปรชนิดข้อมูลโลคอล
3     return total
4
5 cash = float(input('ป้อนจำนวนเงิน : '))
6 vat = 0.07 # ตัวแปรชนิดข้อมูลโกลบออล
7 total = cash - (cash*vat) # ตัวแปรชนิดข้อมูลโกลบออล
8
9 print(f'จำนวนเงินไม่รวมภาษี 7% (ตัวแปรแบบโลคอล) = {calculate(100)} บาท')
10 print(f'จำนวนเงินไม่รวมภาษี 7% (ตัวแปรแบบโกลบออล) = {total} บาท')
```

ป้อนจำนวนเงิน : 1000

จำนวนเงินไม่รวมภาษี 7% (ตัวแปรแบบโลคอล) = 93.0 บาท

จำนวนเงินไม่รวมภาษี 7% (ตัวแปรแบบโกลบออล) = 930.0 บาท



จากตัวอย่าง 7.18 มีการสร้างตัวแปร `total` ไว้สองตำแหน่งคือ บรรทัดที่ 3 และบรรทัดที่ 5 ซึ่งอยู่ภายในฟังก์ชัน `calculate()` เมื่อนำค่าตัวแปรมาแสดงผลจะได้ผลลัพธ์ไม่เหมือนกัน เนื่องจากค่าตัวแปร `total` ในบรรทัดที่ 5 จะทำงานเฉพาะภายในฟังก์ชัน `calculate()` เท่านั้น โดยรับค่าจากบรรทัดที่ 7 ไปประมวลผล ส่วนค่าตัวแปร `vat` เป็นตัวแปรโกลบออลทำให้ฟังก์ชัน `calculate()` เรียกใช้งานและนำไปประมวลผลได้ ซึ่งตัวแปรแบบโลคอลจะนำ **100** ไปคำนวณ (บรรทัดที่ 7) ส่วนตัวแปรแบบโกลบออลจะนำ **1000** ไปคำนวณ

ตัวอย่าง 7.19

การสร้างตัวแปรโลคอลที่มีชื่อตัวแปรเหมือนกันเพื่อใช้งานในฟังก์ชัน

```
1 def sal_rate_1(m, ot):
2     ot_rate = 0
3     bonus = 0.5
4     total = (m + (ot*ot_rate)) * bonus
5     salary = total + m
6     return salary
7
8 def sal_rate_2(m, ot):
9     ot_rate = 20
10    bonus = 0.2
11    total = (m + (ot*ot_rate)) * bonus
12    salary = total + m
13    return salary
14
15 print('เงินเดือนที่ได้รับสูงชั้น = ', sal_rate_1(12000, 5), 'บาท')
16 print('เงินเดือนที่ได้รับสูงชั้น = ', sal_rate_2(20000, 5), 'บาท')
```

เงินเดือนที่ได้รับสุทธิ = 18000.0 บาท  
เงินเดือนที่ได้รับสุทธิ = 24020.0 บาท



จากตัวอย่างโปรแกรมได้สร้างฟังก์ชันขึ้นมาใช้งาน 2 ฟังก์ชัน สำหรับคำนวณเงินเดือนสุทธิหลังจากมีการคิดค่าโบนัสและค่าทำงานล่วงเวลา ให้เราสังเกตเห็นว่าทั้งสองฟังก์ชันนี้มีตัวแปรเหมือนกัน ซึ่งตัวแปรจะทำงานแบบตัวแปรโลคอล คือ ดำเนินการเฉพาะภายในฟังก์ชันเท่านั้นและไม่สามารถเรียกใช้งานข้างนอกฟังก์ชันได้

## 7.6 พังก์ชัน Built-in ภาษา Python

ภาษาไพธอนได้จัดเตรียมฟังก์ชันต่าง ๆ ไว้ให้ผู้พัฒนาโปรแกรมเรียกใช้งานเป็นจำนวนมาก **built-in function** เป็นประเภทฟังก์ชันที่สามารถเรียกใช้งานได้เลย โดยไม่ต้องใช้คำสั่ง **import** มودูลใด ๆ เข้ามานในโปรแกรมก่อน และจากหลาย ๆ บทที่ผ่านมาได้เรียกใช้งานกันไปบ้างแล้ว ในส่วนนี้จะสรุปและนำเสนอ **built-in function** อีก ๆ เพิ่มเติม ที่เราอาจจะได้นำมาใช้งานในการพัฒนาโปรแกรม

เมธอด	ความหมาย	รูปแบบการใช้งาน	คำอธิบายเพิ่มเติม
<code>abs()</code>	ใช้สำหรับหาค่าจำนวนเต็มบวก (absolute)	<code>abs(x)</code>	<code>x</code> คือ ชนิดข้อมูลจำนวนเต็มหรือศนิยม
<code>all()</code>	ใช้สำหรับตรวจสอบสมาชิกทุกตัวเป็นค่าจริง (True) หรือไม่	<code>all(seq)</code>	<code>seq</code> คือ ตัวแปรหรือชนิดข้อมูลแบบลำดับทั้งลิสต์และทูเพิล
<code>any()</code>	ใช้สำหรับตรวจสอบตัวแปรชนิดข้อมูลแบบเรียงลำดับว่า มีค่าว่างหรือไม่ ถ้า ตัวแปรเก็บค่าว่างจะคืนค่ากลับ <b>False</b> ถ้ามีสมาชิกอยู่จะคืนค่า <b>True</b>	<code>any(seq)</code>	<code>seq</code> คือ ตัวแปรหรือชนิดข้อมูลแบบลำดับทั้งลิสต์และทูเพิล
<code>ascii()</code>	ใช้สำหรับแปลงสัญลักษณ์เป็นรหัสเօสกี	<code>ascii(obj)</code>	<code>obj</code> คือ ชนิดข้อมูลอักขระหรือสตริง ลิสต์ ทูเพิล เป็นต้น
<code>callable()</code>	ใช้สำหรับตรวจสอบว่า <code>obj</code> สามารถเรียกใช้งานได้ หรือไม่ ถ้าเรียกใช้งานได้ และคืนค่า <b>True</b> เมื่อไม่สามารถเรียกใช้งานได้	<code>callable(obj)</code>	<code>obj</code> คือ ขอบเขตที่ต้องการตรวจสอบ
<code>chr()</code>	ใช้สำหรับแปลงรหัส Unicode ให้เป็นตัวอักขระ	<code>chr(i)</code>	<code>i</code> คือ รหัส Unicode

ตาราง 7.1: Built-In ฟังก์ชันในภาษาไพธอนที่ใช้งานบ่อย (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน	คำอธิบายเพิ่มเติม
<code>compile()</code>	ใช้สำหรับคอมไพล์ คำสั่งภาษาไพธอน หรือชอร์สโค้ด ส่งค่ากลับคืนมาเป็น อบเจ็ค code มี การใช้งานเหมือนกับ พิงก์ชัน <code>exec()</code> และ <code>eval()</code>	<code>compile(source, filename, mode)</code>	<code>source</code> คือ สตริงหรือ ไบต์สตริง <code>filename</code> คือ แหล่งที่เก็บไฟล์หรือไฟล์ที่ต้องอ่านขึ้นมาคอมไพล์ <code>mode</code> คือ มีด้วยกัน 3 โหมด <code>eval</code> ใช้ กับชอร์สโค้ดที่มีนิพจน์ เดียวเท่านั้น <code>exec</code> ใช้ กับชอร์สโค้ดที่ประกอบด้วยพิงก์ชันหรือคลาสได้ <code>single</code> ใช้กับชอร์สโค้ด ที่มีการโต้ตอบกับผู้ใช้งาน
<code>divmod()</code>	จะคืนค่าผลหารและเศษจากการหารออกมาก	<code>divmod(x, y)</code>	<code>x</code> คือ ตัวตั้ง <code>y</code> คือ ตัวหาร
<code>enumerate()</code>	ใช้สำหรับกำหนดหมายเลขแบบเรียงลำดับ	<code>enumerate(seq [, start])</code>	<code>seq</code> คือ ชนิดข้อมูลแบบเรียงลำดับ <code>start</code> คือ ค่าเริ่มต้น
<code>eval()</code>	ใช้สำหรับประมวลผลคำสั่งนิพจน์ที่อยู่ในรูปแบบชนิดข้อมูลอักขระหรือสตริง	<code>eval(expression)</code>	<code>expression</code> คือ นิพจน์ที่อยู่ในรูปแบบชนิดข้อมูลอักขระหรือสตริง
<code>exec()</code>	ใช้สำหรับ execute คำสั่งโปรแกรม	<code>exec(obj[, globals, locals])</code>	<code>obj</code> คือ ข้อมูลอักขระ, สตริง หรือคำสั่งโปรแกรม <code>globals, locals</code>

ตาราง 7.1: Built-In พิงก์ชันในภาษาไพธอนที่ใช้งานบ่อย (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน	คำอธิบายเพิ่มเติม
<code>filter()</code>	ใช้สำหรับกรองข้อมูลจากชนิดข้อมูลแบบเรียงลำดับ	<code>filter(func, seq)</code>	<code>func</code> คือ ฟังก์ชันที่ทำหน้าที่กรองข้อมูลจากชนิดข้อมูลแบบเรียงลำดับ <code>seq</code> คือ ชนิดข้อมูลแบบเรียงลำดับ
<code>getattr()</code>	ใช้สำหรับเรียกค่าข้อมูลแอ็ตทริบิวต์ในออบเจ็ค	<code>getattr(obj, name[, default])</code>	<code>obj</code> คือออบเจ็คที่ถูกเรียกค่าแอ็ตทริบิวต์ <code>name</code> คือ ชื่อแอ็ตทริบิวต์ <code>default</code> คือ กำหนดข้อความแสดงผล
<code>hasattr()</code>	ใช้สำหรับตรวจสอบชื่อแอ็ตทริบิวต์มีอยู่ในออบเจ็คหรือไม่ถ้ามีคืนค่า <code>True</code> ถ้าไม่มีคืนค่า <code>False</code>	<code>hasattr(obj, name)</code>	<code>obj</code> คือ ออบเจ็คที่ถูกชื่อแอ็ตทริบิวต์ <code>name</code> คือ ชื่อแอ็ตทริบิวต์ที่ต้องการตรวจสอบมีอยู่ในออบเจ็คหรือไม่
<code>help()</code>	ใช้สำหรับกรณีที่ต้องการขอความช่วยเหลือวิธีเรียกใช้คำสั่งต่าง ๆ	<code>help(object)</code>	<code>obj</code> คือ ออบเจ็คที่ต้องการวิธีการใช้งาน
<code>map()</code>	ใช้สำหรับหาผลลัพธ์จากชนิดข้อมูลแบบเรียงลำดับจากฟังก์ชันที่กำหนด	<code>map(func, seq1, ... seq_n)</code>	<code>func</code> คือ ฟังก์ชันที่สร้างขึ้นมาหาผลลัพธ์ <code>seq1, ... seq_n</code> คือชนิดข้อมูลแบบเรียงลำดับ
<code>ord()</code>	ใช้สำหรับแปลงตัวอักษรเป็นค่ารหัส Unicode	<code>ord(s)</code>	<code>s</code> คือ ตัวอักษรที่ต้องการแปลงเป็นรหัส Unicode

ตาราง 7.1: Built-In ฟังก์ชันในภาษาไพธอนที่ใช้งานบ่อย (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน	คำอธิบายเพิ่มเติม
<code>range()</code>	ใช้สำหรับสร้าง กลุ่มข้อมูลตัวเลข จำนวนเต็ม	<code>range(start, stop [, step])</code>	<code>start</code> คือ ค่าเริ่มต้น <code>stop</code> คือ ค่าถัดจากค่า <sup>สุดท้ายใน range</sup> นั้น <code>step</code> คือ ค่าที่ให้เพิ่มขึ้น <sup>แต่ละชั้น</sup>
<code>repr()</code>	ใช้สำหรับแสดง ชนิดข้อมูลอักขระ <sup>หรือสตริงที่มี</sup> เครื่องหมาย <sup>Single Quote ( '...') ถ้าใช้ ฟังก์ชัน print() แสดงผล ถ้าไม่ใช้ จะมีเครื่องหมาย<sup>Double Quote ("...")</sup></sup>	<code>repr(obj)</code>	<code>obj</code> คือ ชนิดข้อมูลที่ ต้องการแสดงผล
<code>reversed()</code>	ใช้สำหรับแสดงชนิด ข้อมูลแบบเรียง <sup>ลำดับแบบย้อนกลับ</sup>	<code>reversed(seq)</code>	<code>seq</code> คือ ชนิดข้อมูลแบบ เรียงลำดับ
<code>round()</code>	ใช้สำหรับปัดเศษ จำนวนทศนิยม	<code>round(number [, ndigits])</code>	<code>number</code> คือตัวเลข ที่ต้องการปัดเศษ <code>ndigits</code> คือจำนวน ทศนิยมที่ต้องการ ถ้าไม่ กำหนดจะไม่มีทศนิยม
<code>slice()</code>	ใช้สำหรับแสดงค่า <sup>ข้อมูลตามช่วงที่ กำหนด</sup>	<code>slice(start, stop, step)</code>	<code>start</code> คือ จุดเริ่มต้นที่ ต้องการตัด <code>stop</code> คือ <sup>จุดสุดท้ายที่ต้องการตัด</sup> <code>step</code> คือ ค่าช่วงห่างใน <sup>การตัด</sup>

ตาราง 7.1: Built-In ฟังก์ชันในภาษาไพธอนที่ใช้งานบ่อย (มีต่อ)

เมธอด	ความหมาย	รูปแบบการใช้งาน	คำอธิบายเพิ่มเติม
<code>sorted()</code>	ใช้สำหรับจัดเรียง ข้อมูลในชนิดข้อมูล แบบเรียงลำดับ คืน ค่ากลับมาเป็นชนิด ข้อมูลลิสต์	<code>sorted(seq [, key][, reverse])</code>	<code>seq</code> คือ ชนิดข้อมูลแบบ เรียงลำดับ <code>key</code> คือ วิธี การจัดเรียงข้อมูลค่าปกติ เรียงจากน้อยสุดไปทาง มากสุด <code>reverse</code> คือ ถ้ากำหนดเป็น <code>True</code> จะเรียงจากค่ามากไปทาง น้อย
<code>vars()</code>	ใช้สำหรับดูค่าข้อมูล ที่ถูกเก็บไว้ในตัวแปร	<code>vars(obj)</code>	<code>obj</code> คือ ขอบเจ็ค โมดูล คลาสหรืออินสแตนซ์ (instance) ที่มีการเก็บ ค่าข้อมูลในตัวแปร
<code>zip()</code>	ใช้สำหรับรวมชนิด ข้อมูลแบบเรียง ลำดับเข้าด้วยกัน คืน ค่ากลับมาเป็นชนิด ข้อมูลทุกเพิล	<code>zip(*seq)</code>	<code>seq</code> คือ ชนิดข้อมูลแบบ เรียงลำดับ

ตาราง 7.1: Built-In พิงก์ชันในภาษาไพธอนที่ใช้งานบ่อย

ในลำดับถัดไปเป็นตัวอย่างการใช้งานบางพิงก์ชัน Built-in ของภาษาไพธอน เพื่อเป็นแนวทาง  
นำไปประยุกต์ใช้งาน สำหรับพิงก์ชันที่ผู้เขียนไม่ได้นำมาแสดงเป็นตัวอย่างการใช้งานนั้น เราสามารถ  
ทดสอบและทำความเข้าใจผลลัพธ์โดยใช้ตัวอย่างเป็นแนวทาง

### ตัวอย่าง 7.20

การใช้งานฟังก์ชัน `abs()`, `all()`, `any()`, `ascii()` และ `chr()`

```
1 x = -10.5
2 y = [1, 2, 3, True, False]
3 z = []
4 msg = 'Python programming'
5 print(abs(x))
6 print(all(y))
7 print(any(z))
8 print(ascii(msg))
9 print('ตัวอักษรรหัส 97 = ', chr(97))
```

10.5  
False  
False  
'Python programming'  
ตัวอักษรรหัส 97 = a



### ตัวอย่าง 7.21

การใช้งานฟังก์ชัน `sorted()`

```
1 x = [-10.51, 4.25, 100.0, -88.32]
2 y = ['Bush', 'Clinton', 'Obama', 'Kenedy']
3 print(x)
4 print(sorted(x))
5 print(sorted(x, reverse=True))
6 print(y)
7 print(sorted(y))
```

[-10.51, 4.25, 100.0, -88.32]  
[-88.32, -10.51, 4.25, 100.0]  
[100.0, 4.25, -10.51, -88.32]  
['Bush', 'Clinton', 'Obama', 'Kenedy']  
['Bush', 'Clinton', 'Kenedy', 'Obama']



## ตัวอย่าง 7.22

การใช้งานฟังก์ชัน `divmod()`, `eval()`, `ord()`, `reversed()` และ `zip()`

```
1 l = [1, 2, 3, 4, 5]
2 s = {'Python', 'Java', 'C', 'C++', 'R'}
3 sl = 'Python Programming'
4 div = divmod(9, 2)
5 exp = '20 + 60'
6 print('ค่าผลหารและเศษ = ', div)
7 print('exp =', eval(exp))
8 print('รหัส Unicode A =', ord('A'))
9 print(list(reversed(l)))
10 print('ค่าข้อมูลตำแหน่งที่ 1-10 ข้ามครั้งละ 2 ตำแหน่ง = ',
11      sl[slice(1, 10, 2)])
12 print(dict(zip(l, s)))
```

```
ค่าผลหารและเศษ = (4, 1)
exp = 80
รหัส Unicode A = 65
[5, 4, 3, 2, 1]
ค่าข้อมูลตำแหน่งที่ 1-10 ข้ามครั้งละ 2 ตำแหน่ง = yhnPo
{1: 'Python', 2: 'C++', 3: 'C', 4: 'Java', 5: 'R'}
```



## สรุปท้ายบท

ฟังก์ชันเป็นส่วนหนึ่งที่มีความสำคัญในการพัฒนาโปรแกรม ซึ่งในบทนี้ได้แนะนำวิธีการสร้างฟังก์ชันขึ้นมาใช้งานแบบต่าง ๆ เช่น การสร้างฟังก์ชันที่ไม่มีการส่งค่า การสร้างฟังก์ชันที่ไม่มีการคืนค่า การสร้างฟังก์ชันที่มีการรับค่าแต่ไม่มีการส่งค่ากลับ เป็นต้น นอกจากนี้ยังได้รู้จักกับวิธีการส่งค่าอาร์กิวเม้นต์ให้กับค่าพารามิเตอร์และยังได้รู้จักการวิธีการสร้างค่าพารามิเตอร์ไว้รอรับค่าอาร์กิวเม้นต์ รวมไปถึงสร้างฟังก์ชันที่ไม่ต้องกำหนดชื่อด้วยคำสั่ง `lambda` ทำให้ลดเวลาในการเขียนคำสั่งโปรแกรม อีกทั้งยังรู้จักกับฟังก์ชัน `Built-in` ของภาษาไพธอน ซึ่งเรียกใช้งานโดยไม่ต้องใช้คำสั่ง `import`

## แบบฝึกหัด

1. จงเขียนคำสั่งโปรแกรมคำนวนหาปริมาตรพีระมิด (Pyramid), ทรงกระบอก (Cylinder) และทรงกรวย (Cone) โดยใช้แยกการคำนวนแต่ละปริมาตรออกเป็นฟังก์ชัน โดยมีเมนูแสดงให้ผู้ใช้งานเลือกการหาปริมาตรที่ต้องการได้ กำหนดให้ป้อนข้อมูลผ่านทางคีย์บอร์ด
2. จงเขียนโปรแกรมแบบฟังก์ชันคำนวนอัตราแลกเปลี่ยนเงินไทย เป็นสกุลเงินในกลุ่มประเทศอาเซียน โดยมีเมนูแสดงให้ผู้ใช้งานเลือกสกุลเงินที่ต้องการเปลี่ยน และให้ผู้ใช้งานป้อนจำนวนเงินผ่านทางคีย์บอร์ด
3. จงเขียนโปรแกรมคำนวนหาค่าดัชนีมวลกาย โดยแยกการคำนวนออกเป็นฟังก์ชันระหว่างเพศชายและเพศหญิง กำหนดให้ผู้ใช้งานต้องป้อนข้อมูลผ่านทางคีย์บอร์ด

# บทที่ 8

## แนะนำการเขียนโปรแกรมเชิงวัตถุ

ในบทนี้เราจะได้ทำความรู้จักและวิธีการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming: OOP) ซึ่งเป็นรูปแบบการเขียนโปรแกรมได้รับความนิยมในปัจจุบัน แต่สำหรับผู้เริ่มฝึกเขียนโปรแกรมอาจจะมีความรู้สึกว่าทำความเข้าใจยาก เพราะรูปแบบการเขียนโปรแกรมมีความซับซ้อน ดังนั้นเราควรเริ่มศึกษาและทำความเข้าใจไปทีละขั้นตอน เนื่องจากมีคำศัพท์ที่ต้องทำความเข้าใจอยู่เป็นจำนวนมาก ถ้าหากเข้าใจหลักการเขียนโปรแกรมแบบ OOP แล้ว จะทำให้การพัฒนาโปรแกรมด้วยภาษาไพธอนที่มีความซับซ้อนกล้ายิ่งเรื่อยๆ

สำหรับในบทนี้เป็นเพียงการแนะนำให้รู้จักกับการเขียนโปรแกรมแบบ OOP เพ่านั้น และไม่ครอบคลุมเนื้อหาทั้งหมดของการเขียนโปรแกรมแบบ OOP เพราะมีรายละเอียดจำนวนมาก เนื้อหาในบางจุดความสามารถศึกษาและค้นคว้าเพิ่มเติมจากแหล่งอื่นที่อธิบายถึงรูปแบบการเขียนโปรแกรม OOP ด้วยภาษาไพธอน

### 8.1 ทำไมต้องเขียนโปรแกรมแบบ OOP

วิธีการเขียนโปรแกรมโครงสร้าง (Structure Programming) หรือการเขียนโปรแกรมแบบเชิงฟังก์ชัน (Function Programming) เป็นรูปแบบการเขียนโปรแกรมแบบดั้งเดิมที่มีการใช้งานมานาน ถึงแม้ว่าจะช่วยแก้ไขปัญหาได้ในระดับหนึ่งและเหมาะสมกับผู้ที่ต้องการเริ่มศึกษาและเป็นพื้นฐานที่สำคัญสำหรับการเขียนโปรแกรม แต่เมื่อเจอปัญหาที่มีความซับซ้อนมาก ฯ จะพบว่าวิธีการเขียนโปรแกรมทั้ง 2 รูปแบบ ไม่สามารถตอบสนองต่อการแก้ไขปัญหาได้ รวมไปถึงการแก้ไขปรับปรุงคำสั่งโปรแกรมในอนาคต ทำให้เกิดวิธีการเขียนคำสั่งโปรแกรมรูปแบบใหม่ขึ้นมาคือ การเขียนโปรแกรมแบบ OOP ทำให้ช่วยแก้ไขปัญหาต่าง ๆ ข้างต้นได้ นอกเหนือนี้ยังสามารถซ่อนช่วยให้เราประหยัดเวลาในการพัฒนาโปรแกรม รวมทั้งการบำรุงรักษาหรือการนำโปรแกรมกลับมาแก้ไขใหม่ และเพิ่มเติมส่วนต่าง ๆ ภายในโปรแกรมได้ง่ายขึ้น

## 8.2 เริ่มต้นเขียนโปรแกรมแบบ OOP ด้วยภาษาไฟรอน

การพัฒนาโปรแกรมรูปแบบนี้จะมองทุกอย่างเป็นวัตถุ (Object) ไม่ว่าจะเป็นการเขียนโปรแกรมเกี่ยวกับการคำนวณหรือการเขียนโปรแกรมให้แสดงรายละเอียดสิ่งของต่าง ๆ เช่น หนังสือ สัตว์ รถยนต์ มอเตอร์ไซค์ จักรยาน โดยแต่ละวัตถุก็จะมีแอ็ตทริบิวต์ (Attribute) เป็นการบ่งบอกถึงคุณสมบัติ (Property) ของวัตถุ และพฤติกรรม (Behavior) หรือเมธอด (Method) เป็นการบ่งบอกถึงวัตถุนั้น ทำอะไรได้บ้าง ทั้งสองอย่างนี้ถูกบรรจุในรูปแบบการเขียนโปรแกรม OOP เราจะได้ทำความรู้จักและวิธีการนำมาใช้งานในหัวข้อต่อไป

### 8.2.1 การสร้างคลาส

สำหรับการเขียนโปรแกรมแบบ OOP จะมีการออกแบบโครงสร้างของขอบเจ็คไว้ก่อนเสมอ เปรียบเสมือนการสร้างต้นแบบหรือพิมพ์เขียวขึ้นมา และมีการทำหนังองค์ประกอบคุณสมบัติและเมธอดต่าง ๆ ให้กับขอบเจ็คนั้น ๆ โดยต้นแบบที่สร้างขึ้นมาในการเขียนโปรแกรมแบบ OOP จะเรียกว่า class (คลาส) ซึ่งต้นแบบที่สร้างขึ้นมาสามารถที่จะนำไปดัดแปลง แก้ไข เพิ่มเติมในภายหลังได้ หรือกล่าวได้อีกอย่างว่าการสร้างคลาสก็เพื่อจัดกลุ่มของโปรแกรมเชิงวัตถุที่มีลักษณะเหมือนกันให้อยู่ในกลุ่มคลาสเดียวกัน รูปแบบการสร้าง class แสดงได้ดังนี้

```
class className: # คำสั่งประกาศสร้างคลาส
    statement_1 # คำสั่งโปรแกรมที่ 1 ภายในคลาส
    ...
    ...
    ...
    statement_n # คำสั่งโปรแกรมที่ n ภายในคลาส
```

ตามหลักการและโดยทั่วไปของการเขียนคำสั่งโปรแกรมแบบ OOP จะกำหนดให้ตัวอักษรตัวแรกของชื่อคลาสเป็นตัวพิมพ์ใหญ่เสมอ ยกตัวอย่างเช่น

```
class Book: # ประกาศสร้างคลาส Book
    statement(s)
```

```
class Dog: # ประกาศสร้างคลาส Dog
    statement(s)
```

```

class Student: # ประกาศสร้างคลาส Student
    statement(s)

class Calculate_area: # ประกาศสร้างคลาส Calculate_area
    statement(s)

```

## 8.2.2 การสร้างเมธอด

จุดประสงค์สำหรับการเมธอด (Method) ก็เพื่อให้ทำหน้าที่แสดงการกระทำอย่างใดอย่างหนึ่งของขอบ  
เจ็ค วิธีสร้างเมธอดขึ้นมาใช้งานก็มีหลากหลายรูปแบบตามจุดประสงค์ นอกเหนือนี้ยังมีแบบสำเร็จรูปที่  
ภาษาไพธอนเตรียมไว้ซึ่งจะนำมาใช้งานในคลาสได้เลย ต่อไปจะขอยกตัวอย่างการสร้างเมธอดแบบต่างๆ  
ที่นำขึ้นมาใช้งานในคลาสดังต่อไปนี้

### Instance Method

การสร้างเมธอดขึ้นมาใช้งานเหมือนกับลักษณะวิธีการสร้างฟังก์ชันโดยทั่วไป แต่จะมีค่าพารามิเตอร์ชื่อ **self** กำกับไว้เป็นตัวแรกเสมอ ซึ่งเป็นการอ้างอิงถึงตัวเองและแอ็ตทริบิวต์ตัวอื่น ๆ ที่อยู่ภายในเมธอด โดยจะถูกเรียกใช้งานผ่านทางอินสแตนซ์ (instance) ซึ่งจะกล่าวถึงในหัวข้อถัดไป หรือเรียกอีกอย่างว่า การสร้างขอบเจ็คขึ้นมาก่อน หลังจากนั้นจึงใช้อินสแตนซ์ที่สร้างขึ้นมาอ้างอิงถึงเมธอดภายในคลาส และส่งผ่านค่าอาร์กิวเมนต์ที่มีค่าพารามิเตอร์อยู่รับค่า หรือไม่มีก็ได้แต่ต้องมี Self ดังนี้

```

class Calculate_area: # ประกาศสร้างคลาส Calculate_area
    def cal_rectangle(self, w, h): # สร้างเมธอดแบบอินสแตนซ์
        → คำนวณพื้นที่สี่เหลี่ยมผืนผ้าและกำหนดพารามิเตอร์
        return w * h # คำลั่งคืนค่ากลับจากการคำนวณ

```

### Class Method

วิธีการสร้างเมธอดแบบนี้มีลักษณะเหมือนกับการสร้างเมธอดแบบอินสแตนซ์ แต่มีความแตกต่างกันตรง  
ที่ค่าพารามิเตอร์ตัวแรกเปลี่ยนจาก **self** เป็น **cls** แทน และมีการกำหนด **@classmethod** ไว้  
ก่อนการสร้างเมธอดขึ้นมาใช้งาน และวิธีการเรียกใช้งานเมธอดแบบคลาสนี้ไม่ต้องมีการสร้างอินสแตนซ์  
สามารถเรียกผ่านคลาสโดยตรงได้เลย

```

class Calculate_area: # ประกาศสร้างคลาส Calculate_area
    @classmethod
    def cal_triangle(cls, b, h): # สร้างเมธอดแบบคลาสคำนวณ
         $\rightarrow$  พื้นที่สามเหลี่ยมและกำหนดพารามิเตอร์
        return 0.5 * b * h # คำลั่งคืนค่ากลับจากการคำนวณ

```

### Static Method

เมธอดนี้มีการสร้างและการเรียกใช้งานแตกต่างจากห้องสองวิธีคือ “ไม่มีค่าพารามิเตอร์ตัวแรกกำกับไว้” แต่จะมีการกำหนด **@staticmethod** กำกับไว้ก่อนการสร้างเมธอดประเภทนี้ สำหรับเมธอดแบบสแตติกนี้จะมีการส่งผ่านค่าอาร์กิวเมนต์ไปให้ค่าพารามิเตอร์ได้โดยตรงเหมือนกับวิธีการสร้างฟังก์ชันแบบทั่วไปได้เลย และไม่มีการสร้างอินสแตนซ์สำหรับการอ้างอิงการเข้าถึงเมธอดแบบสแตติก

```

class Calculate_area: # ประกาศสร้างคลาส Calculate_area
    @staticmethod # กำหนดก่อนการสร้างเมธอดแบบสแตติก
    def cal_circle(r): # สร้างเมธอดสแตติกคำนวณพื้นที่วงกลมและ
         $\rightarrow$  กำหนดพารามิเตอร์
        return 3.14 * r * r # คำลั่งคืนค่ากลับจากการคำนวณ

```

### 8.2.3 การสร้างอินสแตนซ์

การเรียกใช้งานคลาสเป็นการเข้าถึงข้อมูลที่อยู่ภายในคลาส ได้แก่ แอ็ตทริบิวต์ และเมธอด ซึ่งเป็นการบ่งบอกว่ากำลังจะมีการสร้างออบเจ็คขึ้นมาใช้งาน ขอบเจ็คที่สร้างขึ้นมาจะมีชื่อเรียกวีกอย่างว่า อินสแตนซ์ (Instance) มีรูปแบบการสร้างอินสแตนซ์ดังต่อไปนี้

```
instance_name = class_name():
```

instance\_name ชื่ออินสแตนซ์ที่ต้องการสร้าง

class\_name ชื่อคลาสที่ต้องการเข้าถึง

## ตัวอย่าง 8.1

### รูปแบบการสร้างอินสแตนซ์

```
1 class Calculate_area: # ประกาศสร้างคลาส Calculate_area
2     def cal_rectangle(self, w, h):
3         return w * h
4     def cal_triangle(self, b, h):
5         return 0.5 * b * h
6
7 cal = Calculate_area() # สร้างอินสแตนซ์ของคลาส Calculae_area
```

หลังจากที่เราได้รู้จักกับคลาส เมื่อตอน และอินสแตนซ์ไปแล้ว เราสามารถนำทั้ง 3 ส่วนนี้มาเขียนเป็นโปรแกรมแบบ OOP เป็นต้นได้ โดยแสดงดังตัวอย่างต่อไปนี้ ซึ่งเป็นการสร้างโปรแกรมแบบ OOP คำนวนหาพื้นที่ โดยมี `Calculate_area` เป็นชื่อคลาส และประกอบด้วยเมื่อตอน `cal_rectangle` เป็นเมื่อตอนแบบอินสแตนซ์สังเกตได้จากค่าพารามิเตอร์ `self` ตัวแรกได้ถูกกำหนดไว้ เมื่อต้องการเรียกใช้งานต้องมีการสร้างอินสแตนซ์ขึ้นมาก่อน สำหรับเมื่อตอน `cal_triangle` เป็นเมื่อตอนแบบคลาสนี้องจากมีค่าพารามิเตอร์ `cls` ได้ถูกกำหนดไว้เป็นตัวแรก และมี `@classmethod` กำหนดไว้ และเมื่อตอนสุดท้ายเป็นเมื่อตอนสแตกติกซึ่งมี `@staticmethod` กำหนดไว้ การเรียกใช้งานทั้ง 3 เมื่อตอนแสดงดังตัวอย่าง 8.2

## ตัวอย่าง 8.2

การเขียนคำสั่งโปรแกรมแบบ OOP สำหรับคำนวณพื้นที่

```
1 # ประกาศสร้างคลาส Calculate_area
2 class Calculate_area:
3     # เมธอดแบบอินสแตนซ์
4     def cal_rectangle(self, w, h):
5         return w * h
6
7     # เมธอดแบบคลาส
8     @classmethod
9     def cal_triangle(cls, b, h):
10        return 0.5 * b * h
11
12    # เมธอดแบบสแตติก
13    @staticmethod
14    def cal_circle(r):
15        return 3.14 * r * r
16
17 # สร้างอินสแตนซ์คลาส Calculate_area
18 cal = Calculate_area()
19
20 # เรียกใช้เมธอด cal_rectangle พร้อมส่งค่าอาร์กิวเมนต์
21 cal_rec = cal.cal_rectangle(3, 5)
22
23 # เรียกใช้เมธอด cal_triangle พร้อมส่งค่าอาร์กิวเมนต์
24 cal_tri = Calculate_area.cal_triangle(6, 7)
25
26 # เรียกใช้เมธอด cal_circle พร้อมส่งค่าอาร์กิวเมนต์
27 cal_circle = Calculate_area.cal_circle(5)
28
29 # แสดงผลการคำนวณของเมธอด cal_rectangle
30 print('พื้นที่สี่เหลี่ยมผืนผ้า', cal_rec)
31
32 # แสดงผลการคำนวณของเมธอด cal_triangle
33 print('พื้นที่สามเหลี่ยม', cal_tri)
34
35 # แสดงผลการคำนวณของเมธอด cal_circle
36 print('พื้นที่วงกลม', cal_circle)
```

พื้นที่สี่เหลี่ยมผืนผ้า 15

พื้นที่สามเหลี่ยม 21.0

พื้นที่วงกลม 78.5



จากตัวอย่าง 8.2 เป็นการเขียนคำสั่งโปรแกรมแบบ OOP คำนวนพื้นที่ โดยนำเสนอตัวอย่างการใช้เมธอดแบบต่าง ๆ กัน

บรรทัดที่ 4-5

เป็นการสร้างเมธอดอินสแตนซ์ มีพารามิเตอร์ตัวแรกเป็น `self` เสมอ ส่วนพารามิเตอร์ที่เหลือเป็นพารามิเตอร์โดยรับค่าที่ถูกส่งเข้ามาประมวลผล

บรรทัดที่ 8-10

เป็นการสร้างเมธอดคลาส สังเกตได้จากมี `@classmethod` กำกับไว้ก่อนการสร้างเมธอด และมีพารามิเตอร์ตัวแรกเป็น `cls` เสมอ ส่วนพารามิเตอร์ที่เหลือเป็นพารามิเตอร์โดยรับค่าที่ถูกส่งเข้ามาประมวลผล

บรรทัดที่ 13-15

เป็นการสร้างเมธอดสเตรติกซึ่งจะมี `@staticmethod` กำกับไว้ และมีพารามิเตอร์โดยรับค่านำไปประมวลผลต่อเท่านั้น จะไม่มีพารามิเตอร์ `self` หรือ `cls` เมื่อกับเมธอด 2 แบบแรก

บรรทัดที่ 18

สร้างอินสแตนซ์ชื่อ `cal` จากคลาส `Calculate_area` เอาไว้ใช้อ้างอิงการเข้าถึงเมธอดแบบอินสแตนซ์

บรรทัดที่ 21

ส่งค่าข้อมูลให้กับเมธอดแบบอินสแตนซ์ และผลลัพธ์ที่ได้จะถูกเก็บไว้ที่ตัวแปร `cal_rec`

บรรทัดที่ 24

ส่งค่าข้อมูลให้กับเมธอดคลาส โดยการใช้ชื่อคลาสยังคงในการเข้าถึง และผลลัพธ์ที่ได้จากการประมวลผลจะถูกเก็บไว้ที่ตัวแปร `cal_tri`

บรรทัดที่ 27

ส่งค่าข้อมูลให้กับเมธอดแบบสเตรติก โดยการใช้ชื่อคลาสอ้างอิงในการเข้าถึง และผลลัพธ์ที่ได้จะถูกเก็บไว้ที่ตัวแปร `cal_circle`

บรรทัดที่ 30, 33, 36

แสดงผลลัพธ์ที่ได้จากการประมวลของเมธอดต่าง ๆ ที่ถูกส่งกลับคืนมา

### 8.3 การสร้างแอตทริบิวต์ และคอนสตรัคเตอร์

การสร้างแอตทริบิวต์ของคลาส (Class Attribute) ก็เปรียบเสมือนการสร้างตัวแปรชื่อนามาใช้งาน หรือกล่าวได้ว่าเป็นการกำหนดคุณสมบัติของวัตถุในเบื้องต้นที่จะถูกนำไปใช้ร่วมกับเมธอดอื่น ๆ ภายในคลาส โดยจะมีการกำหนดค่าให้กับแอตทริบิวต์หรือไม่มีการกำหนดค่าไว้ก็ได้ มีรูปแบบการสร้างดังแสดงตัวอย่าง โดยที่ `wide`, `high` และ `isbn` คือ แอตทริบิวต์ของคลาส `Book`

### ตัวอย่าง 8.3

แสดงข้อมูลในแอ็ตทริบิวต์ของคลาส

```
1 # ประกาศสร้างคลาส Book
2 class Book:
3     # สร้างแอ็ตทริบิวต์ของคลาสและกำหนดค่า
4     wide = 25
5     high = 30
6     page = 250
7
8     # สร้างเมธอดแบบอินสแตนซ์และพารามิเตอร์รับค่า
9     def showBook(self, name, isbn):
10         self.isbn = isbn
11         self.name = name
12
13         # คืนค่ากลับแสดงผลลัพธ์
14         return self.name, Book.wide, Book.high, Book.page, self.isbn
15
16
17     # สร้างอินสแตนซ์จากคลาส Book()
18 b_IT = Book()
19
20 # สร้างอินสแตนซ์จากคลาส Book()
21 b_Web = Book()
22
23 # แสดงผลลัพธ์จากการสร้างอินสแตนซ์อ้างอิงไปยังเมธอด showBook() พร้อมทั้งส่งค่าอากิวเมนต์
24 print(b_IT.showBook('Python Programming', '123-456-789-0'))
25 print(b_Web.showBook('Web Programming', '777-000-222-0'))
```

```
('Python Programming', 25,30,250, '123-456-789-0')
('Web Programming', 25,30,250, '777-000-222-0')
```



จากตัวอย่างโปรแกรมเราจะเห็นว่ามีการสร้างแอ็ตทริบิวต์ของคลาสขึ้นมาไว้ใช้งาน โดยปกติแล้วแอ็ตทริบิวต์ของคลาสจะถูกสร้างขึ้นมาและกำหนดค่าเบื้องต้นไว้โดยเพื่อความสะดวกในการนำไปใช้งานร่วมกับเมธอดอื่น ๆ ที่อยู่ภายในคลาส นอกจากนี้ภายในคำสั่งโปรแกรมมีการสร้างเมธอด `showBook()` สำหรับแสดงข้อมูลบางส่วนของหนังสือ โดยการนำเอาค่าแอ็ตทริบิวต์ของคลาスマแสดงผล และมีการสร้างพารามิเตอร์โดยรับค่าไว้อีกสองตัว

คอนสตรัคเตอร์ (Constructor) คือ เมธอดประเภทหนึ่งที่ถูกสร้างขึ้นมาไว้ใช้งาน และภายในเมธอดมีการสร้างแอ็ตทริบิวต์เพื่อเป็นการกำหนดข้อมูลเบื้องต้นของวัตถุหรืออินสแตนซ์ที่จะถูกสร้างขึ้น การสร้างคอนสตรัคเตอร์จะเรียกใช้งานเมธอด `__init__()` (เครื่องหมายที่อยู่ด้านหน้าและหลังคือ

Double underscore หรือเรียกอีกอย่างว่า dunder) เมื่อมีการสร้างคุณสตรัคเตอร์ขึ้นมาใช้งาน จะทำให้คุณสตรัคเตอร์มีการทำงานอัตโนมัติทุกครั้ง เมื่อมีการสร้างอินแหนช์จากคลาส คุณสตรัคเตอร์ที่ถูกสร้างขึ้นจะมีค่าพารามิเตอร์ไว้ค้อยรับค่าหรือไม่มีก็ได้ แต่ค่าพารามิเตอร์ที่ต้องมีเสมอคือ self และต้องกำหนดเป็นตัวแรก

#### การสร้างคุณสตรัคเตอร์แบบมีพารามิเตอร์รับค่า

```
class className:  
    # สร้างเมธอดคุณสตรัคเตอร์  
    def __init__(self, param1, param2,  
                 param3, ..., param_n)  
        self.param1 = param1  
        self.param2 = param2  
        self.param3 = param3  
        ...  
        self.param_n = param_n
```

#### การสร้างคุณสตรัคเตอร์แบบไม่มีพารามิเตอร์รับค่า

```
class className:  
    # สร้างเมธอดคุณสตรัคเตอร์  
    def __init__(self)  
        self.attribute1 = ''  
        self.attribute2 = ''  
        self.attribute3 = ''  
        ...  
        self.attribute_n = ''
```

#### ตัวอย่าง 8.4

การสร้างคอนสตรัคเตอร์แบบมีค่าพารามิเตอร์โดยรับค่า

```
1 # ประกาศสร้างคลาส Book
2 class Book:
3     # สร้างคอนสตรัคเตอร์แบบมีการรับค่าพารามิเตอร์
4     def __init__(self, name, wide, high, page, isbn, price):
5         # สร้างแอตทริบิวต์
6         self.name = name
7         self.wide = wide
8         self.high = high
9         self.page = page
10        self.isbn = isbn
11        self.price = price
12
13    def showBook(self): # สร้างเมетодแบบอินสแตนซ์คืนค่ากลับแสดงผลข้อมูล
14        return self.name, self.wide, self.high, self.page, self.isbn,
15        ↵ self.price
16
17    # สร้างอินสแตนซ์ b_IT พร้อมส่งค่า
18    b_IT = Book('Python Programming', 25, 30, 350, '123-456-789-0', 300)
19    print(b_IT.showBook()) # แสดงผลลัพธ์จากอินสแตนซ์ b_IT
20    b_IT.price = 350 # กำหนดราคาหนังสือให้กับอินสแตนซ์ b_IT ใหม่
21    print(b_IT.showBook()) # แสดงผลลัพธ์จากอินสแตนซ์ b_IT อีกครั้ง
22    # สร้างอินสแตนซ์ b_Web พร้อมส่งค่า
23    b_Web = Book('Web Programming', 20, 25, 300, '777-000-222-0', 375)
24    print(b_Web.showBook()) # แสดงผลลัพธ์จากอินสแตนซ์ b_Web'
```

```
('Python Programming', 25,30,350, '123-456-789-0', 300)
('Python Programming', 25,30,350, '123-456-789-0', 350)
('Web Programming', 20,25,300,'777-000-222-0',375)
```



จากตัวอย่าง 8.4 เป็นการเขียนคำสั่งโปรแกรมสร้างคอนสตรัคเตอร์แบบมีค่าพารามิเตอร์โดยรับค่าข้อมูล เมื่อนำค่าพารามิเตอร์ไปสร้างแอตทริบิวต์ในบรรทัดที่ 6-11 จะมี **self** นำหน้าทุกค่าพารามิเตอร์ ก่อนไปใช้งาน เมื่อเรียกใช้คลาสทำให้เมетод **\_\_init\_\_()** ทำงานอัตโนมัติ ทำให้เราส่งข้อมูลเข้าไปยังคลาสได้เลยในขณะที่กำลังมีการสร้างอินสแตนซ์ในบรรทัดที่ 18 และเมื่อต้องการนำค่าข้อมูลที่ถูกเก็บอยู่ในแอตทริบิวต์ตัวใดมาแสดงผล หรือต้องการเปลี่ยนค่าข้อมูลของแอตทริบิวต์ตัวใด ให้ใช้อินสแตนซ์ที่สร้างขึ้นมาตามด้วยเครื่องหมาย **(.)** และตามด้วยชื่อแอตทริบิวต์ที่ต้องการนำมาแสดงผลหรือต้องการเปลี่ยนค่าข้อมูล แสดงตัวอย่างในบรรทัดที่ 20 เป็นการเปลี่ยนราคานั้นสือให้กับอินสแตนซ์ **b\_IT** ใหม่ เห็นได้จากผลลัพธ์ที่ถูกแสดงออกมา

## ตัวอย่าง 8.5

การสร้างคลาสสตรัคเตอร์แบบไม่มีค่าพารามิเตอร์โดยรับค่า

```
1 # ประกาศสร้างคลาส Book
2 class Book:
3     # สร้างคลาสสตรัคเตอร์แบบไม่มีค่าพารามิเตอร์
4     def __init__(self):
5         self.name = ''
6         self.wide = ''
7         self.high = ''
8         self.page = ''
9         self.isbn = ''
10        self.price = ''
11
12    # สร้างเมธอดแบบอินสแตนซ์คืนค่ากลับแสดงผลข้อมูล
13    def showBook(self):
14        return self.name, self.wide, self.high, self.page, self.isbn,
15                  self.price
16
17    b_Linux = Book() # สร้างอินสแตนซ์ b_Linux
18    # กำหนดค่าข้อมูลให้แต่ละแอ็ตทริบิวต์ของอินสแตนซ์ b_IT
19    b_Linux.name = 'Linux Programming'
20    b_Linux.wide = 25
21    b_Linux.high = 30
22    b_Linux.page = 350
23    b_Linux.isbn = '444-454-799-0'
24    b_Linux.price = 300
25
26    b_DB = Book() # สร้างอินสแตนซ์ b_DB
27    # กำหนดค่าข้อมูลให้แต่ละแอ็ตทริบิวต์ของอินสแตนซ์ b_DB
28    b_DB.name = 'Database Programming'
29    b_DB.wide = 30
30    b_DB.high = 35
31    b_DB.page = 377
32    b_DB.isbn = '444-333-222-0'
33    b_DB.price = 350
34
35    # แสดงผลลัพธ์จากอินสแตนซ์ b_Linux และ b_DB โดยอ้างอิงเมธอดแบบอินสแตนซ์ showBook()
36    print(b_Linux.showBook())
37    print(b_DB.showBook())
```

```
('Linux Python Programming', 25,30,350, '444-454-799-0', 300)
('Database Programming', 30,35,377, '444-333-222-0', 350)
```



จากตัวอย่างที่ 8.5 เป็นการสร้างคอนสตรัคเตอร์ที่ไม่มีพารามิเตอร์โดยรับค่า แต่จะมีเฉพาะ[แอดทริบิวต์](#)ที่สร้างขึ้นมาไว้รอรับค่าที่จะถูกกำหนดจากการสร้างอินสแตนซ์ไว้ 2 ตัว ได้แก่ `b_IT` และ `b_DB` แสดงในบรรทัดที่ 17 และ 26 เมื่อต้องการกำหนดค่า[แอดทริบิวต์](#)ให้เราใช้ชื่ออินสแตนซ์ที่สร้างขึ้นมาตามด้วยเครื่องหมาย ([.](#)) และตามด้วยชื่อ[แอดทริบิวต์](#)ที่ต้องการกำหนดค่า สำหรับการนำข้อมูลออก มาแสดงผลก็ให้อ้างถึงเมธอดคลาสผ่านทางอินสแตนซ์แต่ละตัว

## 8.4 การสืบทอด

การสืบทอด (Inheritance) เป็นหลักการสำคัญในการเขียนโปรแกรมเชิงวัตถุ กล่าวคือ คุณสมบัติต่าง ๆ (แอดทริบิวต์) ที่สร้างขึ้นภายในคลาสสามารถที่จะถูกสืบทอด (inherit) ไปยังคลาสอื่น ๆ ได้ เมื่อสร้างคลาสใหม่ขึ้นมาใช้งาน และภายในคลาสใหม่อาจจะมี[แอดทริบิวต์](#)หรือเมธอดที่เหมือนกับคลาสที่สร้างขึ้นมาใช้งานก่อนหน้านี้ หากต้องสร้าง[แอดทริบิวต์](#)หรือเมธอดขึ้นมาใหม่อีก็จะทำให้เสียเวลาในการเขียนคำสั่งโปรแกรมและเกิดความซ้ำซ้อน การสืบทอดจากคลาสนี้ไปยังอีกคลาสนึงซึ่ง่วยทำให้เราประหยัดเวลาในการเขียนคำสั่งโปรแกรม เพราะไม่จำเป็นต้องสร้าง[แอดทริบิวต์](#)หรือเมธอดที่มีอยู่ในคลาสเดิม คลาสขึ้นมาอีก โดยคลาสที่ถูกสืบทอดเราระบุว่า คลาสแม่ (Superclass หรือ Parent Class) สำหรับคลาสที่ได้รับสืบทอดเรียกว่า คลาสลูก (Subclass) นอกจากนี้ยังสามารถสืบทอดได้หลายคลาส

รูปแบบการสร้างคลาสลูกให้สืบทอดจากคลาสแม่

```
class Superclass: # สร้างคลาสแม่
    Statement(s)

class Subclass(Superclass): # สร้างคลาสลูกสืบทอดจากคลาสแม่
    Statement(s)
```

## ตัวอย่าง 8.6

ลักษณะการสืบทอดคลาส

```
1 class Book: # ประกาศสร้างคลาส Book เป็นคลาสมี  
2  
3     def __init__(self, name, size, page, price): # คอนสตรัคเตอร์กำหนด  
4         → แอ็ตทริบิวต์  
5         self.name = name  
6         self.size = size  
7         self.page = page  
8         self.price = price  
9  
10    def showBook(self): # สร้างเมธอดแบบอินสแตนซ์สำหรับแสดงผลข้อมูล  
11        return self.name, self.size, self.page, self.price  
12  
13    class Book_IT(Book): # สร้างคลาส Book_IT เป็นคลาสสูกสืบทอดมาจากคลาส Book  
14        pass  
15  
16  
17    # กำหนดข้อมูลให้กับอินสแตนซ์คลาสมี  
18    book = Book('Python Programming', '25 x 30', 275, 300)  
19    # กำหนดข้อมูลให้กับอินสแตนซ์คลาสสูก  
20    b_IT = Book_IT('Information Technology', '25 x 32', 390, 250)  
21    # แสดงผลข้อมูลอ้างอิงจากอินสแตนซ์คลาสมีเปลี่ยงเมธอด showBook() ของคลาสมี  
22    print(book.showBook())  
23    # แสดงผลข้อมูลอ้างอิงจากอินสแตนซ์คลาสสูกไปยังเมธอด showBook() ของคลาสมี  
24    print(b_IT.showBook())
```

```
('Python Programming', 25 x 30, 275, 300,  
('Information Technology', 25 x 32, 390, 250)
```



จากตัวอย่าง 8.6 เป็นการสาธิตการเขียนคำสั่งโปรแกรมให้มีการสืบทอดจากคลาส Book ซึ่งเป็นคลาสมี ไปยังคลาส Book\_IT ซึ่งเป็นคลาสสูก โดยที่ภายในคลาส Book\_IT มีเฉพาะคำสั่ง **pass** เท่านั้น เมื่อสร้างอินสแตนซ์ในบรรทัดที่ 20 ขึ้นมาแล้วเราระบุที่จะเข้าถึงคอนสตรัคเตอร์ของคลาสมีได้เลย และเรียกใช้งานเมธอดที่มีอยู่ในคลาสมีได้

## ตัวอย่าง 8.7

การสืบทอดคุณสมบัติคลาสแม่ไปยังคลาสลูก

```
1 class Book:
2     def __init__(self, name, size, page, price):
3         self.name = name
4         self.size = size
5         self.page = page
6         self.price = price
7
8     def showBook(self):
9         return self.name, self.size, self.page, self.price
10
11
12 class Book_IT(Book):
13     def __init__(self, name, size, page, price, isbn, publisher):
14         super().__init__(name, size, page, price)
15         self.isbn = isbn
16         self.publisher = publisher
17
18     def showBook_IT(self):
19         return super().showBook(), self.isbn, self.publisher
20
21 book = Book('Python Programming', '25 x 30', 275, 300)
22 b_IT = Book_IT('Information Technology', '25 x 32', 390, 250,
23   ↪ '123-456-789-0', 'ABC')
24 print(book.showBook())
25 print(b_IT.showBook_IT())'
```

```
('Python Programming', '25 x 30', 275, 300)
 (('Information Technology', '25 x 32', 390 ,250), '123-456-789-0',
 ↪ 'ABC')
```



จากตัวอย่าง 8.7 ได้เขียนคำสั่งโปรแกรมเพิ่มเติมภายในคลาส Book\_IT ซึ่งเป็นคลาสลูกสืบทอดมา  
จากคลาส Book ซึ่งเป็นคลาสแม่

- บรรทัดที่ 13 สร้างคอนสตรัคเตอร์เก็บแอตทริบิวต์ของคลาสลูกที่สร้างขึ้นมา และมีการสืบทอดคุณสมบัติมาจากคลาสแม่ ภายในคอนสตรัคเตอร์ของคลาส Book\_IT มีพารามิเตอร์เพิ่มเข้ามาอีก 2 ตัวคือ isbn และ publisher ส่วนที่เหลือคือพารามิเตอร์ที่ต้องสืบทอดมาจากคลาสแม่
- บรรทัดที่ 14 ใช้คำสั่ง `super().__init__(parameter(s))` สร้างแอตทริบิวต์ที่สืบทอดมาจากคลาสแม่ จากนั้นสร้างแอตทริบิวต์ที่เพิ่มเข้ามาในคลาสลูก
- บรรทัดที่ 18 เป็นการสร้างเมธอด showBook\_IT() ซึ่งเป็นเมธอดแบบอินสแตนซ์ สำหรับส่งค่ากลับมาแสดงผลเมื่อมีการเรียกใช้งาน
- บรรทัดที่ 19 ใช้คำสั่ง `super().method()` ที่ต้องการสืบทอด จากตัวอย่างคือเมธอด showBook() และเพิ่มแอตทริบิวต์ส่วนที่เหลือ
- บรรทัดที่ 24 เปลี่ยนจากการอ้างอิงผ่านอินสแตนซ์จากเมธอด showBook() ซึ่งเป็นของคลาสแม่ เป็นเมธอด showBook\_IT() ซึ่งเป็นของคลาสลูกแทน

### ตัวอย่าง 8.8

#### การสืบทอดคุณสมบัติคลาสแม่ไปยังคลาสลูก

```

1  class Book:
2      def __init__(self, name, size, page, price):
3          self.name = name
4          self.size = size
5          self.page = page
6          self.price = price
7
8      def showBook(self):
9          return self.name, self.size, self.page, self.price
10
11
12 class Book_IT(Book):
13     def __init__(self, name, size, page, price, isbn, publisher):
14         super().__init__(name, size, page, price)
15         self.isbn = isbn
16         self.publisher = publisher
17

```

```

18     def showBook_IT(self):
19         return super().showBook(), self.isbn, self.publisher
20
21
22 class BookType(Book_IT):
23     def __init__(self, name, size, page, price, isbn, type,
24                  publisher):
25         super().__init__(name, size, page, price, isbn, publisher)
26         self.type = type
27
28     def showBookType(self):
29         return super().showBook_IT(), self.type
30
31 book = Book('Python Programming', '25 x 30', 275, 300)
32 b_IT = Book_IT('Information Technology', '25 x 32', 390, 250,
33                 '123-456-789-0', 'ABC')
34 b_IT1 = BookType('Information Technology', '25 x 32', 390, 250,
35                   '123-456-789-0', 'Computer', 'ABC')
36 print(book.showBook())
37 print(b_IT.showBook_IT())
38 print(b_IT1.showBookType())

```

```

('Python Programming', '25 x 30', 275, 300)
(('Information Technology', '25 x 32', 390, 250), '123-456-789-0',
 ↪ 'ABC')
((('Information Technology', '25 x 32', 390, 250), '123-456-789-0',
 ↪ 'ABC'), 'Computer')

```

จากตัวอย่าง 8.8 ได้สร้างคลาสลูกขึ้นมาอีก 1 คลาสคือ BookType และเพิ่มแอ็ตทริบิวต์อีก 1 ตัวคือ `type` เพื่อบอกว่าหนังสือเล่มนี้เป็นหนังสือประเภทอะไร โดยคลาสนี้จะสืบทอดคุณสมบัติมาจากคลาส Book\_IT ซึ่งเป็นคลาスマ่ำ สำหรับการเพิ่มค่อนสตรัคเตอร์จะเหมือนกับตัวอย่าง 8.7 ส่วนผลลัพธ์ให้เพิ่มข้อมูลดังตัวอย่างในบรรทัดที่ 21 และแสดงผลลัพธ์โดยอ้างอิงจากอินสแตนซ์ที่ชื่อ `b_IT1` ไปยังเมธอด `showBookType()` ตามตัวอย่างในบรรทัดที่ 25

## 8.5 เมธอด `__str__()` และ `__repr__()`

การจัดรูปแบบการแสดงผลข้อมูลก็เป็นอีกส่วนหนึ่งที่มีความสำคัญ ภาษาไพธอนได้จัดเตรียมเมธอดที่ใช้สำหรับแสดงผลลัพธ์ที่คืนค่าจากคลาสที่ถูกอินสแตนซ์เรียกใช้งานในบางกรณีผลลัพธ์ที่แสดงออกมาจะกลยุบเป็นคำແນงข้อมูลที่อยู่ในหน่วยความจำแทน แต่เราสามารถใช้เมธอด `__str__()` และ `__repr__()` เข้ามาย่วยจัดรูปแบบการแสดงผลข้อมูลได้ แต่ทั้งสองเมธอดนี้มีผลลัพธ์ที่แสดงออกมากต่างกันดังตัวอย่างต่อไปนี้

### ตัวอย่าง 8.9

การใช้เมธอด `__str__()` แสดงผลลัพธ์

```
1 class StudentTest:
2     def __init__(self, name, score1, score2, score3):
3         self.name = name
4         self.score1 = score1
5         self.score2 = score2
6         self.score3 = score3
7
8     def sumScore(self): # เมธอดแบบอินสแตนซ์
9         return self.score1 + self.score2 + self.score3
10
11    def __str__(self): # เมธอดคืนค่ากลับแสดงผลข้อมูล
12        return f'Name: {self.name}, Total of score: {self.sumScore()}'
```

```
Jantra 80
Name: Janis, Total of score: 80
```



จากตัวอย่าง 8.9 ในบรรทัดที่ 11 แสดงการใช้เมธอด `__str__()` และมีการจัดรูปแบบข้อมูล และทำการคืนค่ากลับในบรรทัดที่ 12 ด้วยคำสั่ง `return` โดยมีการใช้ f-string เข้าช่วยในการจัดรูปแบบการแสดงผล เมื่อสังเกตคำสั่งแสดงผลลัพธ์ (`print`) ในบรรทัดที่ 16 และ 17 จะมีวิธีการเขียนคำสั่งที่แตกต่างกัน โดยในบรรทัดที่ 15 จะใช้อินสแตนซ์ในการเข้าถึงแอตทริบิวต์ที่อยู่ในคลาสถึงจะนำข้อมูลกลับมาแสดงผลลัพธ์ได้ แต่ในบรรทัดที่ 16 จะใช้เพียงอินสแตนซ์ที่สร้างขึ้นมาเท่านั้นก็สามารถแสดงผลลัพธ์ได้แล้ว เนื่องจากเมธอด `__str__()` เป็นตัวส่งข้อมูลกลับมาให้

### ตัวอย่าง 8.10

การใช้เมธอด `__repr__()` แสดงผลลัพธ์

```
1 class StudentTest:
2     def __init__(self, name, score1, score2, score3):
3         self.name = name
4         self.score1 = score1
5         self.score2 = score2
6         self.score3 = score3
7
8     def sumScore(self):
9         return self.score1 + self.score2 + self.score3
10
11    def __repr__(self): # แสดงคืนค่ากลับและข้อมูล
12        return repr((self.name, self.sumScore()))
13
14
15 std1 = StudentTest('Janis', 20, 35, 25)
16 print(std1.name, std1.sumScore())
17 print(std1)
```

Janis 80  
('Janis', 80)



เมื่อใช้เมธอด `__repr__()` คืนค่ากลับมาแสดงผล ตามตัวอย่างบรรทัดที่ 11-12 ผลลัพธ์ที่ออกมานะจะเป็นชนิดข้อมูล และเมื่อมีชนิดข้อมูลสตริงปรากฏอยู่ภายในจะมีเครื่องหมาย (' ') ครอบไว้ ถึงแม้ว่าจะเป็นชนิดข้อมูลสตริงก็ตาม อย่างไรก็ตามความสามารถปรับเปลี่ยนวิธีการแสดงผลโดยใช้เมธอด `__repr__()` ได้หลากหลายรูปแบบ ให้เราทดลองแก้ไขคำสั่งรูปแบบการคืนค่ากลับมาแสดงผล และทดสอบการทำงานคำสั่งของโปรแกรมหลาย ๆ แบบ เพื่อให้เกิดความเข้าใจการทำงานของทั้งเมธอดและนำไปใช้งานในการเขียนโปรแกรมแบบ OOP

## 8.6 การโอเวอร์โหลดตัวดำเนินการ

เมื่อสร้างอินสแตนซ์ (วัตถุ) ขึ้นมาจากการคลาสใด ๆ แล้ว และเมื่อต้องการดำเนินการอย่างใดอย่างหนึ่งระหว่างอินสแตนซ์ เช่น การบวก การลบ การหาร เป็นต้น เครื่องหมายตัวดำเนินการต่าง ๆ จะไม่สามารถนำมาใช้งานได้ตรง ๆ กับอินสแตนซ์ แต่มีวิธีการดำเนินการระหว่างอินสแตนซ์ด้วยเครื่องหมาย เช่น บวก ลบ คูณ หาร เป็นต้น เหล่านี้จะทำได้ด้วยการทำโอเวอร์โหลดตัวดำเนินการ (Operator Overloading) โดยเรียกใช้งานเมธอดพิเศษ เช่น เมธอด `__add__()` ใช้สำหรับโอเวอร์โหลดเครื่องหมายบวก

เมธอด `__sub__()` ใช้สำหรับໂອເວອຣີໂຫລດເຄື່ອງໝາຍລບ ແລະ ມີມູນຄົນ ເປັນຕົ້ນ  
ເຄື່ອງໝາຍຄູນ ເປັນຕົ້ນ

### ຕ້ວອຍ່າງ 8.11

ການໂອເວອຣີໂຫລດຕົວດໍາເນີນການບວກ

```
1 class StudentTest:
2     def __init__(self, score):
3         self.score = score
4
5     def __add__(self, other): # ແມ່ນໂອເວອຣີໂຫລດຕົວດໍາເນີນການບວກ
6         result = self.score + other.score
7         return StudentTest(result)
8
9     def __str__(self): # ແມ່ນຈັດຮູບແບບໜິດຂໍ້ອມຸລສຕຣີງ
10        return 'Total of score = {}'.format(self.score)
11
12
13 score1 = StudentTest(20)
14 score2 = StudentTest(30)
15 score3 = StudentTest(25)
16 sumScore = score1 + score2 + score3
17 print(sumScore)
```

Total of score = 75



ຈາກຕ້ວອຍ່າງ 8.11 ເມື່ອຕ້ອງການບວກຂະແນນຂອງແຕ່ລະອິນສແຕນໆ ຈະມີການສ້າງມີມູນຄົນ `__add__()` ໃນບຣທັດທີ 5 ຈຶ່ນມາທໍາການໂອເວອຣີໂຫລດເຄື່ອງໝາຍບວກ ສຳຮັບມີມູນຄົນ `__str__()` ເປັນມີມູນຄົນທີ່ໃຊ້  
ສຳຮັບຈັດຮູບແບບໜິດຂໍ້ອມຸລສຕຣີງ ທັງນີ້ແມ່ນມີການສ້າງມີມູນຄົນນີ້ຈຶ່ນມາ ພລລັພົກທີ່ໄດ້ຈະເປັນຄ່າຕໍ່ແນ່ງຂໍ້ອມຸລທີ່  
ເກີບອູ້ໃນໜ່ວຍຄວາມຈຳແທນ

## ตัวอย่าง 8.12

การโอเวอร์โหลดตัวดำเนินการเปรียบเทียบ

```
1  class Compare:
2      def __init__(self, x):
3          self.x = x
4
5      def __ge__(self, other): # เมธอดโอเวอร์โหลดเครื่องหมายมากกว่าหรือเท่ากับ
6          if (self.x >= other.x):
7              text = 'num1 is greater than or equal num2'
8          else:
9              text = 'num1 is less than num2'
10         return text
11
12     def __ne__(self, other): # เมธอดโอเวอร์โหลดเครื่องหมายไม่เท่ากับ
13         if (self.x != other.x):
14             text = 'Both are not equal'
15         else:
16             text = 'Both are equal'
17         return text
18
19
20     num1 = Compare(200)
21     num2 = Compare(450)
22     result_n = num1 >= num2
23     print('The result of comparison is = ', result_n)
24
25     str1 = Compare('Python')
26     str2 = Compare('C++')
27     result_s = str1 == str2
28     print('The result of comparison is = ', result_s)
```

The result of comparison is = num1 is less than num2  
The result of comparison is = False



ยังมีเมธอดอื่น ๆ อีกจำนวนมากที่นำมาใช้สำหรับโอเวอร์โหลดตัวดำเนินการและเครื่องหมายต่าง ๆ ในภาษาไพธอน จึงขอยกตัวอย่างเมธอดที่น่าจะได้นำมาใช้งานบ่อย ๆ ในการเขียนโปรแกรมแบบ OOP ดังแสดงในตารางต่อไปนี้

สัญลักษณ์	ชื่อเมธอด
+	<code>__add__(self, other)</code>
-	<code>__sub__(self, other)</code>
*	<code>__mul__(self, other)</code>
/	<code>__truediv__(self, other)</code>
//	<code>__floordiv__(self, other)</code>
%	<code>__mod__(self, other)</code>
**	<code>__pow__(self, other)</code>

ตาราง 8.1: เมธอดโอลเวอร์ไอลด์ตัวดำเนินการ

สัญลักษณ์	ชื่อเมธอด
<	<code>__lt__(self, other)</code>
>	<code>__gt__(self, other)</code>
<=	<code>__le__(self, other)</code>
>=	<code>__ge__(self, other)</code>
==	<code>__eq__(self, other)</code>
!=	<code>__ne__(self, other)</code>

ตาราง 8.2: เมธอดโอลเวอร์ไอลด์ตัวดำเนินการเปรียบเทียบ

สัญลักษณ์	ชื่อเมธอด
-=	<code>__isub__(self, other)</code>
+=	<code>__iadd__(self, other)</code>
*=	<code>__imul__(self, other)</code>
**=	<code>__ipow__(self, other)</code>
/=	<code>__idiv__(self, other)</code>
//=	<code>__ifloordiv__(self, other)</code>
%=	<code>__imod__(self, other)</code>

ตาราง 8.3: เมธอดโอลเวอร์ไอลด์การกำหนดค่า

## 8.7 การโอเวอร์โหลดดึงเมธอด

เมื่อคลาสลูกสืบทอดมาจากคลาสมแม่ ทำให้คลาஸลูกมีคุณสมบัติและมีพฤติกรรมเหมือนกับคลาสมแม่ทุกประการ แต่ในบางกรณีคลาஸลูกอาจจะมีพฤติกรรมที่ต่างไปจากคลาสมแม่ได้ การโอเวอร์โหลดดึงเมธอด (Method Overloading) เป็นการกำหนดพฤติกรรมของคลาஸลูกให้ต่างจากคลาสมแม่ และมีชื่อเมธอดที่เหมือนกัน ยกตัวอย่างเช่น เราสร้างคลาสโทรศัพท์มือถือแล้วกำหนดให้เป็นคลาสมแม่ และมีเมธอดเปิด-ปิดหน้าจอด้วยวิธีการสไลเดอร์หน้าจอ และสร้างคลาஸลูกขึ้นมาสืบทอดและมีชื่อเมธอดที่เหมือนกันกับคลาสมแม่ แต่วิธีการเปิด-ปิดหน้าจอที่ต่างกัน เช่น สแกนลายนิ้วมือ สแกนใบหน้า ป้อนรหัสก่อนเข้าใช้งานโทรศัพท์ เป็นต้น

ตัวอย่าง 8.13

การโอเวอร์โหลดดึงเมธอด

```
1 class smartPhone: # สร้างคลาสมแม่
2     def __init__(self):
3         self.brand = 'iPhone'
4
5     def openSmartPhone(self):
6         return 'Access smartPhone with Passcode'
7
8     def __str__(self):
9         return f'{self.brand}: {self.openSmartPhone()}'
10
11
12 class smartPhoneOne(smartPhone): # สร้างคลาஸลูกที่ 1
13     def __init__(self):
14         super().__init__()
15
16     def openSmartPhone(self):
17         return 'Access smartPhone with Passcode or FingerID'
18
19
20 class smartPhoneTwo(smartPhoneOne): # สร้างคลาஸลูกที่ 2 สืบทอดคุณสมบัติมาจากคลาส
→   → ลูกที่ 1
21     def __init__(self):
22         super().__init__()
23
24     def openSmartPhone(self):
25         return 'Access smartPhone with Passcode or FaceID'
26
```

```

27
28 a = smartPhone() # สร้างอินสแตนซ์คลาสแม่
29 b = smartPhoneOne() # สร้างอินสแตนซ์คลาสลูกที่ 1
30 c = smartPhoneTwo() # สร้างอินสแตนซ์คลาสลูกที่ 2
31 b.brand = 'iPhone 5' # กำหนดค่าแอ็ตทริบิวต์แสดงผลคลาสลูกที่ 1
32 c.brand = 'iPhone X' # กำหนดค่าแอ็ตทริบิวต์แสดงผลคลาสลูกที่ 2
33 print(a)
34 print(b)
35 print(c)

```

iPhone: Access smartPhone with Passcode  
 iPhone 5: Access smartPhone with Passcode or FingerID  
 iPhone X: Access smartPhone with Passcode or FaceID



การทำโอเวอร์โหลดดึงเมธอดนี้ทำให้เราสามารถเปลี่ยนพฤติกรรม (เมธอด) ของคลาสลูกที่สืบทอดมาจากคลาสแม่ได้ เราจะสังเกตเห็นได้จากชื่อเมธอดคลาสแม่และคลาสลูกทั้งสองคลาสที่สืบทอดต่อกันมาขึ้นเมธอดเดียวกัน แต่มีข้อมูลที่แสดงผลแตกต่างกัน นอกจากนี้ความสามารถเปลี่ยนแปลงข้อมูลแอ็ตทริบิวต์ที่อยู่ในคลาสแม่ให้แสดงผลที่คลาสลูกต่างจากคลาสแม่ได้ เห็นได้จากบรรทัดที่ 31-32 มีการกำหนดข้อมูลให้กับแอ็ตทริบิวต์ **brand**

## 8.8 การห่อหุ้ม/ซ่อนข้อมูล

การเขียนโปรแกรมแบบ OOP เราสามารถกำหนดให้แอ็ตทริบิวต์หรือเมธอดได ๆ ในคลาสสามารถถูกเรียกใช้งานหรือเปลี่ยนแปลงแก้ไขข้อมูลจากการอ้างอิงผ่านอินสแตนซ์ได ซึ่งเป็นการป้องกันการเข้าถึงแอ็ตทริบิวต์หรือเมธอดที่มีการกำหนดค่าไว้เรียบร้อยแล้ว ในบางครั้งเราอาจจะไปเปลี่ยนแปลงค่าข้อมูล ตัวแปรโดยไม่ได้ตั้งใจ การห่อหุ้ม/หรือการซ่อนข้อมูล (Data Encapsulation) จะทำให้ค่าข้อมูลที่อยู่ภายในคลาสมีความปลอดภัยมากขึ้น เมื่อต้องการป้องกันไม่ให้เข้าถึงแอ็ตทริบิวต์หรือเมธอดได ๆ ภายในคลาส ให้ใช้เครื่องหมาย \_\_ (Double Underscore) นำหน้าชื่อแอ็ตทริบิวต์หรือเมธอนั้น ๆ แสดงถึงตัวอย่างต่อไปนี้

### ตัวอย่าง 8.14

การห่อข้อมูลและเมรอดภายในคลาส

```
1 class Student: # ประกาศสร้างคลาส Book
2     def __init__(self): # สร้างคอนสตรัคเตอร์และกำหนดแอ็ตทริบิวต์
3         self.ID = 685121
4         self.fname = 'Tony'
5         self.lname = 'Sinatra'
6         self.__score = 65 # แอ็ตทริบิวต์ที่ลูกป้องกันการเข้าถึง
7
8     def ShowData(self): # เมรอดแสดงข้อมูล
9         print(self.ID, self.fname, self.lname, self.__score)
10
11    def __updateData(self, degree): # เมรอดที่ลูกป้องกันการเข้าถึง
12        self.degree = degree
13        print(self.ID, self.fname, self.lname, self.__score,
14             self.degree)
15
16 std = Student()
17 std.ShowData()
18 std.__updateData('Master')
```

```
685121 Tony Sinatra 65
Traceback (most recent call last):
  File "/Users/epsilononxe/untitled.py", line 18, in <module>
    std.__updateData('Master')
AttributeError: 'Student' object has no attribute '__updateData'
```



ผลลัพธ์ที่ได้จากการสั่งให้โปรแกรมทำงานจะเห็นว่าเกิดความผิดพลาดขึ้น เพราะคลาส Student ไม่มีจัดแอ็ตทริบิวต์ \_\_updateData จากเรียกใช้งานผ่านอินสแตนซ์ที่สร้างขึ้นมา ทำให้แสดงผลลัพธ์จากเมรอด Showdata() เท่านั้น สำหรับแอ็ตทริบิวต์ \_\_score ก็มีการป้องกันการเข้าถึงเช่นกัน หากเราทำการแก้ไขข้อมูลโดยตรงจะทำให้เกิดข้อผิดพลาด เช่นกัน สำหรับวิธีการนำข้อมูลจากเมรอด \_\_updateData มาแสดงทำได้ดังตัวอย่างต่อไปนี้

### ตัวอย่าง 8.15

การเข้าถึงแอตทริบิวต์หรือเมธอดที่ถูกห่อหุ้ม

```
1 class Student: # ประกาศสร้างคลาส Book
2     def __init__(self): # สร้างคอนสตรัคเตอร์และกำหนดแอตทริบิวต์
3         self.ID = 685121
4         self.fname = 'Tony'
5         self.lname = 'Sinatra'
6         self.__score = 65 # แอตทริบิวต์ที่ถูกป้องกันการเข้าถึง
7
8     def ShowData(self): # เมธอดแสดงข้อมูล
9         print(self.ID, self.fname, self.lname, self.__score)
10
11    def __updateData(self, degree): # เมธอดที่ถูกป้องกันการเข้าถึง
12        self.degree = degree
13        print(self.ID, self.fname, self.lname, self.__score,
14              self.degree)
15
16 std = Student()
17 std.ShowData()
18 std.__score = 50 # แก้ไขข้อมูลแอตทริบิวต์ __score
19 std._Student__updateData('Master') # ข้อมูล __score มีค่าเท่าเดิม
20 std._Student__score = 50 # แก้ไขข้อมูลแอตทริบิวต์ __score ผ่านคลาส Student
21 std._Student__updateData('Master') # ข้อมูลแอตทริบิวต์ __score เป็น 50
```

```
685121 Tony Sinatra 65
685121 Tony Sinatra 65 Master
685121 Tony Sinatra 50 Master
```



จากตัวอย่าง 8.15 เมื่อเราต้องการแก้ไขข้อมูลแอตทริบิวต์หรือต้องการเรียกใช้งานเมธอดที่ถูกป้องกันข้อมูลไว้ ให้เราใช้เครื่องหมาย \_ (Underscore) ตามด้วยชื่อคลาสที่ต้องการเรียกใช้งานตามด้วยแอตทริบิวต์หรือเมธอด แสดงตามตัวอย่างในบรรทัดที่ 19-21 และจะสังเกตเห็นว่าในบรรทัดที่ 18 ถึงแม้ว่าเราทำการเปลี่ยนข้อมูลแอตทริบิวต์ \_\_score แต่เมื่อแสดงผลออกมา ค่าแอตทริบิวต์ของ \_\_score ยังคงเป็น **65** เช่นเดิม

## สรุปท้ายบท

ในบทนี้ได้เราได้เรียนรู้วิธีการเขียนโปรแกรมแบบ OOP ซึ่งเป็นวิธีการเขียนโปรแกรมที่มีความซับซ้อนมากขึ้น แต่มีความยืดหยุ่นกว่าการเขียนโปรแกรมรูปแบบอื่น ๆ การเขียนโปรแกรมแบบ OOP มีวิธีการเขียนและเทคนิคอื่น ๆ อีกจำนวนมาก แต่เราได้รู้จักวิธีการเบื้องต้น เช่น การสร้างคลาส เมธอด และทริบิวต์ การໂອເວອຣີໂລດດຶງຕົວດໍາເນີນການ ເປັນຕົນ ການເຂີຍໂປຣແກຣມຮູບແບບນີ້ຄ່ອນຂ້າງທຳຄວາມເຂົ້າໄດ້ຢາກສໍາຫຼັບຜູ້ເວີມຕົນ ແຕ່ຄ້າທາກຜູ້ເວີມຕົນຝຶກຝົນອຸ່ນປ່ອຍ ພ່ອ ການເຂີຍໂປຣແກຣມແບບນີ້ຈະກ່າຍເປັນເວື່ອງທີ່ຈ່າຍ ແລະໂປຣແກຣມທີ່ຖືກພັດນາແບບ OOP ນັ້ນຈະສາມາດແກ້ໄຂແລະປັບປຸງໂປຣແກຣມໄດ້ຈ່າຍ

## แบบฝึกหัด

1. จงสร้างคลาส Vehicle ที่มีแอ็ตทริบิวต์ name, max\_speed และ mileage และมีเมธอด showProperty() เพื่อแสดงข้อมูลของอินสแตนซ์ เช่น

```
Vehicle -> Name: BMW i8, MaxSpeed: 225 kmph, Trip:  
→ 12334 km
```

2. จงสร้างคลาส Bus ที่สืบทอดแอ็ตทริบิวต์และเมธอดทั้งหมดจากคลาส Vehicle

3. จงสร้างคลาส Bus ที่สืบทอดจากคลาส Vehicle และ

- (a) มีแอ็ตทริบิวต์ max\_capacity
- (b) โอบอร์โอลดเมธอด showProperty() เพื่อแสดงข้อมูลของอินสแตนซ์ เช่น

```
Bus -> Name: Fuso Aero, MaxSpeed: 155 kmph,  
→ Mileage: 162334 km, MaxCapacity: 35
```

- (c) มีเมธอด showMaxCapacity() ที่ใช้แสดงข้อมูลแอ็ตทริบิวต์ max\_capacity เช่น

```
The max capacity of Fuso Aero is 35 passengers.
```



## บทที่ 9

# การจัดการข้อผิดพลาด

จากหลักบทที่ผ่านเราได้เจอกับปัญหาการแจ้งเตือนข้อผิดพลาด (Error) จากการเขียนคำสั่งโปรแกรมที่ไม่ถูกต้องไปบ้างแล้ว เช่น ใส่ค่าตัวแปรที่ยังไม่ประกาศ ไม่ใส่เครื่องหมาย Colon (:) การป้อนชนิดข้อมูลผิดประเภท เป็นต้น ข้อผิดพลาดต่าง ๆ เหล่านี้สามารถเกิดขึ้นได้กับทุก ๆ คน แม้แต่โปรแกรมเมอร์ที่มีความเชี่ยวชาญแล้วก็ตาม ในบทนี้เราจะมาศึกษาถึงวิธีการจัดการข้อผิดพลาด (Exceptions Handling) ซึ่งข้อผิดพลาดที่เกิดขึ้นในการเขียนโปรแกรมแบ่งออกได้เป็น 3 ประเภทใหญ่ ๆ ได้แก่

1. Syntax Error คือ การเขียนคำสั่งโปรแกรมผิดหลักไวยากรณ์ที่กำหนดไว้ เช่น การลืมใส่เครื่องหมายวงเล็บ การลืมใส่เครื่องหมาย Colon (:) การนำคำสั่งมาตั้งชื่อเป็นตัวแปร การเรียกใช้งานไลบรารีผิด เป็นต้น ข้อผิดพลาดลักษณะนี้จะเกิดขึ้นตอนที่เราสั่งให้โปรแกรมประมวลผล ส่งผลทำให้โปรแกรมไม่สามารถทำงานได้ แสดงดังตัวอย่างต่อไปนี้
2. Runtime Error คือ การเขียนคำสั่งโปรแกรมที่ไม่ถูกต้องโดยผู้เขียนโปรแกรมเอง เช่น การกำหนดให้กรอกข้อมูล แต่ผู้ใช้งานกรอกชนิดข้อมูลผิดประเภทตามที่ประกาศไว้ การกำหนดเลขศูนย์ให้เป็นตัวหารประกาศตัวแปรใช้งานผิดประเภท เป็นต้น หรือเกิดจากทรัพยากรของเครื่องคอมพิวเตอร์เอง เช่น หน่วยจัดเก็บข้อมูลเต็ม หน่วยความจำไม่เพียงพอ ไม่สามารถเขียนต่อกับฐานข้อมูลได้ เป็นต้น
3. Logic Error คือ การทำงานของโปรแกรมผิดพลาดที่เกิดมาจากตัวผู้เขียนโปรแกรมเอง ซึ่งอาจจะมีสาเหตุมาจากความเข้าใจผิด จากระบวนการทำงานของโปรแกรมหรือสูตรการคำนวณ เป็นสาเหตุให้การคำนวณของโปรแกรมผิดพลาด และส่งผลให้ผลลัพธ์ที่ได้ออกมานั้นไม่ถูกต้องตามต้องการ การเกิดข้อผิดพลาดลักษณะนี้ผู้เขียนโปรแกรมต้องระมัดระวังเป็นพิเศษ เนื่องจากจะไม่มีการแจ้งเตือนให้เห็นแต่ต้องสังเกตเอาเอง และยังเป็นข้อผิดพลาดที่หายากมากที่สุดถ้าโปรแกรมมีขนาดใหญ่

### ตัวอย่าง 9.1

การแจ้งเตือนข้อผิดพลาดเมื่อผู้เขียนโปรแกรมไม่ใส่เครื่องหมาย Colon (:) ปิดท้ายคำสั่ง `for`

```
1 for i in range(1, 10)
2     print(f'รอบที่ {i}')
```

```
File 'src/test.py', line 1
    for i in range(1, 10)
          ^

```

```
SyntaxError: invalid syntax
```



### ตัวอย่าง 9.2

ลักษณะการแจ้งเตือนข้อผิดพลาดเมื่อกรอกข้อมูลไม่ถูกต้อง

```
1 n = int(input('กรุณาระบุจำนวนตัวเลข 1-5 : '))
2 for i in range(1, n):
3     print(f'รอบที่ {i} ', end = ' ')
```

```
กรุณาระบุจำนวนตัวเลข 1-5 : 2.5
Traceback (most recent call last):
  File 'src/test.py', line 1, in <module>
    n = int(input('กรุณาระบุจำนวนตัวเลข 1-5 : '))
ValueError: invalid literal for int() with base 10: '2.5'
```



### ตัวอย่าง 9.3

ลักษณะการทำงานของโปรแกรมที่ผิดพลาดจาก Logic

```
1 b = float(input('ป้อนความยาวฐาน = '))
2 h = float(input('ป้อนความสูง = '))
3 area = 0.5 * b * h
4 print('พื้นที่สามเหลี่ยม = ', area)
```

```
ป้อนความยาวฐาน = 4
ป้อนความสูง = 6
พื้นที่สามเหลี่ยม = 8.0
```



ตัวอย่างนี้เป็นโปรแกรมคำนวนหาพื้นที่สามเหลี่ยม โดยให้ผู้ใช้งานป้อนความยาวฐานและความสูง ผ่านทางคีย์บอร์ดด้วยฟังก์ชัน `input()` ในบรรทัดที่ 1 และ 2 ส่วนบรรทัดที่ 3 เป็นการคำนวนหาพื้นที่สามเหลี่ยม เราจะสังเกตเห็นว่าสูตรคำนวนพื้นที่สามเหลี่ยมผิด ซึ่งอาจจะเกิดได้จากหลายสาเหตุ เช่น กดเครื่องหมายคุณเป็นเครื่องหมายบวก หรือเข้าใจสูตรคำนวนผิด จึงส่งผลให้ผลการคำนวนออกมาไม่ถูกต้อง ดังนั้น Logic Error จึงเป็นส่วนที่ต้องระมัดระวัง

การจัดการกับข้อผิดพลาดเป็นสิ่งที่สำคัญมาก เนื่องจากช่วยให้เราจุดเก้าอี้และสาเหตุที่ทำให้เกิดข้อผิดพลาดที่เกิดขึ้นได้ง่าย ในบทนี้เราจะได้เรียนรู้วิธีการจัดการกับข้อผิดพลาด โดยการนำเอาประเภทของ Exception ต่าง ๆ ที่ภาษาไพธอนได้จัดเตรียมไว้ให้ นำมาตรวจสอบข้อผิดพลาดที่อาจจะเกิดขึ้นในโปรแกรมที่เราได้พัฒนา

## 9.1 ประเภทของ Exception ในภาษาไพธอน

เมื่อเรากำลังพัฒนาโปรแกรมอยู่แล้วทำการทดสอบ ผลปรากฏว่ามีเหตุการณ์ข้อผิดพลาดเกิดขึ้น ซึ่งผลกระทบต่อการทำงานโปรแกรมหรือทำให้โปรแกรมไม่สามารถทำงานต่อไปได้ และมีข้อความแจ้งเตือนประกอบด้วย 2 ส่วน โดยส่วนที่ 1 คือ ประเภท Exception ที่เกิดขึ้น และแสดงบรรทัดที่เกิด Exception และส่วนที่ 2 แสดงด้วยชื่อ Exception และหลังเครื่องหมาย Colon (:) เป็นส่วนคำอธิบายถึงสาเหตุที่เกิดข้อผิดพลาดขึ้น ในส่วนนี้จะนำเสนอประเภท Exception ที่เราอาจจะพบอยู่เสมอ ในขณะที่กำลังทดสอบโปรแกรมและนำเอามาใช้งานจัดการกับข้อผิดพลาด หากเราต้องการข้อมูลเพิ่มสามารถเข้าไปศึกษาได้ที่ <https://docs.python.org/3/library/exceptions.html>

Exception	ความหมาย
<b>ArithmetError</b>	เกิดข้อผิดพลาดเกี่ยวกับการคำนวณทางคณิตศาสตร์ทั้งหมด
<b>AssertionError</b>	เกิดจากการใช้คำสั่ง assert
<b>AttributeError</b>	กำหนดหรือการอ้างถึงแอ็อดทริบิวต์ไม่ถูกต้อง
<b>ConnectionError</b>	ไม่สามารถเชื่อมต่อได้
<b>EOFError</b>	เกิดขึ้นเมื่ออ่านข้อมูลถึงจุดสุดท้าย
<b>FileExistsError</b>	ตั้งชื่อสร้างไฟล์หรือไดเรกทอรีซ้ำ
<b>FileNotFoundException</b>	ค้นหาไฟล์หรือไดเรกทอรีไม่พบ
<b>FloatingPointError</b>	การดำเนินการเกี่ยวกับชนิดข้อมูลเลขทศนิยมผิดพลาด
<b>ImportError</b>	เรียกใช้งานไลบรารีผิด
<b>IndentationError</b>	เกิดข้อผิดพลาดจากการย่อหน้า
<b>IndexError</b>	ไม่พบตำแหน่งที่ระบุ
<b>ModuleNotFoundError</b>	ไม่พบโมดูลที่ระบุ
<b>KeyError</b>	ไม่พบคีย์ที่ระบุ
<b>KeyboardInterrupt</b>	เกิดการขัดจังหวะการทำงานด้วยปุ่ม Ctrl+C หรือ Delete
<b>MemoryError</b>	หน่วยความจำไม่เพียงพอ
<b>NameError</b>	ไม่พบชื่อในตัวแปรทั้งแบบ local และ global
<b>OverflowError</b>	ผลการคำนวณเกินค่าที่ได้กำหนดไว้
<b>OSError</b>	เกิดจากปัญหาของระบบปฏิบัติการ
<b>PermissionError</b>	ความผิดพลาดที่ไม่มีสิทธิ์เข้าใช้งาน
<b>RuntimeError</b>	ข้อผิดพลาดขณะโปรแกรมกำลังทำงาน
<b>SyntaxError</b>	เขียนคำสั่งโปรแกรมผิดไวยากรณ์
<b>SystemError</b>	ความผิดพลาดที่เกิดจากระบบ
<b>TabError</b>	ปัญหาการใช้คีย์ tab และเคาะวรรค (space)
<b>TypeError</b>	ระบุชนิดข้อมูลไม่ถูกต้อง
<b>UnboundLocalError</b>	เรียกใช้งานตัวแปรแบบ local ที่ยังไม่ได้กำหนดค่า
<b>ValueError</b>	กำหนดค่าไม่เหมาะสมกับชนิดข้อมูล
<b>ZeroDivisionError</b>	ความผิดพลาดจากการนำเลข 0 มาหาร

ตาราง 9.1: ประเภทของ Exceptions ในภาษา Python

## 9.2 การตรวจจับข้อผิดพลาดประเภท Exception

การเกิดข้อผิดพลาดจากโปรแกรมหรือจากการทำงานของระบบ ทำให้โปรแกรมไม่สามารถทำงานต่อไปได้ ซึ่งเราจะต้องเขียนคำสั่งจัดการกับเหตุการณ์ที่เกิดขึ้นไว้ด้วย สำหรับคำสั่งที่ช่วยให้เราสามารถตรวจจับของชนิดข้อผิดพลาดและทราบถึงสาเหตุที่เกิดขึ้นในภาษาไพธอน ได้แก่ คำสั่ง **try...except** ซึ่งมีรูปแบบการใช้งานดังต่อไปนี้

### 9.2.1 การใช้คำสั่ง **try...except** ตรวจจับประเภท Exception

คำสั่ง **try...except** ใช้ตรวจจับข้อผิดพลาดประเภท Exception ที่อาจจะเกิดขึ้นขณะโปรแกรมกำลังทำงาน โดยคำสั่งที่อยู่ภายใต้ขอบเขตคำสั่ง **try** เป็นคำสั่งที่ต้องการให้ตรวจจับความผิดพลาด และคำสั่งที่อยู่ในขอบเขตคำสั่ง **except** เป็นคำสั่งที่ต้องการให้ทำงานเมื่อมีข้อผิดพลาดเกิดขึ้น มีโครงสร้างการใช้คำสั่งดังนี้

รูปแบบการเขียนคำสั่ง **try...except**

```
try:  
    statements      # คำสั่งที่ต้องการตรวจจับความผิดพลาด  
except Exception_1: # ประเภทของความผิดพลาด  
    statement_exc_1 # คำสั่งที่ให้ทำงานเมื่อตรวจพบข้อผิดพลาด  
    ...  
except Exception_n: # ประเภทของความผิดพลาดอื่น  
    statement_exc_n # คำสั่งที่ให้ทำงานเมื่อตรวจพบข้อผิดพลาด
```

เราสามารถกำหนดให้มีการตรวจจับคำสั่งที่คาดว่าจะเกิดข้อผิดพลาดภายใต้คำสั่ง **try** ได้หลายคำสั่ง และใช้คำสั่ง **except** ตรวจจับประเภท Exception ได้หลายตัว

#### ตัวอย่าง 9.4

การตรวจจับข้อผิดพลาดเมื่อผู้ใช้งานป้อนข้อมูลไม่ถูกต้อง และหารด้วยเลข 0

```
1 x = 5
2 try:
3     n = int(input('กรุณาป้อนตัวเลข: '))
4     z = x / n
5     print(z)
6 except ZeroDivisionError: # ตรวจจับข้อผิดพลาดเมื่อตัวหารเป็นเลข 0
7     print('ไม่สามารถหารด้วย 0 ได้')
8 except ValueError: # ตรวจจับข้อผิดพลาดเมื่อป้อนข้อมูลไม่ถูกต้อง
9     print('ข้อมูลที่คุณป้อนไม่ใช่ตัวเลขจำนวนเต็ม')
```

กรุณาป้อนตัวเลข:  
ไม่สามารถหารด้วย 0 ได้

กรุณาป้อนตัวเลข : 5.5  
ข้อมูลที่คุณป้อนไม่ใช่ตัวเลขจำนวนเต็ม



บรรทัดที่ 3-5 คือ คำสั่งที่ต้องการตรวจสอบข้อผิดพลาด บรรทัดที่ 6 คำสั่งตรวจจับข้อผิดพลาด เมื่อผู้ใช้งานป้อนข้อมูลเป็นเลข 0 ทำให้บรรทัดที่ 7 แสดงผล บรรทัดที่ 8 คำสั่งตรวจจับข้อผิดพลาด เมื่อผู้ใช้งานป้อนข้อมูลไม่ใช่ตัวเลขจำนวนเต็ม ทำให้บรรทัดที่ 9 แสดงผล

#### 9.2.2 การใช้คำสั่ง `try...except` ตรวจจับ Exception หลายตัว

เราสามารถกำหนดประเภท Exception ให้ตรวจจับความผิดพลาดได้หลายประเภท โดยกำหนดไว้ในส่วนของคำสั่ง `except` มีโครงสร้างการเขียนคำสั่งดังนี้

รูปแบบการเขียนคำสั่ง `try...except` ตรวจจับ Exceptions หลายตัว

```
try:
    statements
except (Exception_1, Exception_2, ..., Exception_n):
    statement_exc
```

### ตัวอย่าง 9.5

การกำหนดให้คำสั่ง **except** ตรวจสอบข้อผิดพลาดประเภท Exceptions แบบหลายตัว

```
1 lst = [5, 6, 8, 9, 10, 15]
2 dct = {1:'Tennis', 2:'Football', 3:'Racing', 4:'Running'}
3 try:
4     print('ตำแหน่งที่ 5 ของ lst =', lst[5])
5     print('ตำแหน่งที่ 2 ของ dct =', dct[2]) # ใส่คีย์ลูกต้อง
6 except(IndexError, KeyError): # ตัวจับแบบหลายตัว
7     print('ตำแหน่ง หรือ คีย์ที่ระบุไม่ถูกต้อง')
```

ตำแหน่งที่ 5 ของ lst = 15  
ตำแหน่งที่ 2 ของ dct = Football



### ตัวอย่าง 9.6

```
1 lst = [5, 6, 8, 9, 10, 15]
2 dct = {1:'Tennis', 2:'Football', 3:'Racing', 4:'Running'}
3 try:
4     print('ตำแหน่งที่ 5 ของ lst =', lst[5])
5     print('ตำแหน่งที่ 2 ของ dct =', dct[2]) # ใส่คีย์ลูกต้อง
6 except(IndexError, KeyError): # ตัวจับแบบหลายตัว
7     print('ตำแหน่ง หรือ คีย์ที่ระบุไม่ถูกต้อง')
```

ตำแหน่งที่ 5 ของ lst = 15  
ตำแหน่ง หรือ คีย์ที่ระบุไม่ถูกต้อง



## 9.2.3 การใช้คำสั่ง **try...except...else** ตรวจสอบประเภท Exception

เมื่อตรวจสอบข้อผิดพลาดการทำงานของโปรแกรม จะแสดงผลตามประเภท Exception ที่ได้ระบุไว้ หลังคำสั่ง **except** แต่ถ้าต้องการแสดงผลคำสั่งโปรแกรมหลังจากทำงานจบและไม่พบข้อผิดพลาดเกิดขึ้นเราจะใช้คำสั่ง **else** ดังนี้

รูปแบบการเขียนคำสั่ง **try...except...else** ตรวจจับข้อผิดพลาด

```
try:  
    statements  
except Exception_1: # ประเภทของข้อผิดพลาดที่ 1  
    ...  
except Exception_n: # ประเภทของข้อผิดพลาดที่ n  
    statements_n  
else:  
    statements # คำสั่งที่ทำงานเมื่อไม่พบข้อผิดพลาด 1-n
```

### ตัวอย่าง 9.7

#### การใช้คำสั่ง **try...except...else**

```
1 x = [5, 12, 6, 9, 13]  
2 try:  
3     n = int(input('กรุณาป้อนตัวเลข : '))  
4     i = int(input('กรุณาป้อนตำแหน่งข้อมูลในลิสต์ : '))  
5     z = x[i]  
6 except IndexError:  
7     print('ไม่พบตำแหน่งที่คุณระบุในลิสต์ x')  
8 except ArithmeticError:  
9     print('ตัวหารเป็น零 0')  
10 except ValueError:  
11     print('ข้อมูลที่คุณป้อนไม่ใช่ตัวเลข')  
12 else:  
13     print('ไม่พบข้อผิดพลาดของ Exception')  
14     print('ผลลัพธ์ที่ได้ = ', z)
```

กรุณาป้อนตัวเลข : 5  
กรุณาป้อนตำแหน่งข้อมูลในลิสต์ : 3  
ไม่พบข้อผิดพลาดของ Exception  
ผลลัพธ์ที่ได้ = 9



กรุณาป้อนตัวเลข : 5  
กรุณาป้อนตำแหน่งข้อมูลในลิสต์ : 10  
ไม่พบตำแหน่งที่คุณระบุในลิสต์ x

การทำงานของโปรแกรมตามตัวอย่างนี้ เมื่อป้อนตัวเลขและระบุตำแหน่งข้อมูลที่อยู่ในลิสต์ได้ถูกต้อง โปรแกรมจะแสดงผลคำสั่งที่ได้กำหนดไว้หลังคำสั่ง **else** แต่เมื่อป้อนข้อมูลหรือระบุตำแหน่งไม่ถูกต้องจะแสดงการตรวจสอบข้อผิดพลาดตามประเภท Exception ส่งผลให้คำสั่งที่อยู่หลังคำสั่ง **else** ไม่ทำงาน

#### 9.2.4 การใช้คำสั่ง **try...except...finally** ตรวจจับประเภท Exception

เมื่อนำคำสั่ง **try...except...finally** จะส่งผลให้คำสั่งโปรแกรมที่อยู่ในขอบเขตของคำสั่ง **finally** ทำงานทุกรอบไม่ว่าจะตรวจสอบข้อผิดพลาดหรือไม่ก็ตาม แตกต่างจากคำสั่ง **else** ที่จะแสดงผลคำสั่งโปรแกรมที่อยู่ในขอบเขตเฉพาะกรณีที่ไม่พบข้อผิดพลาดเท่านั้น มีโครงสร้างรูปแบบการใช้งานดังต่อไปนี้

รูปแบบการเขียนคำสั่ง **try...except...finally** ตรวจจับข้อผิดพลาด

```
try:  
    statements  
except Exception_1: # ประเภทของข้อผิดพลาดที่ 1  
    ...  
except Exception_n: # ประเภทของข้อผิดพลาดที่ n  
    statements_n  
finally:  
    statements      # คำสั่งที่ทำงานเสมอแม้ไม่พบข้อผิดพลาด
```

## ตัวอย่าง 9.8

การเขียนคำสั่งโปรแกรม `try...except...finally`

```
1 x = [5, 12, 6, 9, 13]
2 try:
3     n = int(input('กรุณาป้อนตัวเลข : '))
4     i = int(input('กรุณาป้อนตำแหน่งข้อมูลในลิสต์ : '))
5     z = x[i] / n
6     print('ผลลัพธ์ที่ได้ = ', z)
7 except IndexError:
8     print('ไม่พบตำแหน่งที่คุณระบุในลิสต์ x')
9 except ArithmeticError:
10    print('ตัวหารเป็นเลข 0')
11 except ValueError:
12    print('ข้อมูลที่คุณป้อนไม่ใช่ตัวเลข')
13 finally:
14    print('จบการทำงาน')
```

กรุณาป้อนตัวเลข : 2  
กรุณาป้อนตำแหน่งข้อมูลในลิสต์ : 3  
ผลลัพธ์ที่ได้ = 4.5  
จบการทำงาน

กรุณาป้อนตัวเลข : 3  
กรุณาป้อนตำแหน่งข้อมูลในลิสต์ : 10  
ไม่พบตำแหน่งที่คุณระบุในลิสต์ x  
จบการทำงาน



ตัวอย่างคำสั่งโปรแกรมนี้ เป็นการนำเอาคำสั่ง `finally` เข้ามาใช้แทนคำสั่ง `else` เมื่อโปรแกรมทำงานได้ถูกต้องจะไม่มีการแจ้งเตือนข้อผิดพลาด ส่งผลให้คำสั่งโปรแกรมบรรทัดที่ 13 ทำงาน ถ้าป้อนข้อมูลไม่ถูกต้องโปรแกรมจะแสดงข้อความการแจ้งเตือนข้อผิดพลาด และคำสั่งโปรแกรมบรรทัดที่ 13 ก็จะทำงานเช่นกัน แตกต่างจากตัวอย่างที่ผ่านมา ที่ใช้คำสั่ง `else` ซึ่งจะทำงานเฉพาะกรณีที่ไม่มีข้อผิดพลาดเกิดขึ้น

### 9.2.5 การใช้คำสั่ง `try...except` ซ้อนกันเพื่อตรวจจับประเภท Exception

คำสั่ง `try...except` ยังสามารถนำมาเขียนซ้อนกันได้ ปกติคำสั่งโปรแกรมที่คาดว่าจะเกิดข้อผิดพลาดอยู่ในขอบเขตของคำสั่ง `try` ถ้าเราต้องการแยกบางคำสั่งโปรแกรมที่ต้องการตรวจจับ ошибกามาให้เราเพิ่มระดับชั้นของคำสั่ง `try...except` เข้าไป และกำหนดประเภท Exception มีโครงสร้างรูปแบบการเขียนคำสั่ง `try...except` ซ้อนกันดังนี้

รูปแบบการเขียนคำสั่ง `try...except` ซ้อนกันตรวจจับข้อผิดพลาด

```
try:  
    statements      # คำสั่งที่ต้องการตรวจจับความผิดพลาด  
    try:  
        statements      # คำสั่งที่ต้องการตรวจจับความผิดพลาด  
    except Exception:  # ประเภทของข้อผิดพลาด  
        statements      # คำสั่งให้ทำงานเมื่อพบข้อผิดพลาด  
    except Exception:  # ประเภทของข้อผิดพลาด  
        statements      # คำสั่งให้ทำงานเมื่อพบข้อผิดพลาด
```

#### ตัวอย่าง 9.9

การเขียนคำสั่ง `try...except` ซ้อนกัน

```
1 try:  
2     n = int(input('กรุณาป้อนตัวเลขจำนวนเต็ม : '))  
3     try:  
4         x = 10  
5         z = x / n  
6         print(z)  
7     except ZeroDivisionError:  
8         print('ไม่สามารถหารด้วยเลข 0')  
9     except ValueError:  
10        print('คุณป้อนข้อมูลไม่ถูกต้อง')
```

กรุณาป้อนตัวเลขจำนวนเต็ม : 5.5  
คุณป้อนข้อมูลไม่ถูกต้อง

กรุณาป้อนตัวเลขจำนวนเต็ม : 0  
ไม่สามารถหารด้วยเลข 0

จากตัวอย่างการทำงานของโปรแกรม เมื่อผู้ใช้งานป้อนข้อมูลที่ไม่มีตัวเลขจำนวนเต็ม ส่งผลให้คำสั่งโปรแกรมในบรรทัดที่ 10 แสดงผล เมื่อผู้ใช้งานป้อนเลข 0 ส่งผลให้คำสั่งโปรแกรมในบรรทัดที่ 8 แสดงผล

### 9.3 การสร้างข้อความแจ้งเตือนข้อผิดพลาดด้วยคำสั่ง raise exception

นอกจากเราใช้คำสั่ง **try...except** ตรวจจับคำสั่งโปรแกรมที่คาดว่าจะเกิดความผิดพลาดขึ้น และแจ้งเตือนตามประเภทของ Exception แล้ว ยังมีคำสั่ง **raise** ให้เราใช้งานสำหรับสร้างข้อความการแจ้งเตือนข้อผิดพลาดขึ้นมาใช้งานเอง เพื่อให้ทราบถึงสาเหตุที่เกิดขึ้นของข้อผิดพลาด ตัวอย่างเช่น การตรวจสอบเงื่อนไขว่าโปรแกรมทำงานถูกต้องตามที่กำหนดไว้หรือไม่ โดยคำสั่ง **raise** มีรูปแบบการใช้งานดังนี้

รูปแบบการเขียนคำสั่ง **raise** สร้างข้อความแจ้งเตือนข้อผิดพลาด

```
raise [Exception [, args [, traceback]]]
```

**Exception** ประเภทของ Exception ที่ตรวจจับความผิด จะมีหรือไม่มีก็ได้

**args** อาร์กิวเมนต์ของ Exception

**traceback** สาเหตุที่ทำให้เกิดข้อผิดพลาด จะมีหรือไม่มีก็ได้

#### ตัวอย่าง 9.10

การใช้คำสั่ง raise สร้างข้อความแจ้งเตือนข้อผิดพลาด ตรวจสอบค่าตัวเลขจากผู้ใช้ที่ป้อนเข้ามาผ่านทางคีย์บอร์ด

```
1 n = int(input('ป้อนตัวเลข 1-5 : '))
2 if n > 5: # ตรวจสอบตัวเลขที่ป้อนเข้ามา
3     # ใช้คำสั่ง raise สร้างข้อความแจ้งเตือนข้อผิดพลาด
4     raise TypeError('คุณป้อนค่าตัวเลขมากกว่าที่กำหนด', n)
5 else:
6     print('จบการทำงาน !!')
```

```
ป้อนตัวเลข 1-5 : 8
Traceback (most recent call last):
  File 'src/test.py', line 4, in <module>
    raise TypeError('คุณป้อนค่าตัวเลขมากกว่าที่กำหนด', n)
TypeError: ('คุณป้อนค่าตัวเลขมากกว่าที่กำหนด', 8)
```

## 9.4 การสร้างและเรียกใช้งาน Exception ที่สร้างขึ้นเอง

นอกจากเราจะเรียกใช้งานประเภทของ Exception ที่ภาษา Python จัดเตรียมไว้ให้แล้ว เรายังสามารถสร้าง Exception ขึ้นมาใช้งานเองได้ (User-defined Exception) โดยการสร้างคลาสประเภทของข้อผิดพลาดที่สืบทอดมาจากคลาสประเภทของ Exception นั้น ๆ หรือสืบทอดมาจากคลาส Exception เองได้เลย หรือเรียกได้ว่าเป็น Exception ที่เราสร้างขึ้นมาใช้เองเป็น subclass (คลาสลูก) และประเภทของ Exception เป็น superclass (คลาสแม่) เราสามารถศึกษาวิธีการสร้างประเภทของ Exception ขึ้นมาใช้เองได้ตามด้วยต่อไปนี้

### ตัวอย่าง 9.11

การสร้างประเภท Exception ขึ้นมาใช้งานเองโดยมีคลาส Exception เป็น superclass

```
1 class StockError(Exception):
2     # สร้าง subclass ชื่อ LessThanStockError ที่มีคลาส Exception เป็น superclass
3     def __init__(self):
4         # กำหนดให้มีข้อความแสดงผลเมื่อมีผิดพลาดเกิดขึ้น
5         Exception.__init__(self, 'สินค้าคงเหลือไม่เพียงพอ')
6
7 x = 10
8 print(f'สินค้าคงเหลือคือ {x} ชิ้น')
9 n = int(input('จำนวนสินค้าที่ขายได้ = '))
10 z = x - n
11 if z < 0:
12     # เรียกใช้ประเภท Exception ของ StockError
13     raise StockError
14 else:
15     print(f'สินค้าคงเหลือ = {z} ชิ้น')
```

สินค้าคงเหลือคือ 10 ชิ้น  
จำนวนสินค้าที่ขายได้ = 3  
สินค้าคงเหลือ = 7 ชิ้น

สินค้าคงเหลือคือ 10 ชิ้น  
จำนวนสินค้าที่ขายได้ = 15  
Traceback (most recent call last):  
 File 'src/test.py', line 13, in <module>  
 raise StockError  
\_\_main\_\_.StockError: สินค้าคงเหลือไม่เพียงพอ



## ตัวอย่าง 9.12

การสร้างการแจ้งเตือนเมื่อผู้ใช้งานกรอกคะแนนสอบ

```
1 class ScoreTestError(Exception):
2     pass
3
4 class FailTestError(Exception):
5     pass
6
7 try:
8     n = int(input('กรุณารอกรอกคะแนนเป็นจำนวนเต็ม (0-100) : '))
9     if n < 50:
10         raise FailTestError
11     elif n > 100:
12         raise ScoreTestError
13     else:
14         print(f'ยินดีด้วยครับคุณสอบผ่าน คะแนนที่คุณได้ คือ {n} ')
15 except FailTestError:
16     print(f'คุณไม่ผ่านเกณฑ์ 50 คะแนน คะแนนที่คุณได้ {n} คะแนน')
17 except ScoreTestError:
18     print(f'คะแนนที่คุณได้ {n} > 100 โปรดตรวจสอบอีกครั้ง')
19 except ValueError:
20     print('คุณป้อนข้อมูลไม่ถูกต้อง')
```

กรุณารอกรอกคะแนนเป็นจำนวนเต็ม (0-100) : 45.5

คุณป้อนข้อมูลไม่ถูกต้อง



กรุณารอกรอกคะแนนเป็นจำนวนเต็ม (0-100) : 45

คุณไม่ผ่านเกณฑ์ 50 คะแนน คะแนนที่คุณได้ 45 คะแนน

กรุณารอกรอกคะแนนเป็นจำนวนเต็ม (0-100) : 80

ยินดีด้วยครับคุณสอบผ่าน คะแนนที่คุณได้ คือ 80

กรุณารอกรอกคะแนนเป็นจำนวนเต็ม (0-100) : 888

คะแนนที่คุณได้ คือ 888 > 100 โปรดตรวจสอบอีกครั้ง

## 9.5 การยืนยันความถูกต้อง

การทดสอบโปรแกรมเป็นสิ่งที่ผู้พัฒนาโปรแกรมต้องทำอยู่เสมอ เพื่อตรวจหาข้อผิดพลาดจากการทำงานของคำสั่งโปรแกรม นอกจากการใช้คำสั่ง `try...except` การใช้คำสั่ง `assert` เพื่อตรวจจับข้อผิดพลาดที่เกิดขึ้นก็เป็นอีกหนึ่งวิธีที่มีความสะดวก ซึ่งเป็นการยืนยันและเพื่อให้แน่ใจว่าคำสั่งโปรแกรมนั้นจะไม่มีโอกาสเกิดข้อผิดพลาดขึ้นอย่างแน่นอน (Assertion) ถ้าคำสั่ง `assert` ตรวจสอบข้อผิดพลาดจะแสดงประเภท Exception ของ `AssertionError` ออกมาก โดยมีรูปแบบการใช้งานดังนี้

รูปแบบการใช้คำสั่ง `assert`

`assert expression [, arguments]`

`expression` เงื่อนไขที่ต้องการทดสอบการทำงานของโปรแกรม

`arguments` คำอธิบายการเกิดข้อผิดพลาด จะมีหรือไม่มีก็ได้

### ตัวอย่าง 9.13

การใช้คำสั่ง `assert` ทดสอบความถูกต้องการทำงานของคำสั่งโปรแกรม

```
1 x = int(input('ค่าของ x = '))
2 y = int(input('ค่าของ y = '))
3 assert x == y, 'ค่าตัวแปร x และ y ต้องเท่ากัน'
4 print('จบการทำงาน')
```

ค่าของ x = 1  
ค่าของ y = 1  
จบการทำงาน

ค่าของ x = 1  
ค่าของ y = 2  
Traceback (most recent call last):  
 File 'src/test.py', line 3, in <module>  
 assert x == y, 'ค่าตัวแปร x และ y ต้องเท่ากัน'  
AssertionError: ค่าตัวแปร x และ y ต้องเท่ากัน



จากตัวอย่าง 9.13 ในบรรทัดที่ 3 เป็นการใช้คำสั่ง `assert` ตรวจจับข้อผิดพลาดของโปรแกรม เพื่อยืนยันความถูกต้องของค่าตัวแปร x กับ y ต้องมีค่าเท่ากัน

### ตัวอย่าง 9.14

การใช้คำสั่ง **assert** ทดสอบความถูกต้องการทำงานของคำสั่งโปรแกรมร่วมกับคำสั่ง **try...except**

```
1 try:  
2     n = int(input('กรุณารอค่าคะแนนเป็นจำนวนเต็ม (0-30): '))  
3     # ตรวจจับความผิดพลาดเมื่อค่าตัวแปร n มีค่านอกกว่า 30  
4     assert n <= 30  
5     print('คะแนนที่คุณกรอก คือ ', n)  
6 except ValueError:  
7     # ส่งข้อผิดพลาดประเภท Exception ของ ValueError  
8     print('คุณป้อนค่าตัวเลขไม่ถูกต้อง')  
9 except AssertionError:  
10    # ส่งข้อผิดพลาดประเภท Exception ของ AssertionError  
11    print('คุณป้อนค่าตัวเลขสูงกว่า 30')
```

กรุณารอค่าคะแนนเป็นจำนวนเต็ม (0-30): 25.2  
คุณป้อนค่าตัวเลขไม่ถูกต้อง



กรุณารอค่าคะแนนเป็นจำนวนเต็ม (0-30): 30  
คะแนนที่คุณกรอก คือ 30

กรุณารอค่าคะแนนเป็นจำนวนเต็ม (0-30): 45  
คุณป้อนค่าตัวเลขสูงกว่า 30

## สรุปท้ายบท

ข้อผิดพลาดจากการทำงานของโปรแกรมมีมาจากการหลâyสารเหตุ ไม่ว่าเกิดจากระบบปฏิบัติการหรือเกิดจากการเขียนคำสั่งโดยผู้เขียนโปรแกรมเอง แต่ภาษาไพธอนก็ได้จัดเตรียมประเภท Exception ต่าง ๆ ให้เราใช้งานจำนวนมาก เพื่อจัดการกับข้อผิดพลาดโดยการใช้คำสั่ง **try...except** และเรายังสามารถสร้างการแจ้งเตือนข้อผิดพลาดขึ้นมาใช้งานเองได้ด้วยคำสั่ง **raise** รวมไปถึงการใช้คำสั่ง **assert** เพื่อยืนยันความถูกต้อง (Assertion) การทำงานของคำสั่งโปรแกรมในจุดต่าง ๆ ที่อาจเกิดปัญหาด้วย

## แบบฝึกหัด

- จ�อธิบายผลลัพธ์ที่ได้จากการคำสั่งโปรแกรมต่อไปนี้ ในกรณีที่ป้อนข้อมูลได้ถูกต้องและไม่ถูกต้อง

```
1 try:
2     n = int(input('กรอกจำนวนเต็มบวกไม่เกิน 10 : '))
3     if n <= 10:
4         while n <= 10:
5             print(n, end=' ')
6             n = n + 1
7         print('OK')
8     except ValueError:
9         print('ป้อนข้อมูลไม่ถูกต้อง')
```

- จจอธิบายผลลัพธ์ที่ได้จากการคำสั่งโปรแกรมต่อไปนี้ และอธิบายสาเหตุที่ทำให้เกิดการแจ้งเตือนข้อผิดพลาดขึ้น พร้อมทั้งแก้ไขโปรแกรมให้ถูกต้อง

```
1 try:
2     n = int(input('กรอกจำนวนเต็มบวกไม่เกิน 10 : '))
3     if x <= 10:
4         while n <= 10:
5             print(n, end=' ')
6             n = n + 1
7         print('OK')
8     except ValueError:
9         print('ป้อนข้อมูลไม่ถูกต้อง')
10    except nameError:
11        Print('ตั้งชื่อตัวแปรไม่ถูกต้อง')
```

- เขียนคำสั่งโปรแกรมให้มีการดักจับ Exception ดังต่อไปนี้

- (a) **FloatingPointError**
- (b) **KeyError**
- (c) **IndexError**
- (d) **ValueError**



# บทที่ 10

## การจัดการกับไฟล์

ในบทนี้เราจะได้เรียนรู้วิธีการจัดการกับไฟล์หรือแฟ้มข้อมูล (File) ซึ่งเป็นเรื่องที่มีความสำคัญมากอีกบทหนึ่งที่ผู้อ่านควรทำความเข้าใจ โดยส่วนใหญ่แล้วเรามักจะมีการจัดเก็บข้อมูลไว้ในหน่วยความจำ (Storage Device) เช่น Hard Disk Drive, Solid-State Drive หรือ Flash Storage

ทั้งนี้เราจะได้เรียนรู้ตั้งแต่ขั้นตอนการนำไฟล์มาประมวลผล ไปจนถึงการบันทึกข้อมูลลงในไฟล์เดิมหรือการสร้างไฟล์ใหม่ รวมถึงการจัดการกับไดเรกทอรี (Directory) หรือโฟลเดอร์ (Folder) ซึ่งภาษาโปรแกรมได้จัดเตรียมเมโมรอดต่าง ๆ จำนวนมากไว้ให้ใช้งาน

### 10.1 โหมดของไฟล์

เมื่อเราได้สร้างไฟล์ขึ้นมาจัดเก็บข้อมูลและมีการแยกจัดเก็บไฟล์อย่างเป็นระเบียบภายใต้ไดเรกทอรีอีกชั้น พร้อมทั้งมีการตั้งชื่อไฟล์และไดเรกทอรีให้สื่อความหมายให้ง่ายต่อการค้นหา ถ้าต้องการอ่านไฟล์หรือเขียนไฟล์จากตำแหน่งที่ได้จัดเก็บไว้ในสื่ออุปกรณ์ต่าง ๆ เช่น ฮาร์ดดิสก์ หรือแฟลชไดรฟ์ การอ่านและเขียนไฟล์จากตำแหน่งที่จัดเก็บด้วยภาษาโปรแกรมจะมีการระบุโหมดการอ่านและเขียนไฟล์ออกเป็น 2 โหมด ได้แก่ โหมดไฟล์ข้อความ (Text File Mode) และโหมดไฟล์ไบนาリー (Binary File Mode)

#### 10.1.1 โหมดไฟล์ข้อความ (Text File Mode)

การอ่านและเขียนด้วยโหมดไฟล์ข้อความเป็นการทำงานระหว่างโปรแกรมกับไฟล์ข้อความ (.txt) ที่มีการใช้รหัส Unicode (Unicode encoding และ Unicode decoding) ถ้าเป็นการเขียนไฟล์จะมีสัญลักษณ์ขึ้นบรรทัดใหม่ \n บนระบบ Unix หรือ \r\n บนระบบ Windows และ \r บนระบบ macOS ที่ตำแหน่งท้ายบรรทัด หากเป็นการอ่านไฟล์จะแสดงเฉพาะสัญลักษณ์ \n ที่ด้านท้ายไฟล์ การเขียนคำสั่งโปรแกรมจัดการกับไฟล์ข้อความต้องใช้ตัวอักษรและสัญลักษณ์ ดังตาราง 10.1

โหมด ความหมาย	
a	เปิดไฟล์ขึ้นมาเขียนต่อท้ายไฟล์เดิม (append) ถ้าไม่พบไฟล์ที่ระบุจะสร้างไฟล์ขึ้นมาใหม่
a+	เปิดไฟล์ขึ้นมาอ่านและเขียนต่อท้ายไฟล์เดิม ถ้าไม่พบไฟล์ที่ระบุจะสร้างไฟล์ขึ้นมาใหม่
r	เปิดไฟล์ขึ้นมาอ่านอย่างเดียว (read only)
r+	เปิดไฟล์ขึ้นมาอ่านและเขียน ถ้าไม่พบไฟล์ที่ระบุจะเกิด error
w	เปิดไฟล์ขึ้นมาเขียนอย่างเดียว (write) ข้อมูลเดิมที่มีอยู่ในไฟล์จะถูกเขียนทับ ถ้าไม่พบไฟล์ที่ระบุจะสร้างไฟล์ขึ้นมาใหม่
w+	เปิดไฟล์ขึ้นมาอ่านและเขียน ข้อมูลเดิมที่อยู่ในไฟล์จะถูกเขียนทับ ถ้าไม่พบไฟล์ที่ระบุจะสร้างไฟล์ขึ้นมาใหม่

ตาราง 10.1: ตัวอักษรสัญลักษณ์ที่ใช้จัดการกับไฟล์ข้อความ (Text File)

### 10.1.2 โหมดไฟล์ใบ娜รี (Binary File Mode)

การจัดการกับข้อมูลด้วยโหมดไฟล์ใบ娜รีจะอยู่ในระดับบิต คือ 0 กับ 1 หรือเลขฐานสองนั้นเอง และเป็นโหมดที่ไม่มีการเปลี่ยนแปลงข้อมูลก่อนเขียนลงไฟล์ เมื่อเราต้องการเขียนคำสั่งโปรแกรมจัดการกับโหมดไฟล์ใบ娜รีต้องกำกับด้วยตัวอักษรสัญลักษณ์ ดังตาราง 10.2

โหมด ความหมาย	
ab	เปิดไฟล์ในนารีขึ้นมาเขียนต่อท้ายไฟล์เดิม ถ้าไม่พบไฟล์ที่ระบุไว้จะสร้างไฟล์ขึ้นมาใหม่
ab+	เปิดไฟล์ในนารี ขึ้นมาอ่านและเขียน ถ้าไม่พบไฟล์ที่ระบุไว้จะสร้างไฟล์ขึ้นมาใหม่
rb	เปิดไฟล์ในนารีขึ้นมาอ่านอย่างเดียว
rb+	เปิดไฟล์ในนารีขึ้นมาอ่านและเขียน ถ้าไม่พบไฟล์ที่ระบุจะเกิด error
wb	เปิดไฟล์ในนารีขึ้นมาเขียนอย่างเดียว แต่ ข้อมูลเดิมที่มีอยู่ในไฟล์จะถูกเขียนทับ ถ้าไม่พบไฟล์ที่ระบุไว้จะสร้างไฟล์ขึ้นมาใหม่
wb+	เปิดไฟล์ในนารีขึ้นมาอ่านและเขียน ข้อมูลเดิมที่อยู่ในไฟล์จะถูกเขียนทับ ถ้าไม่พบไฟล์ที่ระบุจะสร้างไฟล์ขึ้นมาใหม่

ตาราง 10.2: ตัวอักษรสัญลักษณ์ที่ใช้จัดการกับไฟล์ในนารี (Binary File)

## 10.2 เมธอดสำหรับจัดการกับไฟล์

การเขียนคำสั่งโปรแกรมจัดการกับไฟล์ มีเมธอดและฟังก์ชันให้เรียกใช้งาน สำหรับจัดการกับข้อความที่จะทำการอ่านหรือเขียน ดังตาราง 10.3

เมื่อต้องการเปิดไฟล์ขึ้นมาอ่านหรือเขียนข้อมูลลงไฟล์ใหม่ หรือการสร้างไฟล์ใหม่ตามโหมดต่าง ๆ ข้างต้น รวมทั้งการนำເມຣອດหรือຟັງກ່ຽວມາຮ່ວມໃຊ້ງານ ໃຫ້ໃໝ່ເມຣອດ `open()` ເພື່ອຂອເຊື່ອມຕ່ອກບໍລິຫານທີ່ຕ້ອງການເປີດຂຶ້ນມາໃຊ້ງານກ່ອນ ທີ່ມີຮູບແບບການເຂົ້າມີຄຳສັ່ງດັ່ງນີ້

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>close()</code>	ใช้สำหรับปิดไฟล์เพื่อบันทึกข้อมูล	<code>file_name.close()</code>
<code>flush()</code>	ใช้สำหรับนำข้อมูลที่อยู่ในบัฟเฟอร์บันทึกลงไฟล์ (แต่ยังไม่ปิดไฟล์)	<code>file_name.flush()</code>
<code>seek()</code>	ใช้สำหรับระบุตำแหน่งเริ่มต้นการอ่านข้อมูล	<code>file_name.seek(offset [, whence])</code>
<code>next()</code>	ฟังก์ชันใช้สำหรับการอ่านข้อมูลจากไฟล์ทีละบรรทัดในรูปแบบของการทำซ้ำ	<code>next(iterator[, default])</code>
<code>read()</code>	ใช้สำหรับการอ่านไฟล์ตามขนาดข้อมูลที่ระบุ	<code>file_name.read(size)</code>
<code>readline()</code>	ใช้สำหรับอ่านข้อความจากไฟล์ทีละบรรทัด	<code>file_name.readline([size])</code>
<code>readlines()</code>	ใช้สำหรับอ่านข้อความทุกบรรทัดในไฟล์ผลลัพธ์ที่ได้เป็นชนิดข้อมูลลิสต์	<code>file_name.readlines([sizehint])</code>
<code>tell()</code>	ใช้สำหรับบอกตำแหน่งปัจจุบันของตัวชี้ภายในไฟล์	<code>file_name.tell()</code>
<code>truncate()</code>	ใช้สำหรับตัดข้อความก่อนเขียนลงไฟล์	<code>file_name.truncate([size])</code>
<code>write()</code>	ใช้สำหรับเขียนข้อความลงไฟล์ แต่ต้องใช้งานร่วมกับเมธอด <code>flush()</code> หรือ <code>close()</code>	<code>file_name.write(str)</code>
<code>writelines()</code>	ใช้สำหรับเขียนข้อความแบบลำดับ (Sequence)	<code>file_name.writelines(sequence)</code>
<code>rename()</code>	ใช้สำหรับเปลี่ยนชื่อไฟล์ที่ต้องการ	<code>os.rename("old_name", "new_name")</code>
<code>remove()</code>	ใช้สำหรับลบไฟล์ที่ต้องการ	<code>os.remove("file_name")</code>

ตาราง 10.3: เมธอดและฟังก์ชันสำหรับการจัดการไฟล์

```
file_objec = open("file_name", mode= "r",
                   buffering=-1)
```

**file\_object** ตัวแปรที่ใช้เก็บค่าข้อมูลไฟล์ เป็นการสร้างออบเจ็คไฟล์ขึ้นมา

**open** เมธอดที่ใช้สำหรับขอเชื่อมต่อเพื่อเปิดใช้งานไฟล์

**file\_name** การระบุตำแหน่งและชื่อไฟล์ (full path or relative path) ที่ต้องการเปิด

**mode** โหมดในการเปิดไฟล์ หากไม่กำหนดจะเป็นการเปิดไฟล์ในโหมดอ่านอย่างเดียว (**r**)

**buffering** กำหนดขนาดหน่วยความจำซึ่วคราวสำหรับพักข้อมูลก่อนนำไปประมวลผล จะกำหนดหรือไม่กำหนดก็ได้ ค่าปกติคือ -1

ถ้ากำหนดเท่ากับ 0 เป็นการระบุไม่ใช้หน่วยความจำ และใช้ได้เฉพาะกับโหมดอ่านนั้น

ถ้ากำหนดเท่ากับ 1 เป็นการอ่านและเขียนไฟล์ทีละ 1 บรรทัด และใช้กับโหมดเทกซ์ไฟล์เท่านั้น

ถ้ากำหนดมากกว่า 1 เป็นการจดหน่วยความจำบัฟเฟอร์ตามขนาดที่กำหนด

เพื่อให้ผู้อ่านได้เห็นภาพการทำงานร่วมกับไฟล์ พิจารณาตัวอย่างต่อไปนี้

- ให้สร้างไดเรกทอรีและไฟล์ขึ้นมาในที่นี้ผู้เขียนสร้างไดเรกทอรีชื่อ **MyDirectory** และสร้างไฟล์ชื่อ **MyFile.txt** ไว้ที่ **~/MyDirectory/MyFile.txt**
- เปิดไฟล์ **MyFile.txt** ขึ้นมาแล้วพิมพ์ข้อความ

Hello there. This is a demo of reading file.

การเขียนคำสั่งโปรแกรมจัดการกับไฟล์ จะมีการระบุโหมดหรือไม่ระบุโหมดการทำงานก็ได้ หากไม่ระบุโหมดค่าเริ่มต้นจะเป็นโหมด `r` เพื่อให้ผู้อ่านได้เห็นความแตกต่างการทำงาน ให้ทดลองเขียนคำสั่งโปรแกรมอ่านและเขียนไฟล์ดังแสดงตามตัวอย่างต่อไปนี้

### ตัวอย่าง 10.1

#### การเขียนโปรแกรมอ่านไฟล์ด้วยเมธอด `read()`

```
1 file = open(r"~/MyDirectory/MyFile.txt") # เชื่อมต่อกับไฟล์ MyFile.txt
2 data = file.read() # อ่านออบเจ็ค file ด้วยเมธอด read()
3 print(data) # แสดงผลลัพธ์จากค่าตัวแปร data
4 file.close() # ใช้เมธอด close() ปิดการเชื่อมต่อการอ่านไฟล์
```

Hello there. This is a demo of reading file.



### ตัวอย่าง 10.2

#### การเขียนโปรแกรมให้เขียนไฟล์ด้วยเมธอด `write()`

```
1 msg = "Python is easy programming language to learn." # ข้อมูลที่ต้องการเขียนลง
   → ไฟล์ใหม่
2 try:
3     # สร้างออบเจ็ค file และสร้างชื่อไฟล์ใหม่ด้วยโหมด w
4     file = open(r"~/MyDirectory/MyFile.txt","w")
5     file.write(msg) # เขียนข้อมูลจากค่าตัวแปร msg ลงออบเจ็ค file
6     file.close() # ปิดการเชื่อมต่อออบเจ็คไฟล์
7 except FileNotFoundError: # ตรวจสอบว่าไฟล์ไม่ถูกต้อง
8     print("Directory Not Found") # แสดงผลลัพธ์เมื่อรับคำแนะนำไฟล์ไม่ถูกต้อง
9 else:
10    print("File Written Successfully") # แสดงผลลัพธ์เมื่อรับคำแนะนำไฟล์ถูกต้อง
```

File Written Successfully



หากระบุตำแหน่งได้เรียบร้อยแล้วจัดเก็บไฟล์ผิด ข้อความในบรรทัดที่ 8 จะแสดงผลการแจ้งเตือนถ้าระบุตำแหน่งได้เรียบร้อยแล้วแต่ไฟล์ไม่ถูกต้อง ข้อความในบรรทัดที่ 10 จะแสดงผลลัพธ์ และเมื่อเปิดไฟล์ตามตำแหน่งได้เรียบร้อยที่ระบุขึ้นมา จะพบไฟล์เพิ่มขึ้นมาอีกหนึ่งไฟล์ชื่อ `AnotherFile.txt` เมื่อเปิดเข้าไปดูผู้อ่านจะพบข้อความที่ได้กำหนดไว้ในตัวแปร `msg`

### ตัวอย่าง 10.3

การเขียนโปรแกรมให้เขียนไฟล์ด้วยเมธอด `write()` และระบุตำแหน่งไฟล์ไม่ถูกต้อง

```
1 msg = "Python is easy programming language to learn." # ข้อมูลที่ต้องการเขียนลง
   ↳ ไฟล์ใหม่
2 try:
3     # สร้างออบเจ็ค file และสร้างชื่อไฟล์ใหม่ด้วยโหมด w
4     file = open(r"~/MyDirectory/AnotherFile.txt","w")
5     file.write(msg) # เขียนข้อมูลจากค่าตัวแปร msg ลงออบเจ็ค file
6     file.close() # ปิดการเขียนต่อออบเจ็คไฟล์
7 except FileNotFoundError: # ตรวจสอบว่ามีไฟล์ที่ต้องการเขียนอยู่ในพื้นที่ที่ระบุไว้
8     print("Directory Not Found") # แสดงผลลัพธ์เมื่อระบุตำแหน่งไฟล์ไม่ถูกต้อง
9 else:
10    print("File Written Successfully") # แสดงผลลัพธ์เมื่อระบุตำแหน่งไฟล์ถูกต้อง
```

Directory Not Found



## ตัวอย่าง 10.4

การเขียนคำสั่งโปรแกรมประยุกต์ใช้งานร่วมกับเมธอด

```
1 file = open(r"~/MyDirectory/MyFile.txt","r") # เปิดไฟล์ขึ้นมาอ่าน
2 print("filePointer =", file.tell()) # แสดงตัวชี้ตำแหน่งเริ่มต้นด้วยเมธอด tell()
3 print(file.read()) # แสดงผลลัพธ์รวม
4 print("filePointer =", file.tell()) # แสดงตัวชี้ตำแหน่งเริ่มต้นด้วยเมธอด tell() อีก
   ↪ ครั้ง
5 file.seek(10) # กำหนดให้อ่านข้อความด้วยเมธอด seek()
6 print(file.read()) # แสดงผลข้อความ
7 file.close() # ปิดการเชื่อมต่อ กับออบเจ็ค file
8 print()

9
10 # เปิดไฟล์ขึ้นมาเขียน หากไม่เพบจะสร้างไฟล์ใหม่
11 file_n = open(r"~/MyDirectory/MyFile.txt","w")
12 # เก็บข้อความลงไฟล์ Myfile_new.txt
13 file_n.write("The most beautiful programming language is Python.")
14
15 file_n.truncate(30) # ตัดข้อความที่จะเก็บลงไฟล์ด้วยเมธอด truncate() แค่ 30 ตัวอักษร
   ↪ แรก
16 file_n.close() # ปิดการเชื่อมต่อ กับออบเจ็ค file_n
17 file_n = open(r"~/MyDirectory/MyFile_New.txt") # เปิดไฟล์ขึ้นมาอ่าน
18 print("Contents of MyFile_New.txt =", file_n.read()) # แสดงผลลัพธ์ข้อความที่
   ↪ อยู่ในออบเจ็ค file_n
19 file_n.close()
```

```
filePointer = 0
Python is easy to learn and powerful programming language.
filePointer = 58
easy to learn and powerful programming language.
```



Contents of MyFile\_new.txt = The most beautiful programming

### 10.3 การอ่านและเขียนไฟล์ไบนาเรีย (Read/Write Binary File)

จากตัวอย่างที่ผ่านมาเราได้ทดลองเขียนคำสั่งโปรแกรมจัดการกับไฟล์ข้อความไปแล้ว การจัดการกับไฟล์ไบนาเรียจะมีลักษณะการเขียนคำสั่งโปรแกรมที่คล้ายกัน ดังตัวอย่างต่อไปนี้

#### ตัวอย่าง 10.5

การเขียนคำสั่งโปรแกรมอ่านและเขียนข้อความไฟล์ไบนาเรีย

```
1 data = b"Python is easy to learn." # สร้างตัวแปร data เก็บข้อความชนิดไบนาเรีย
2 file = open(r"~/MyDirectory/Binary.txt", "wb") # สร้างออบเจ็คไฟล์ชื่อ file ใน
   ↳ โหมด wb
3 file.write(data) # เขียนค่าตัวแปร data เขียนลงไฟล์อوبเจ็ค file
4 file.close() # ปิดการใช้งานต่อ กับออบเจ็ค file
5 file = open(r"~/MyDirectory/Binary.txt", "rb").read() # สร้างออบเจ็คไฟล์ชื่อ
   ↳ file ในโหมด rb
6 print(file) # แสดงผลลัพธ์ในออบเจ็ค file
```

```
b'Python is easy to learn.'
```



### 10.4 เมธอดสำหรับการจัดการไดเรกทอรี (Directory Methods)

นอกจากเราจะจัดการกับไฟล์โดยการเรียกใช้งานเมธอดต่าง ๆ ที่ได้กล่าวมาข้างต้นแล้ว ไดเรกทอรีหรือโฟลเดอร์ก็มีเมธอดสำหรับการจัดการ เช่นกัน แต่เราต้องทำการ import โมดูล os ก่อนเรียกใช้งานเมธอดดูต่อไปนี้

เมธอด	ความหมาย	รูปแบบการใช้งาน
getcwd()	ใช้สำหรับแสดงชื่อไดเรกทอรีที่กำลังใช้งานอยู่	os.getcwd()
mkdir()	ใช้สำหรับสร้างไดเรกทอรี	os.mkdir(dir_name)
chdir()	ใช้สำหรับเข้าไปยังไดเรกทอรีที่ต้องการ	os.chdir(dir_name)
rename()	ใช้สำหรับเปลี่ยนชื่อไดเรกทอรี	os.rename("old_name", "new_name")
rmdir()	ใช้สำหรับลบไดเรกทอรีที่ต้องการ	os.rmdir("dir_name")

ตาราง 10.4: เมธอดสำหรับจัดการกับไดเรกทอรี

### ตัวอย่าง 10.6

การเขียนคำสั่งโปรแกรมการใช้เมธอดจัดการกับไดเรกทอรี

```

1 import os # import โมดูล os เพื่อเรียกใช้คำสั่งจัดการกับไดเรกทอรี
2 print(os.getcwd()) # แสดงตำแหน่งไดเรกทอรี ณ ปัจจุบันที่กำลังใช้งาน
3 try:
4     os.mkdir(r"~/MyDirectory/NewDir") # สร้างไดเรกทอรี
5 except FileNotFoundError:
6     print("This directory already exists") # แสดงผลแล้วเตือนเมื่อสร้างไดเรกทอรีซ้ำ
7 else:
8     os.chdir(r"~/MyDirectory/NewDir") # เปลี่ยนตำแหน่งการใช้งานไดเรกทอรีด้วยเมธอด
9     → chdir()
10    print(os.getcwd()) # แสดงตำแหน่งการเข้าใช้งานไดเรกทอรี
11    file = open(r"~/MyDirectory/NewDir/NewFile.txt", "w") # สร้างไฟล์ใหม่
12    file.write("This is python language.") # เขียนข้อความลงไฟล์
13    file.close() # บันทึกข้อความลงไฟล์และยกเลิกการเชื่อมต่อ กับออบเจ็ค file

```

/Users/username/MyDirectory/  
/Users/username/MyDirectory/NewDir



## 10.5 การตรวจสอบการจัดการไฟล์และไดเรกทอรี

เมื่อเราได้ดำเนินการจัดการกับไฟล์ โดยเริ่มจากตั้งแต่การเปิดไฟล์ การใช้งานใหม่ดอ่านหรือเขียนไฟล์ ตำแหน่งไฟล์ที่ถูกเรียกใช้งานรวมไปถึงการปิดไฟล์ สถานะต่าง ๆ ทั้งหมดนี้สามารถตรวจสอบได้จากเมธอดในตาราง 10.5

### ตัวอย่าง 10.7

การตรวจสอบสถานะจากการเปิดไฟล์ขึ้นมาอ่าน

```
1 file = open(r"~/MyDirectory/MyFile.txt", "r")
2 print("Is writable?", file.writable())
3 print("Is readable?", file.readable())
4 print("Is seekable =", file.seekable)
5 print(" FileMode =", file.mode)
6 print("FileName =", file.name)
7 print("FileStatus =", file.closed)
8 file.close()
9 print("FileStatus =", file.closed)
```

```
Is writable? False
Is readable? True
Is seekable = <built-in method seekable of _io.TextIOWrapper object at
             0x100667d40>
 FileMode = r
 FileName = ~/MyDirectory/MyFile.txt
 FileStatus = False
 FileStatus = True
```

เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>closed()</code>	ใช้ตรวจสอบสถานะการปิดไฟล์	<code>file_name.closed()</code>
<code>mode()</code>	ใช้สำหรับตรวจสอบโหมดการทำงาน	<code>file_name.mode()</code>
<code>name()</code>	ใช้สำหรับตรวจสอบชื่อไฟล์ที่เปิดขึ้นมาอ่านหรือเขียน	<code>file_name.name()</code>
<code>readable()</code>	ใช้สำหรับตรวจสอบการอ่านไฟล์ ถ้าได้คืนค่าเป็น <code>True</code> ถ้าไม่ได้คืนค่าเป็น <code>False</code>	<code>file_name.readable()</code>
<code>seekable()</code>	ใช้สำหรับตรวจสอบการเลื่อนตำแหน่งตัวชี้ ถ้าได้คืนค่าเป็น <code>True</code> ถ้าไม่ได้คืนค่าเป็น <code>False</code>	<code>file_name.seekable()</code>
<code>writable()</code>	ใช้สำหรับตรวจสอบการเขียนไฟล์ ถ้าได้คืนค่าเป็น <code>True</code> ถ้าไม่ได้คืนค่าเป็น <code>False</code>	<code>file_name.writable()</code>

ตาราง 10.5: เมธอดตรวจสอบการจัดการไฟล์และไดเรกทอรี

## 10.6 การอ่านและเขียนไฟล์ด้วยคำสั่ง `with`

คำสั่ง `with` ได้นำมาใช้งานสำหรับการเปิดไฟล์ขึ้นมาอ่านหรือเขียน มีความพิเศษคือไม่ต้องใช้เมธอด `close()` ในการปิดไฟล์ เพราะคำสั่ง `with` จะทำการปิดไฟล์และลบข้อมูลในหน่วยความจำให้อัตโนมัติ เป็นการอำนวยความสะดวกให้กับผู้เขียนโปรแกรมมากขึ้น มีรูปแบบการเขียนคำสั่งโปรแกรมดังนี้

```
with open("file_name", "mode") as file_object
```

`file_name` ไฟล์ที่ต้องการเปิดขึ้นมาอ่านหรือเขียน

`mode` โหมดการอ่านหรือเขียนไฟล์

`file_object` ตัวแปรเก็บออบเจ็คไฟล์

### ตัวอย่าง 10.8

การใช้คำสั่ง with เปิดไฟล์ขึ้นมาอ่านด้วยโหมด "r"

```
1 with open(r"~/MyDirectory/MyFile.txt","r") as file:  
2     print(file.read())
```

Python is easy to learn and powerful programming language.



### ตัวอย่าง 10.9

การใช้คำสั่ง with เปิดไฟล์ขึ้นมาอ่านและเขียน ด้วยโหมด "w+"

```
1 try: # ตรวจสอบข้อติดพลาดของคำสั่งโปรแกรม  
2     with open(r"~/MyDirectory/MyFile.txt", "w+") as file:  
3         file.write("Python is easy to learn.")  
4 except FileNotFoundError:  
5     print("File not found") # แสดงผลเมื่อกำหนดตำแหน่งสร้างไฟล์ไม่ถูกต้อง  
6 else:  
7     print("File written successfully") # แสดงผลเมื่อกำหนดตำแหน่งสร้างไฟล์ถูกต้อง
```

File written successfully



การทำงานในบางครั้งเราต้องมีการจัดเก็บข้อมูลต่าง ๆ ลงไฟล์ เพื่อนำมาใช้งานหรือดูข้อมูลย้อนหลัง เราสามารถจัดเก็บข้อมูลลงไฟล์ให้มีลักษณะคล้ายกับตารางที่ประกอบด้วยแ眷และคอลัมน์ และคุณข้อมูลแต่ละชุดด้วยช่องว่างหรือเครื่องหมายอื่น เช่น เครื่องหมาย comma(,) ตัวอย่างต่อไปนี้เป็นการประยุกต์การทำงานร่วมกับไฟล์เพื่อบันทึกข้อมูล

## ตัวอย่าง 10.10

โปรแกรมบันทึกข้อมูลคะแนนลิงไฟล์

```
1 header = "StudentID, MidTerm, Final, Total, Grade\n"
2 with open(r"~/MyDirectory/Score.txt", "w+") as file:
3     j = 0
4     file.write(header)
5     stu = int(input("ป้อนจำนวนนักศึกษา = "))
6     while j < stu:
7         print(f'นักศึกษาคนที่ {j+1}')
8         stuID = input("รหัสนักศึกษา = ")
9         midterm = float(input("ป้อนคะแนนกลางภาค = "))
10        final = float(input("ป้อนคะแนนปลายภาค = "))
11        total = midterm + final
12        if total < 50:
13            grade = "F"
14        elif total < 60:
15            grade = "D"
16        elif total < 70:
17            grade = "C"
18        elif total < 80:
19            grade = "B"
20        else:
21            grade = "A"
22        text = f'{stuID},{midterm},{final},{total},{grade}\n'
23        file.write(text)
24        print('-'*45)
25        j = j + 1
26 print('Operated successfully')
```

ป้อนจำนวนนักศึกษา = 3  
นักศึกษาคนที่ 1  
รหัสนักศึกษา = 6500001  
ป้อนคะแนนกลางภาค = 23  
ป้อนคะแนนปลายภาค = 44  
-----  
นักศึกษาคนที่ 2  
รหัสนักศึกษา = 6500002  
ป้อนคะแนนกลางภาค = 12  
ป้อนคะแนนปลายภาค = 11  
-----  
นักศึกษาคนที่ 3  
รหัสนักศึกษา = 6500003  
ป้อนคะแนนกลางภาค = 45  
ป้อนคะแนนปลายภาค = 43  
-----  
Operated successfully



กระบวนการทำงานของโปรแกรมตัวอย่างนี้ เมื่อสั่งโปรแกรมให้ทำงาน ค่าตัวแปร header จะถูกเขียนลงไฟล์ก่อน จากนั้นผู้ใช้ป้อนจำนวนนักเรียนที่ต้องการกรอกคะแนน เก็บไว้ที่ตัวแปร stu และนำไปเปรียบเทียบกับค่าตัวแปร j ด้วยคำสั่ง while ในบรรทัดที่ 6 ถ้าค่าตัวแปร j น้อยกว่าค่าตัวแปร stu ก็จะทำงานไปเรื่อย ๆ จนกว่าค่าตัวแปร j จะมีค่ามากกว่าค่าตัวแปร stu ถึงจะหยุดการทำงาน โดยผู้ใช้งานจะมีการกรอกคะแนนสอบกลางภาค (midterm) และ คะแนนสอบปลายภาค (final) จากนั้นโปรแกรมจะนำค่าตัวแปรทั้ง 2 (midterm, final) มารวมกัน เก็บไว้ที่ตัวแปร total ในบรรทัดที่ 11 เพื่อนำไปเปรียบเทียบตัดเกรด จะมีการสรุปการกรอกคะแนนของนักเรียนให้ผู้ใช้งานทราบด้วย ก่อนโปรแกรมจะทำการเขียนข้อมูลบันทึกลงไฟล์ในตำแหน่ง ~/MyDirectory/Score.txt

### ตัวอย่าง 10.11 การเปิดไฟล์ Score.txt ขึ้นมาอ่าน

```
1 with open(r"~/MyDirectory/Score.txt", 'r') as file:  
2     print(file.read())
```

```
StudentID, MidTerm, Final, Total, Grade  
6500001,23.0,44.0,67.0,C  
6500002,12.0,11.0,23.0,F  
6500003,45.0,43.0,88.0,A
```



## สรุปท้ายบท

ในบทนี้ได้ เราได้แนะนำคำสั่งโปรแกรมภาษาไพธอนในการเปิดไฟล์ขึ้นมาอ่านและเขียนไฟล์ในโหมดต่าง ๆ นอกจากนี้เรายังสามารถเรียกใช้งานเมธอดมาตรฐานในภาษาไพธอน เพื่อช่วยให้ผู้เขียนโปรแกรมจัดการกับไฟล์และได้รากหอรีทั้งการสร้าง การลบ การเปลี่ยนชื่อ

## แบบฝึกหัด

- ให้เขียนโปรแกรมบันทึกข้อมูลค่าใช้จ่ายประจำวัน โดยมีการคำนวณค่าใช้จ่ายทั้งหมด รวมทั้งสรุปผลยอดคงเหลือจากเงินที่ได้รับแต่ละวัน
- จงเขียนโปรแกรมสั่งอาหาร โดยมีการบันทึกรายชื่ออาหารที่สั่งจากลูกค้า ราคา สรุปยอดรวมจากการสั่งอาหารแต่ละครั้งลงไฟล์ ภายในโปรแกรมต้องสร้างเมนูแสดงให้เลือกกดหมายเลขอแทน การกรอกชื่อรายการอาหารและราคา รายการอาหารต้องมีอย่างน้อย 3 อย่าง สำหรับราคาอาหาร มี 3 ราคา ขนาดเล็ก 100 บาท, ขนาดกลาง 200 บาท, ขนาดใหญ่ 300 บาท
- จงเขียนโปรแกรมบันทึกหมายเลขโทรศัพท์ลงไฟล์ ประกอบด้วยชื่อ นามสกุล หมายเลขโทรศัพท์ และให้เขียนโปรแกรมค้นหาหมายเลขโทรศัพท์จากการป้อนชื่อ ถ้ามีอยู่ในไฟล์ที่บันทึกไว้ให้ทำการแสดงข้อมูลออกมาทั้งหมด หากไม่พบให้แสดงคำว่า "Not found"

# บทที่ 11

## การใช้งานโมดูลและแพ็คเกจ

ในบทที่ผ่านมาเราได้ศึกษาเกี่ยวกับวิธีการสร้างฟังก์ชันขึ้นมาใช้งาน รวมไปถึงการเรียกใช้งานฟังก์ชันในรูปแบบต่าง ๆ และได้รู้จักกับฟังก์ชัน Built-in ของภาษาไพธอนไปแล้ว ในบทนี้เราจะมาเรียนรู้วิธีการนำเอาฟังก์ชันต่าง ๆ มารวมกัน แล้วสร้างเป็นโมดูลและแพ็คเกจ (Module and Package) ซึ่งทำให้เราสามารถเก็บไว้ใช้ในอนาคตที่น่าจะเห็นจากภาษาไพธอนได้จัดเตรียมไว้ให้ใช้งาน และรวมถึงโมดูลอื่นๆ ที่ได้รับการพัฒนาจากผู้พัฒนาโปรแกรมจากคนอื่น ๆ

### 11.1 การตรวจสอบโมดูลในภาษาไพธอน

โมดูล คือ การรวมกันของคำสั่งโปรแกรมที่ได้เขียนขึ้นมา ซึ่งได้ออกแบบให้ทำหน้าที่อย่างโดยย่างหนาย และภายในโมดูลอาจจะประกอบด้วยคำสั่งโปรแกรม ฟังก์ชัน คลาส จำนวนมาก ขึ้นอยู่กับการออกแบบ การทำหน้าที่ ภาษาไพธอนได้จัดเตรียมโมดูลไว้ให้ผู้ใช้งานเรียกใช้งานอย่างหลากหลาย เราสามารถตรวจสอบได้โดยการใช้คำสั่ง `dir()` หรือเข้าไปที่เว็บไซต์เอกสารสนับสนุนหลักของไพธอน<sup>1</sup> แสดงดังต่อไปนี้

#### ตัวอย่าง 11.1

##### การตรวจสอบโมดูลภายในภาษาไพธอน

```
1 import math  
2  
3 print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__',  
'__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',  
'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees',  
'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',  
'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',  
'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp',  
'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan',  
'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder',  
'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

<sup>1</sup><https://docs.python.org/3/py-modindex.html>

## 11.2 รูปแบบการเรียกใช้งานโมดูลและฟังก์ชัน

ก่อนการใช้งานโมดูลที่ภาษาไพธอนได้จัดเตรียมไว้ หรือจากโมดูลที่ติดตั้งเพิ่มเติม หรือแม้แต่โมดูลที่เราได้สร้างขึ้นมาใช้งานเองก็ตาม เราต้องใช้คำสั่ง **import** ก่อนเสมอ อีกจะเรียกใช้งานฟังก์ชันในโมดูลนั้น ๆ ได้ในส่วนนี้จะอยู่ตัวอย่างวิธีการใช้คำสั่งเรียกใช้งานโมดูลที่มีอยู่ในภาษาไพธอนด้วยรูปแบบต่าง ๆ ซึ่งแบ่งออกได้เป็น 5 รูปแบบ ดังต่อไปนี้

- การใช้งานเฉพาะคำสั่ง **import** เมื่อต้องการเรียกใช้งานฟังก์ชันที่อยู่ภายนอกโมดูล ที่อยู่นอกเหนือจากฟังก์ชัน Built-in ที่ภาษาไพธอนได้จัดเตรียมไว้ให้ เราต้องโหลดโมดูลเข้ามายังโปรแกรมก่อนด้วยคำสั่ง **import** จากนั้นจึงจะอ้างถึงฟังก์ชันภายนอกโมดูลได้ โดยมีรูปแบบการใช้งานดังนี้

```
import module1, module2, ..., module_n
```

```
module1.function_name  
module2.function_name  
module_n.function_name
```

**import**

คำสั่งโหลดโมดูลเข้ามาใช้งานในโปรแกรม

**module1, ..., module\_n**

ชื่อโมดูลที่ต้องการโหลดฟังก์ชันมาใช้งาน

**function\_name**

ชื่อฟังก์ชันภายนอกโมดูล

### ตัวอย่าง 11.2

การเรียกดูโมดูลและฟังก์ชันภายนอกโมดูล `math`, `os`, `time`

```
1 import math, os, time  
2  
3 print(math.pow(2, 3))  
4 print(os.mkdir(r'new_module_dir'))  
5 print(time.time())
```

8.0

None

1621090724.8482409



2. การใช้คำสั่ง **from** ร่วมกับคำสั่ง **import** เมื่อต้องการใช้งานเฉพาะฟังก์ชันที่จำเป็นภายในโมดูล จะใช้คำสั่ง **from** และตามด้วยชื่อโมดูลที่ต้องการใช้งาน หลังคำสั่ง **import** จะเป็นชื่อฟังก์ชันภายในโมดูลนั้นแทน มีรูปแบบการใช้งานดังนี้

```
from module_name import fn_1, ..., fn_n
```

**from**

คำสั่งโหลดโมดูล

**module\_name**

ชื่อโมดูลที่ต้องการโหลดฟังก์ชันเข้ามาใช้งานในโปรแกรม

**import**

คำสั่งเรียกใช้ฟังก์ชันจากโมดูล

**fn\_1, ..., fn\_n**

ชื่อฟังก์ชันภายในโมดูลที่ต้องการเรียกใช้งาน

### ตัวอย่าง 11.3

การใช้คำสั่ง **from ... import ...** และวิธีการเรียกใช้งานฟังก์ชันภายในโมดูล

```
1 from math import pi
2 from os import mkdir
3 from time import time
4
5 print(pi)
6 print(mkdir(r'mydirectories'))
7 print(time())
```

3.141592653589793

None

1621091504.998086



3. การใช้คำสั่ง **from** ร่วมกับคำสั่ง **import \*** มีลักษณะการใช้งานคล้ายกับการใช้คำสั่งแบบที่ 1 แต่แตกต่างกันที่วิธีการเขียนคำสั่งเรียกใช้งาน ซึ่งจะเป็นการโหลดฟังก์ชันที่อยู่ในโมดูลทั้งหมดเข้าสู่โปรแกรม วิธีการนี้จะมีการใช้พื้นที่หน่วยความจำมากกว่าแบบที่ 2 ที่เรียกใช้งานเฉพาะฟังก์ชันที่ต้องการเท่านั้น มีรูปแบบการใช้งานดังนี้

```
from module_name import *
```

**from** คำสั่งเรียกใช้งานโมดูล  
**module\_name** ชื่อโมดูลที่ต้องการโหลดฟังก์ชันเข้ามาใช้  
**import \*** คำสั่งโหลดฟังก์ชันทั้งหมดจากโมดูล

#### ตัวอย่าง 11.4

การใช้คำสั่ง **from ... import \*** โหลดฟังก์ชันทั้งหมดภายในโมดูล

```
1 from math import *
2 r = 15
3
4 print(pi * (r**2))
5 print(pow(2, 5))
6 print(sin(90))
7 print(sin(pi))
```

```
706.8583470577034
32.0
0.8939966636005579
1.2246467991473532e-16
```



4. การใช้คำสั่ง **import** เปลี่ยนชื่อโมดูลด้วยคีย์เวิร์ด **as** การใช้คีย์เวิร์ด **as** เป็นการเปลี่ยนชื่อโมดูลที่เรียกเข้ามาใช้งานในโปรแกรม มีประโยชน์อย่างมากในขณะที่เปลี่ยนคำสั่งโปรแกรม เพราะบางโมดูลมีชื่อยาวหรือมีชื่อที่จำได้ยาก โดยอาจจะเปลี่ยนชื่อโมดูลให้เป็นชื่อใหม่ที่สั้นลงหรือให้จำได้ง่ายขึ้น มีรูปแบบการใช้งานดังนี้

```
import module_name as name
```

**import** คำสั่งโหลดโมดูลเข้ามาใช้  
**module\_name** ชื่อโมดูลที่ต้องการโหลดฟังก์ชันเข้ามาใช้  
**as** คีย์เวิร์ดที่ใช้เปลี่ยนชื่อโมดูลเป็นชื่อใหม่  
**name** ชื่อโมดูลใหม่ที่ตั้งขึ้นมาแทนชื่อโมดูลที่ถูกเรียกใช้

#### ตัวอย่าง 11.5

การใช้คำสั่ง **import ... as ....** เปลี่ยนชื่อโมดูล

```
1 import math as m  
2 r = 15  
3  
4 print(2 * m.pi * r)
```

94.24777960769379



5. การใช้คำสั่ง **from** ร่วมกับคำสั่ง **import** และคีย์เวิร์ด **as** ฟังก์ชันที่ถูกเรียกมาใช้งานจากโมดูลที่ถูกโหลดเข้ามาในโปรแกรมสามารถเปลี่ยนชื่อใหม่ได้ เมื่อกับการเปลี่ยนชื่อโมดูลโดยการใช้คีย์เวิร์ด **as** มีรูปแบบการใช้งานดังนี้

```
from module_n import function_n as name
```

<b>from</b>	คำสั่งโหลดโมดูลเข้ามาใช้งานในโปรแกรม
<b>module_n</b>	ชื่อโมดูลที่ต้องการนำเข้ามาใช้งาน
<b>import</b>	คำสั่งเรียกใช้งานฟังก์ชันจากโมดูล
<b>function_n</b>	ชื่อฟังก์ชันภายในโมดูลที่โหลดเข้ามาใช้งาน
<b>as</b>	คีย์เวิร์ดที่ใช้เปลี่ยนชื่อโมดูลเป็นชื่อใหม่
<b>name</b>	ชื่อฟังก์ชันใหม่ที่ตั้งขึ้นมาแทนชื่อฟังก์ชันที่ถูกเรียกใช้งาน

#### ตัวอย่าง 11.6

การใช้คำสั่ง **from ... import ... as ...** เปลี่ยนชื่อฟังก์ชัน

```
1 from math import pi as PI
2 from math import cos as cosine
3 from math import pow as power
4
5 r = 15
6 print(2 * PI * r)
7 print(cosine(PI))
8 print(power(2, 5))
```

```
94.24777960769379
-1.0
32.0
```



### 11.3 การสร้างโมดูลขึ้นใช้งาน

นอกเหนือจากโมดูลที่ภาษาไพธอนได้จัดเตรียมไว้ให้ใช้งานที่มีอยู่จำนวนมาก เรายังสามารถสร้างโมดูลขึ้นมาใช้งานเองได้เช่นกัน และเนื่องจากภาษาไพธอนเป็นภาษาที่นำมาโดยไม่เสียค่าใช้จ่าย ทำให้มีนักพัฒนาโปรแกรมต่างพากันพัฒนาโมดูลและเผยแพร่ออกมายังช่องทางอื่นๆ เช่น PyPI โมดูลมีลักษณะเหมือนกับไฟล์คำสั่งโปรแกรมภาษาไพธอนที่มีนามสกุล .py เพียงแต่เราไม่ได้นำมาใช้งานโดยตรง แต่เราจะ

เขียนคำสั่งโปรแกรมอ้างถึงโมดูลและเรียกใช้งานฟังก์ชันที่อยู่ภายใต้โมดูลแทน ซึ่งมีประโยชน์อย่างมากในกรณีที่มีผู้พัฒนาโปรแกรมหลายคนและเรียกใช้งานฟังก์ชันซึ่งเดียวกัน ขั้นตอนการสร้างโมดูลมีดังนี้

1. ก่อนอื่นให้เราตรวจสอบ Default Path ซึ่งเป็นส่วนที่ภาษาไพธอนจะทำการค้นหาโมดูล เมื่อเราสร้างขึ้นมาใช้งานและถูกเรียกใช้

### ตัวอย่าง 11.7

#### การตรวจสอบ Default Path ของการเรียกโมดูล

```
1 import sys  
2  
3 print(sys.path)
```

```
[ '/Users/username/Tasks/src', '/Users/username/.pyenv/versions/ 3.9.0/lib/python39.zip', '/Users/username/.pyenv/versions/  
↳ 3.9.0/lib/python3.9', '/Users/username/.pyenv/versions/3.9.0/  
↳ lib/python3.9/lib-dynload', '/Users/username/.pyenv/versions/  
↳ 3.9.0/lib/python3.9/site-packages' ]
```

จากตัวอย่างเราจะเห็นว่ามี Default Path จำนวนมาก ซึ่งเป็น path ที่ภาษาไพธอนจะทำการเรียกโมดูลต่าง ๆ ที่ได้ทำการติดตั้งไว้ หรือเราได้ทำการติดตั้งโมดูลอื่นเพิ่มเติมในภายหลัง เมื่อโหลดโมดูลด้วยการใช้คำสั่ง **import** ไพธอนจะไปค้นหาโมดูลที่อ้างถึงจาก path เหล่านี้

2. สร้างไฟล์ใหม่ขึ้นมาพร้อมทั้งเขียนคำสั่งโปรแกรมให้อยู่ในรูปแบบของฟังก์ชัน ในตัวอย่างนี้ เป็นการสร้างโมดูลแบบง่ายที่ไม่ซับซ้อน

### ตัวอย่าง 11.8

การสร้างไฟล์ใหม่ my\_calculation\_module.py สำหรับทำโมดูล

```
1 # my_calculation_module.py
2
3 def plus(x, y):
4     return x + y
5
6 def minus(x, y):
7     return x - y
8
9 def multiply(x, y):
10    return x * y
11
12 def divide(x, y):
13    return x / y
```

จากตัวอย่างเป็นการสร้างฟังก์ชันการบวก การลบ การคูณ และการหาร โดยมีพารามิเตอร์อยู่รับค่าจากโปรแกรมที่เรียกใช้งาน 2 ค่า คือค่า x และค่า y และส่งค่ากลับไปยังโปรแกรมที่เรียกใช้งานด้วยคำสั่ง **return**

- ให้ตรวจสอบคำสั่งโปรแกรมหรือทดสอบการทำงานเพื่อให้แน่ใจว่า จะไม่เกิดข้อผิดพลาดเมื่อเรียกใช้งาน จากนั้นให้ทำการบันทึกไฟล์โดยใช้ชื่อ `my_calculation_module.py` ไว้ที่ `Default Path` ที่หมายได้ก่อนหน้านี้ (สมมติว่าตำแหน่งไฟล์คือ `/Users/username/Tasks/src/my_calculation_module.py`)
  - ทดสอบการโหลดโมดูลที่ได้สร้างไว้ขึ้นมาใช้งานในโปรแกรม ดังตัวอย่างต่อไปนี้

### ตัวอย่าง 11.9

การเขียนคำสั่งໂຫດໂມດູລາໃຊ້ຈານ

```
1 from my_new_module import *
2
3 print('ผลลัพธ์จากการบวก 10 กับ 15 = ', plus(10, 15))
4 print('ผลลัพธ์จากการลบ 45 กับ 15 = ', minus(45, 15))
5 print('ผลลัพธ์จากการคูณ 10 กับ 15 = ', multiply(10, 15))
6 print('ผลลัพธ์จากการหาร 45 กับ 15 = ', divide(45, 15))
```

ผลลัพธ์จากการบวก 10 กับ 15 = 25  
ผลลัพธ์จากการลบ 45 กับ 15 = 30  
ผลลัพธ์จากการคูณ 10 กับ 15 = 150  
ผลลัพธ์จากการหาร 45 กับ 15 = 3.0



5. เราสามารถตรวจสอบฟังก์ชันต่าง ๆ ภายในໂມດູລາທີ່ສ້າງขື້ນມາໄດ້ ໂດຍຄຳສັ່ງ `dir()` ດັ່ງຕัวອຍ່າງຕ່ອງໄປນີ້

### ตัวอย่าง 11.10

การตรวจสอบฟังก์ชันພາຍໃນໂມດູລາທີ່ສ້າງขື້ນມາໃຊ້ຈານເອງ

```
1 import my_new_module
2
3 print(dir(my_new_module))
```

['\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_',
→ '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_',
→ 'divide', 'minus', 'multiply', 'plus']



## 11.4 การສ້າງແພັກເກົງ

ແພັກເກົງ (Package) ແມ່ນອັກປີໄດ້ຮັກທອຣີ (Directory) ຈຶ່ງເປັນທີ່ເກີບຮັບຮວມຫຼຸດໂມດູລາແລະໄຟລ໌ຄຳສັ່ງໂປຣແກຣມທີ່ຈະຄຸກເຮົາໃຊ້ຈານແລະພາຍໃນໂມດູລັກຈະປະກອບໄປດ້ວຍຝັກໜີ້ຕ່າງໆ ຈຳນວນນັກ ຫື້ນອູ່ກັບຈ່າຍໃຫ້ໂມດູລັນນີ້ທ່າງນາ້ນເກີຍກັບອະໄຮຮູ້ອາຈະແຍກໂມດູລອອກມາເປັນໂມດູລຍ່ອຍໆ ກໍ່ໄດ້ ດັ່ງນັ້ນເພື່ອຄວາມເປັນຮະເບີຍບໍ່ຮູ້ເປັນການຮັບຮວມໄຟລ໌ຄຳສັ່ງໂປຣແກຣມທີ່ທ່ານ໏ທີ່ເໝືອນກັນໃຫ້ອູ່ທີ່ເດືອກນັ້ນ ອາຈະຕ້ອງ

สร้างแพ็คเกจเก็บโมดูลรวมไว้ที่เดียวกัน ภายในแพ็คเกจจะมีไฟล์ชื่อ `__init__.py` ซึ่งเป็นไฟล์เปล่าๆ และจะต้องสร้างขึ้นมาอง โดยมีขั้นตอนดังต่อไปนี้

1. สร้างไดเรกทอรีและตั้งชื่อเป็น `my_package` เก็บไว้ที่

`/Users/username/Tasks/src`

2. จากนั้นสร้างโมดูลขึ้นมาใหม่อีกหนึ่งโมดูล คือ `my_animal_module.py`

### ตัวอย่าง 11.11 การสร้างโมดูล `my_animal_module.py`

```
1 # my_animal_module.py
2
3 def animals_l():
4     animals = ['Dog', 'Cat', 'Duck', 'Bird', 'Snake']
5     return animals
6
7 def animals_s():
8     animals = {'Bear', 'Zebra', 'Giraffe', 'Buffola', 'Rad'}
9     return animals
10
11 def animals_t():
12     animals = {'Chicken', 'Bat', 'Spider', 'Horse', 'Mouse'}
13     return animals
```

3. นำโมดูลทั้งหมดที่สร้าง ซึ่งก็คือ

- `my_calculation_module.py` และ
- `my_animal_module.py`

บันทึกไว้ที่ไดเรกทอรี `/Users/username/Tasks/src/my_package` และสร้างไฟล์เพิ่มขึ้นอีกหนึ่งไฟล์ชื่อ `__init__.py`

จากนั้นสร้างไฟล์เปล่า

- `test_calculation.py` และ
- `test_animal.py`

บันทึกไว้ที่ `/Users/username/Tasks/src`

```
src
└── my_module
    ├── __init__.py
    ├── my_animal_module.py
    └── my_calculation_module.py
└── test_calculation.py
└── test_animal.py
```

4. เขียนคำสั่งโปรแกรมที่ไฟล์ `test_calculation.py` และ `test_animal.py` เพื่อโหลดโมดูลมาใช้งานจากแพ็กเกจที่ได้สร้างไว้ตามตัวอย่างต่อไปนี้

### ตัวอย่าง 11.12

การเขียนคำสั่งโหลดโมดูล `my_calculation` จากแพ็กเกจ `my_module` และการแสดงผลลัพธ์

```
1 # test_calculation.py
2
3 from my_module.my_calculation_module import *
4
5 print('ผลลัพธ์จากการบวก 10 กับ 15 = ', plus(10, 15))
6 print('ผลลัพธ์จากการลบ 45 กับ 15 = ', minus(45, 15))
7 print('ผลลัพธ์จากการคูณ 10 กับ 15 = ', multiply(10, 15))
8 print('ผลลัพธ์จากการหาร 45 กับ 15 = ', divide(45, 15))
```

ผลลัพธ์จากการบวก 10 กับ 15 = 25

ผลลัพธ์จากการลบ 45 กับ 15 = 30

ผลลัพธ์จากการคูณ 10 กับ 15 = 150

ผลลัพธ์จากการหาร 45 กับ 15 = 3.0



### ตัวอย่าง 11.13

การเขียนคำสั่งให้หลอดโมดูล my\_animal จากแฟ้มเก่า my\_module และการแสดงผลลัพธ์

```
1 # test_animal.py
2
3 from my_module.my_animal_module import *
4
5 for animal in animals_l():
6     print(animal, end=', ')
7 print(' ')
8
9 for animal in animals_s():
10    print(animal, end=', ')
11 print(' ')
12
13 for animal in animals_t():
14    print(animal, end=', ')
15 print(' ')
```

Dog, Cat, Duck, Bird, Snake,  
Bear, Giraffe, Zebra, Rad, Buffola,  
Spider, Horse, Chicken, Mouse, Bat,



## 11.5 โมดูลสำหรับในภาษาไพธอน

ภาษาไพธอนได้จัดเตรียมโมดูลที่สำคัญไว้ให้ผู้พัฒนาโปรแกรมเรียกใช้งานจำนวนมาก เช่นโมดูล math, os, sys, time เป็นต้น โดยที่ผู้พัฒนาโปรแกรมไม่ต้องเสียเวลาติดตั้งเพิ่มเติม ในที่นี่เราจะกล่าวถึงโมดูลที่น่าจะมีโอกาสได้นำมาใช้งานบ่อยครั้งมากที่สุดในการพัฒนาโปรแกรมคือ โมดูลคณิตศาสตร์ (Mathematics Module), โมดูลแสดงปฏิกิริทิน (Calendar Module) และโมดูลแสดงเวลา (Time Module)

### 11.5.1 โมดูลคณิตศาสตร์

โมดูล math เป็นหนึ่งในโมดูลที่ภาษาไพธอนได้ติดตั้งไว้ให้เรียกใช้งาน นอกจากระบบยังมีโมดูล cmath ที่ใช้สำหรับการคำนวณจำนวนเชิงซ้อน และภายในโมดูลมีฟังก์ชันต่าง ๆ เช่น ฟังก์ชันการคำนวณพื้นที่รูปทรงเรขาคณิต การหาค่า ln การหาค่าเลขยกกำลัง เป็นต้น

#### ตัวอย่าง 11.14

การเขียนคำสั่งตรวจสอบรายชื่อฟังก์ชันในโมดูล math

```
1 import math
2
3 print(dir(math))

['__doc__', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees',
 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',
 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp',
 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan',
 'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder',
 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

#### ตัวอย่าง 11.15

การเขียนคำสั่งตรวจสอบรายชื่อฟังก์ชันในโมดูล cmath

```
1 import cmath
2
3 print(dir(cmath))

['__doc__', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atanh',
 'cos', 'cosh', 'e', 'exp', 'inf', 'infj', 'isclose', 'isfinite',
 'isinf', 'isnan', 'log', 'log10', 'nan', 'nanj', 'phase', 'pi',
 'polar', 'rect', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau']
```

ฟังก์ชันภายในโมดูล math ของภาษาไพธอน สามารถแยกออกเป็นกลุ่มการใช้งานได้ดังต่อไปนี้ ทั้งนี้เราสามารถดูฟังก์ชันที่อาจมีการเพิ่มเติมในอนาคต<sup>2</sup>

<sup>2</sup><https://docs.python.org/3/library/math.html>

## ฟังก์ชันการคำนวณตัวเลขและการแทนค่า

ฟังก์ชันการคำนวณตัวเลขและการแทนค่า (Numeric-gheoretic and representation functions) ในโมดูล math ที่สำคัญและพบได้บ่อยจะนำเสนอดังตารางต่อไปนี้

ฟังก์ชัน/รูปแบบการใช้งาน	ความหมาย	คำอธิบายเพิ่มเติม
<code>math.ceil(x)</code>	จำนวนเต็มที่น้อยที่สุดที่ไม่ต่ำกว่า $x$ (ceiling)	$x$ คือ จำนวนเต็มหรือจำนวนจริง
<code>math.copysign(x,y)</code>	ส่งกลับค่าของ $x$ เป็นจำนวนจริง และใส่เครื่องหมายตามค่าของ $y$	$x$ คือ จำนวนเต็มหรือจำนวนจริง $y$ คือ จำนวนเต็มหรือจำนวนจริง
<code>math.fabs(x)</code>	หาค่า float absolute คืนค่ากลับมาเป็นชนิดข้อมูลทางคณิต	$x$ จำนวนเต็มหรือจำนวนจริง
<code>math.factorial(x)</code>	หาค่า factorial ของ $x$	$x$ คือ ตัวเลขจำนวนจริง
<code>math.floor(x)</code>	หาจำนวนเต็มที่มีค่ามากที่สุด	$x$ คือ ตัวเลขจำนวนจริง
<code>math.fmod(x,y)</code>	หาค่าเศษจากการหาร	$x$ คือ ตัวตั้ง $y$ คือ ตัวหาร
<code>math.frexp(x)</code>	หาค่า mantissa exponent	$x$ คือ จำนวนจริง
<code>math.fsum(x)</code>	หาค่าผลรวมจากชนิดข้อมูลแบบเรียงลำดับ	$x$ คือ ชนิดข้อมูลแบบลำดับ
<code>math.gcd(x,y)</code>	หาค่าตัวหารร่วมนากที่มีค่านากที่สุด	$x, y$ คือ จำนวนเต็มใดๆ
<code>math.isclose(x,y)</code>	วัดความใกล้เคียงกันของค่า floating point คืนค่ากลับเป็น <code>True</code> และ <code>False</code>	$x, y$ คือ จำนวนจริง ที่ต้องการนำมาเปรียบเทียบ
<code>math.isfinite(x)</code>	ตรวจสอบว่าเป็นจำนวนจำกัด (finite number) หรือไม่ ถ้าใช่คืนค่า <code>True</code> ถ้าไม่ใช่คืนค่า <code>False</code>	$x$ คือ จำนวนจริง ที่ต้องการตรวจสอบ

ตาราง 11.1: ฟังก์ชันการคำนวณตัวเลขและการแทนค่า (มีต่อ)

ฟังก์ชัน/รูปแบบการใช้งาน	ความหมาย	คำอธิบายเพิ่มเติม
<code>math.isinf(x)</code>	ตรวจสอบเป็นจำนวนอนันต์ (infinity number) ทั้งจำนวนเต็มบวก และเต็มลบ ถ้าใช่คืนค่า <code>True</code> ถ้าไม่ใช่คืนค่า <code>False</code>	$x$ คือ จำนวนจริง ที่ต้องการตรวจสอบ
<code>math.isnan(x)</code>	ตรวจสอบเป็นค่า <code>Nan</code> (Not a Number) หรือไม่ ถ้าใช่คืนค่า <code>True</code> ถ้าไม่ใช่คืนค่า <code>False</code>	$x$ คือค่าใด ๆ ที่ต้องการตรวจสอบ
<code>math.frexp(x, i)</code>	หาค่า $x * (2**i)$ ซึ่งเป็นค่า ตรงข้ามของฟังก์ชัน <code>fmod()</code>	$x, i$ คือจำนวนเต็มบวก หรือจำนวนเต็มลบ
<code>math.modf(x)</code>	แยกจำนวนเต็มและเศษของ จำนวนจริงออกจากกัน	$x$ คือ จำนวนจริง
<code>math.trunc(x)</code>	ตัดเศษทศนิยมทิ้ง (truncate) ตาม จำนวนหลักที่ระบุ	$x$ คือ จำนวนจริงที่ต้องการตัดเศษทิ้ง

ตาราง 11.1: ฟังก์ชันการคำนวณตัวเลขและการแทนค่า

### ตัวอย่าง 11.16

เปรียบเทียบการหาค่าเศษจากการหารด้วยฟังก์ชัน `fmod()` และคำสั่งการหารเอาเศษในภาษาไพธอน

```

1 import math
2
3 x = -1e-100
4 y = 1e100
5 print('fmod(x, y) = ', math.fmod(x, y))
6 print('x % y = ', x % y)

```

```

fmod(x, y) = -1e-100
x % y = 1e+100

```



### ตัวอย่าง 11.17

เปรียบเทียบการหาค่าผลรวมสมาชิกในลิสต์

```
1 import math  
2  
3 x = [0.1] * 10  
4 print('x = ', x)  
5 print('sum(x) = ', sum(x))  
6 print('math.fsum(x) = ', math.fsum(x))
```

```
x = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]  
sum(x) = 0.9999999999999999  
math.fsum(x) = 1.0
```



### ตัวอย่าง 11.18

การเขียนคำสั่งเรียกใช้งานกลุ่มฟังก์ชันการคำนวณและการแทนค่า

```
1 import math  
2  
3 print('ค่าจำนวนเต็มของ 15.4588 =', math.ceil(15.4588))  
4 print('แปลงค่าจำนวนเต็มลบ -15 =', math.copysign(-15, 0.0))  
5 print('ค่า factorial(10) =', math.factorial(10))  
6 print('ค่า ldexp(20, 5) =', math.ldexp(20, 5))  
7  
8 lst = [1.5, 3.5, 5.5, 1.2, 4.5]  
9 print('ค่า fsum(lst) = ', math.fsum(lst))
```

```
ค่าจำนวนเต็มของ 15.4588 = 16  
แปลงค่าจำนวนเต็มลบ -15 = 15.0  
ค่า factorial(10) = 3628800  
ค่า ldexp(20, 5) = 640.0  
ค่า fsum(lst) = 16.2
```



## ฟังก์ชันที่กำลังและลอการิทึม

ฟังก์ชันเลขยกกำลังและลอการิทึมภายในโมดูล math ที่สำคัญเราดูได้จากตารางต่อไปนี้

ฟังก์ชัน/รูปแบบการใช้	ความหมาย	คำอธิบายเพิ่มเติม
<code>math.exp(x)</code>	ค่า $e$ ยกกำลัง $x$	$x$ คือ จำนวนจริง และ $e$ คือ Euler's number มีค่า <b>2.718281...</b>
<code>math.expm1(x)</code>	ค่า $\text{math.exp}(x) - 1$	$x$ คือ จำนวนจริง
<code>math.log(x[, base])</code>	ค่าลอการิทึม (logarithm) ของ $x$	$x$ คือ จำนวนจริง ที่มากกว่า <b>0</b> $base$ คือฐานของลอการิทึม ถ้าไม่กำหนดจะเป็นฐาน $e$
<code>math.log1p(x)</code>	ค่า $\text{math.log}(1+x)$ แม่นยำเมื่อ $x$ เข้าใกล้ <b>0</b>	$x$ คือ จำนวนจริง ที่มากกว่า <b>0</b>
<code>math.log2(x)</code>	หาค่า $\text{math.log}(x, 2)$	$x$ คือ จำนวนจริง ที่มากกว่า <b>0</b>
<code>math.log10(x)</code>	ค่า $\text{math.log}(x, 10)$	$x$ คือ จำนวนจริง ที่มากกว่า <b>0</b>
<code>math.pow(x, y)</code>	ค่า $x$ ยกกำลัง $y$	$x$ และ $y$ คือ จำนวนจริง ฟังก์ชันนี้ไม่นิยามเมื่อ $x < 0$ และ $y$ ไม่เป็นจำนวนเต็ม
<code>math.sqrt(x)</code>	ค่ารากที่สองของ $x$	$x$ คือ จำนวนจริง ที่ไม่ต่ำกว่า <b>0</b>

ตาราง 11.2: ฟังก์ชันเลขยกกำลังและลอการิทึมภายในโมดูล math

### ตัวอย่าง 11.19

การเขียนคำสั่งเรียกใช้งานกลุ่มฟังก์ชันเลขยกกำลังและลอการิทึมของโมดูล math

```
1 import math
2 print('ค่า expm1(5.5) =', math.expm1(5.5))
3 print('ค่า log(5) =', math.log(5))
4 print('ค่า log10(2) =', math.log(2))
5 print('ค่า log1p(2) =', math.log1p(2))
6 print('ค่า log2(10) =', math.log2(10))
7 print('ค่า pow(10, 3) =', math.pow(10, 3))
8 print('ค่า sqrt(27) =', math.sqrt(27))
```

```
ค่า expm1(5.5) = 243.69193226422038
ค่า log(5) = 1.6094379124341003
ค่า log10(2) = 0.6931471805599453
ค่า log1p(2) = 1.0986122886681098
ค่า log2(10) = 3.321928094887362
ค่า pow(10, 3) = 1000.0
ค่า sqrt(27) = 5.196152422706632
```



### 11.5.2 โมดูลแสดงผลวันที่และเวลา

การแสดงผลวันที่และเวลาเป็นส่วนหนึ่งที่สำคัญในการพัฒนาโปรแกรม ซึ่งผู้เขียนจะขอกล่าวถึงโมดูล calendar และโมดูล time และแสดงตัวอย่างการเรียกใช้งานฟังก์ชันบางตัวที่มีอยู่ภายในทั้งสองโมดูล

#### โมดูลปฏิทิน (Calendar Module)

โมดูล calendar ใช้สำหรับแสดงผลวัน เดือน ปี เราสามารถเรียกใช้งานฟังก์ชันและแอ็ตทริบิวต์ได้ตามตารางต่อไปนี้

ชื่อฟังก์ชัน	ความหมาย	รูปแบบการใช้และคำอธิบาย
<code>calendar()</code>	แสดงผลปฏิทินตามปีค.ศ. ที่กำหนด	<code>calendar.calendar(year, w=1, l=1, c=3)</code> โดย <code>year</code> คือ ปีที่ต้องการแสดงผล, <code>w</code> คือ ระยะซ่องไฟการแสดงผลของวันที่, <code>l</code> คือ ระยะบรรทัดของแต่ละสัปดาห์, <code>c</code> คือ ระยะห่างของแต่ละเดือน
<code>firstweekday()</code>	แสดงวันเริ่มต้นในสัปดาห์ ค่าเริ่มต้นเป็น 0 คือวันจันทร์ และ 6 คือวันอาทิตย์	<code>calendar.firstweekday()</code>
<code>isleap()</code>	ตรวจสอบปีที่มี 366 วัน ถ้าใช่คือค่า <code>True</code> ถ้าไม่ใช่คืนค่า <code>False</code>	<code>calendar.isleap(year)</code> <code>year</code> คือ ปีที่ต้องการตรวจสอบว่ามี 366 วัน (leap year) หรือไม่
<code>leapdays()</code>	แสดงจำนวนตัวเลข ปีที่มี 366 วัน โดยกำหนดพารามิเตอร์เป็นช่วงของปี	<code>calendar.leapdays(y1, y2)</code> <code>y1</code> คือ ปีเริ่มต้น, <code>y2</code> คือ ปีสิ้นสุด
<code>month()</code>	แสดงเดือนที่ปฏิทินตามปีค.ศ.ที่กำหนด	<code>calendar.month(year, month, w=2, l=1)</code> โดย <code>year</code> คือ ปีที่ต้องการแสดงผล, <code>month</code> คือ เดือนที่ต้องการแสดงผล, <code>w</code> คือ ระยะซ่องไฟการแสดงผลของวันที่, <code>l</code> คือ ระยะบรรทัดของแต่ละสัปดาห์
<code>monthcalendar()</code>	แสดงเดือนในปฏิทินปีค.ศ. ที่กำหนด และแสดงผลในรูปแบบลิสต์	<code>calendar.monthcalendar(year, month)</code> โดย <code>year</code> คือ ปีที่ต้องการแสดงผล, <code>month</code> คือเดือนที่ต้องการแสดงผล

ตาราง 11.3: ฟังก์ชันภายในโมดูล `Calendar` (มีต่อ)

ชื่อฟังก์ชัน	ความหมาย	รูปแบบการใช้และคำอธิบาย
monthrange()	แสดงรหัสวันเริ่มต้นของเดือน 0 คือวันจันทร์ และ 6 คือวันอาทิตย์ และจำนวนวันของแต่ละเดือน	calendar. monthrange(year, month) year คือ ปีที่ต้องการแสดงรหัสวันเริ่มต้นของเดือน, month คือ เดือนที่ต้องการแสดงรหัสวันเริ่มต้น
prcal()	แสดงผลปฏิทินตามปี พ.ศ. หรือ ค.ศ. เมื่อong กับฟังก์ชัน calendar	calendar.prCal(year, w=2, l=1, c=6)
prmonth()	แสดงเดือนในปฏิทินตามปี พ.ศ. หรือค.ศ. เมื่อong กับฟังก์ชัน month	calendar.prCal(year, w=2, l=1, c=6)
setfirstweekday()	กำหนดวันเริ่มต้นของสัปดาห์ กำหนดตามรหัสวัน 0 คือ วันจันทร์ และ 6 คือวันอาทิตย์	calendar. .setfirstweekday(weekday) weekday คือ รหัสวันในสัปดาห์ กำหนดเป็น 0-6
timegm()	แสดงค่าเวลาเริ่มจากปี ค.ศ. 1970 จากฟังก์ชัน gmtime() ของโมดูล time	calendar.timegm(tuple) tuple คือ รูปแบบเวลาที่ได้จากการคืนค่ากลับมาจากการฟังก์ชัน gmtime()
weekday()	แสดงรหัสวันในสัปดาห์ จากการกำหนดปี ค.ศ. เดือน และวันที่	calendar.weekday(year, month, day) year คือ ปีที่ต้องการแสดงรหัสวันกำหนดได้ตั้งแต่ปี 1970 ถึงปีปัจจุบัน, month คือ เดือนที่ต้องการแสดงรหัสวันกำหนดเป็น 1-12, day คือ วันที่ต้องการแสดงรหัสวันกำหนดเป็น 1-31

ตาราง 11.3: ฟังก์ชันภายในโมดูล Calendar

ชื่อแอ็ตทริบิวต์	ความหมาย	รูปแบบการใช้และคำอธิบาย
day_name	ใช้สำหรับแสดงชื่อวันแบบเต็ม	calendar.day_name
day_abbr	แสดงชื่อวันรูปแบบย่อ	calendar.day_abbr
month_name	สำหรับแสดงชื่อดือนแบบเต็ม	calendar.month_name
month_abbr	แสดงชื่อดือนรูปแบบย่อ	calendar.month_abbr

ตาราง 11.4: แอ็ตทริบิวต์ภายในโมดูล Calendar

### ตัวอย่าง 11.20

การเขียนคำสั่งเรียกใช้งานฟังก์ชันภายในโมดูล calendar

```

1 import calendar
2 print('วันเริ่มต้นในสปดาห์ = ', calendar.firstweekday())
3 print('ปี 2017 มี 366 วัน = ', calendar.isleap(2017))
4 print('วันแรกและจำนวนวัน เดือนมกราคม ปี 2017 = ', calendar.monthrange(2017,
   → 1))
5 print('ปี 2017 เดือน 1 วันที่ 31 ตรงรหัสวัน = ', calendar.weekday(2017, 1, 31))

```

วันเริ่มต้นในสปดาห์ = 0  
 ปี 2017 มี 366 วัน = False  
 วันแรกและจำนวนวัน เดือนมกราคม ปี 2017 = (6, 31)  
 ปี 2017 เดือน 1 วันที่ 31 ตรงรหัสวัน = 1



### ตัวอย่าง 11.21

การเขียนคำสั่งเรียกใช้แอ็ตทริบิวต์ภายในโมดูล calendar

```

1 import calendar
2
3 for day in calendar.day_name:
4     print(day, end=' ')
5 print()
6 print(list(calendar.day_abbr))

```

Monday Tuesday Wednesday Thursday Friday Saturday Sunday  
 ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']



## โมดูลเวลา

โมดูล time ใช้สำหรับแสดงผลเวลา ในส่วนนี้จะขอแนะนำฟังก์ชันบางตัว หากเรายากทราบว่าในโมดูล time มีฟังก์ชันอะไรบ้าง ให้ใช้คำสั่ง `help(time)` ตรวจสอบฟังก์ชันได้ หรือเข้าไปที่เว็บไซต์เอกสารสนับสนุนหลักของเพรอน<sup>3</sup> มีฟังก์ชันให้เรียกใช้งานตามตารางต่อไปนี้

### ตัวอย่าง 11.22

การเขียนคำสั่งเรียกใช้แอตทริบิวต์ภายใน โมดูล calendar

```
1 import time
2
3 print(time.asctime())
4 print(time.asctime(time.gmtime()))
5 print(time.asctime(time.localtime()))
6 print(time.strftime('%a %d %b %Y %H:%M:%S'))
7 print(time.strftime('%a %d %b %Y %H:%M:%S', time.gmtime()))'
```

```
Tue Sep 10 16:52:49 2019
Tue Sep 10 09:52:49 2019
Tue Sep 10 16:52:49 2019
5a 10 Sep 2019 16:52:49
5a 10 Sep 2019 09:52:49
```



### ตัวอย่าง 11.23

การเขียนคำสั่งเรียกใช้งานฟังก์ชัน strftime() ในโมดูล time

```
1 import time
2
3 print(time.strftime('%c %Z'))
4 print(time.strftime('%a, %d %b %Y %H:%M:%S %Z', time.gmtime()))
```

```
Tue Sep 10 16:53:23 2019 SE Asia Standard Time
Tue, 10 Sep 2019 09:53:23 SE Asia Standard Time
```



<sup>3</sup><https://docs.python.org/3/library/time.html?highlight=time#module-time>

ชื่อฟังก์ชัน	ความหมาย	รูปแบบการใช้
<code>asctime()</code>	แสดงวัน เดือน ปี และเวลา	<code>time.asctime([t])</code> t คือ ค่าที่ได้จากฟังก์ชัน <code>gmtime()</code> หรือ <code>localtime()</code>
<code>clock()</code>	แสดงผลเวลาประมาณผลของ CPU แสดงผลออกมาเป็นหน่วยวินาที	<code>time.clock()</code>
<code>ctime()</code>	แสดงเวลาปัจจุบันตาม location	<code>time.ctime([secs])</code> secs คือ วินาทีเริ่มนับจากปี 1970 กำหนดหรือไม่ก็ได้
<code>gmtime()</code>	แสดงเวลาปัจจุบันในรูปแบบโซนเวลา UTC (Coordinated Universal Time: UTC หรือ Greenwich Mean Time: GMT)	<code>time.ctime([secs])</code> secs คือ วินาทีเริ่มนับจากปี 1970 กำหนดหรือไม่ก็ได้
<code>localtime()</code>	แสดงเวลาปัจจุบันในรูปแบบ <code>struct_time</code> ตามโซนเวลาของผู้ใช้งาน	<code>time.ctime([secs])</code> secs คือ วินาทีเริ่มนับจากปี 1970 กำหนดหรือไม่ก็ได้
<code>sleep()</code>	ใช้สำหรับหน่วงเวลา	<code>time.sleep([t])</code> t คือ เวลาที่ต้องการหน่วงมีหน่วยเป็นวินาที
<code>strftime()</code>	จัดรูปแบบแสดงผลเวลาของฟังก์ชัน <code>gmtime()</code> และ <code>localtime()</code>	<code>time.strftime(format[, t])</code> format คือ การจัดรูปแบบแสดงผลเวลา และ t คือ ฟังก์ชัน <code>gmtime()</code> หรือ <code>localtime()</code> ถ้าไม่กำหนดจะแสดงเวลาที่ได้จากฟังก์ชัน <code>localtime()</code>

ตาราง 11.5: ฟังก์ชันภายในโมดูล time

สัญลักษณ์	ความหมาย
%a	แสดงชื่อของวัน เช่น Sun, Mon, Tue, ...
%A	แสดงชื่อของวัน เช่น Sunday, Monday, Tuesday, ...
%b	แสดงชื่อของเดือน เช่น Jan, Feb, Mar, ...
%B	แสดงชื่อของเดือน เช่น January, February, March, ...
%c	แสดงวัน เดือน ปี และเวลา ตามตำแหน่งที่อยู่ของผู้เรียกใช้
%d	แสดงวันที่ของเดือน [01-31]
%H	แสดงตัวเลขชั่วโมงในรูปแบบเวลา 24 ชั่วโมง [00-24]
%I	แสดงตัวเลขชั่วโมงในรูปแบบเวลา 12 [01-12]
%j	แสดงจำนวนวันของปี ณ ปัจจุบัน [001-366]
%m	แสดงตัวเลขเดือน [01-12]
%M	แสดงตัวเลขนาที [00-59]
%p	แสดงเวลาแบบ AM หรือ PM ตามตำแหน่งของผู้ใช้งาน
%S	แสดงตัวเลขวินาที [00-61]
%U	แสดงสัปดาห์ปัจจุบันของปี [00-53] โดยที่วันอาทิตย์เป็นวันแรกของสัปดาห์
%w	แสดงตัวเลขประจำวัน 0 คือ วันอาทิตย์ และ 6 คือวันเสาร์
%W	แสดงสัปดาห์ปัจจุบันของปี [00-53] โดยที่วันจันทร์เป็นวันแรกของสัปดาห์
%x	แสดงรูปแบบวัน/เดือน/ปี [09/11/17]
%X	แสดงเวลาในรูปแบบ H:M:S [11:13:48]
%y	แสดงปี ค.ศ. แบบบิ๊ก
%Y	แสดงปี ค.ศ. แบบเต็ม
%z	แสดงโซนเวลา
%Z	แสดงชื่อโซนเวลา
%%	แสดงเครื่องหมาย %

ตาราง 11.6: สัญลักษณ์การจัดรูปแบบแสดงผลเวลา

## สรุปท้ายบท

ในบทนี้เราได้รู้จักวิธีการสร้างโมดูลและแพ็กเกจไว้ใช้งาน ซึ่งภายในโมดูลประกอบด้วยฟังก์ชัน คลาสที่ทำหน้าที่แตกต่างกันออกไปตามที่กำหนด และภายในแพ็กเกจจะประกอบด้วยโมดูลที่สร้างขึ้นมา อย่างไรก็ตามเราต้องสร้างไฟล์ `__init__.py` ไว้ภายในแพ็กเกจด้วยถึงจะเรียกใช้งานได้ นอกจากนี้ยังได้รู้จักกับโมดูลมาตรฐานของภาษาไพธอนที่ได้จัดเตรียมไว้ให้ผู้ใช้เรียกใช้งาน คือ โมดูล `math` และ `cmath` ซึ่งเป็นโมดูลทางด้านการคำนวนคณิตศาสตร์ รวมทั้งโมดูลการจัดการแสดงผลลัพธ์ เดือน ปี และเวลา คือ โมดูล `calendar` และ `time`

## แบบฝึกหัด

1. จงเขียนโปรแกรมสร้างโมดูลคิดอัตราส่วนลดสินค้า จากการขายสินค้าจำนวน 3 อย่าง ถ้าลูกค้าซื้อสินค้ารวมเป็นเงิน 1500 บาทขึ้นไป ได้ส่วนลด 5% ถ้าลูกค้าซื้อสินค้ารวมเป็นเงิน 2500 บาทขึ้นไป มีส่วนลด 10% ถ้าลูกค้าซื้อสินค้ารวมเป็นเงินมากกว่า 5000 บาทขึ้นไป ได้ส่วนลด 15% โดยให้ป้อนราคาสินค้าทั้ง 3 อย่างผ่านทางคีย์บอร์ด พร้อมทั้งให้แสดงราคาสินค้าแต่ละชิ้นที่ลูกค้าซื้อ รวม ราคาส่วนลด และราคาที่ต้องจ่ายทั้งหมด
2. จงเขียนโปรแกรมคำนวณค่าอาหารบุฟเฟต์ คิดอัตราผู้ใหญ่คนละ 199 บาท เด็กคนละ 159 บาท บันทึกลงไฟล์ ภายในไฟล์ต้องบอกวันที่และเวลา จำนวนคน โดยให้แยกจำนวนเด็กและผู้ใหญ่ที่เข้ามารับประทานที่ร้าน