

เอกสารประกอบการสอนรายวิชา
477-201 การเขียนโปรแกรมคอมพิวเตอร์
การเขียนโปรแกรมภาษา Python เบื้องต้น
(Basic Python Programming)

ดร. จันทวรรณ ปิยะวัฒน์

สาขาวิชาระบบสารสนเทศทางธุรกิจ
ภาควิชาบริหารธุรกิจ คณะวิทยาการจัดการ
มหาวิทยาลัยสงขลานครินทร์
ภาคการศึกษาที่ 1/2563

คำนำ

เอกสารประกอบการสอนเล่มนี้จัดทำขึ้นสำหรับการสอนรายวิชา 477-201 การเขียนโปรแกรมคอมพิวเตอร์ (Computer Programming) ในภาคการศึกษาที่ 1 ปีการศึกษา 2563 ซึ่งเป็นรายวิชาบังคับของนักศึกษาหลักสูตรระบบสารสนเทศทางธุรกิจ ชั้นปีที่ 2 ภาควิชาบริหารธุรกิจ คณะวิทยาการจัดการ มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่ จำนวน 3 หน่วยกิต 3(2-2-5) เป็นการสอนทฤษฎี 2 ชั่วโมงต่อสัปดาห์ ปฏิบัติ 2 ชั่วโมงต่อสัปดาห์ และนักศึกษาควรศึกษาค้นคว้าด้วยตัวเอง 5 ชั่วโมงต่อสัปดาห์

วิชา 477-201 การเขียนโปรแกรมคอมพิวเตอร์ มีจุดมุ่งหมายให้นักศึกษาได้มีพื้นฐานความรู้ความเข้าใจในหลักการเขียนโปรแกรมคอมพิวเตอร์เบื้องต้นด้วยภาษา Python ส่วนประกอบต่างๆของโปรแกรมคอมพิวเตอร์ และภาษา Python สามารถเขียนโปรแกรมคอมพิวเตอร์พื้นฐานด้วยภาษา Python ได้ตามการวิเคราะห์และออกแบบขั้นตอนการทำงานของโปรแกรมอย่างมีระบบ สามารถเขียนโปรแกรมแบบมีเงื่อนไขเพื่อการตัดสินใจ เขียนคำสั่งเพื่อให้โปรแกรมทำงานวนซ้ำได้ และเข้าใจการใช้งานโมดูลส่วนเสริมต่างๆ ของโปรแกรมภาษา Python เพื่อนำความรู้เหล่านี้ไปใช้ในการเขียนโปรแกรมระดับในระดับที่ยากขึ้นซึ่งได้แก่ การเขียนโปรแกรมแบบฟังก์ชันและการเขียนโปรแกรมเชิงวัตถุได้

หนังสือเล่มนี้ได้จัดแบ่งเนื้อหาออกเป็น 11 บท ในแต่ละบทจะมีแบบฝึกหัดท้ายบทเพื่อให้ผู้เรียนได้ลองวิเคราะห์และออกแบบแนวทางแก้ไขปัญหาและพัฒนาออกมาเป็นโปรแกรมด้วยภาษา Python ที่ได้เรียนรู้ไปแล้วได้ ทั้งนี้ผู้จัดทำหวังเป็นอย่างยิ่งว่าเอกสารประกอบการสอนฉบับนี้จะให้ความรู้และเป็นประโยชน์แก่ผู้เรียนและผู้อ่านทุกๆ ท่าน เพื่อสร้างความรู้ความเข้าใจในการฝึกเขียนโปรแกรมคอมพิวเตอร์เบื้องต้นให้ดียิ่งขึ้น หากมีข้อเสนอแนะประการใด ผู้จัดทำขอรับไว้ด้วยความขอบพระคุณยิ่ง

ดร.จันทวรรณ ปิยะวัฒน์

สาขาวิชาระบบสารสนเทศทางธุรกิจ

ภาควิชาบริหารธุรกิจ คณะวิทยาการจัดการ

มหาวิทยาลัยสงขลานครินทร์

สารบัญ

คำนำ	iii
1 ความรู้เบื้องต้นเกี่ยวกับ Python	1
1.1 Python คืออะไร	1
1.2 Python ทำงานอย่างไร	2
1.3 อัลกอริทึม (Algorithm) และ ผังงาน (Flowchart)	3
1.4 การติดตั้งโปรแกรม Python Runtime	4
2 ส่วนประกอบต่างๆ ของภาษา Python	9
2.1 ตัวแปร (Variables)	9
2.2 การตั้งชื่อตัวแปร	9
2.3 การตั้งชื่อตัวแปรพร้อมกันหลายตัวแปร	10
2.4 คำสงวน (Keywords)	10
2.5 เลขประจำตัวตำแหน่งของตัวแปร	11
2.6 ชนิดของข้อมูล (Types)	11
2.7 เครื่องหมายสำหรับการคำนวณทางคณิตศาสตร์	13
2.7.1 การคำนวณทางคณิตศาสตร์ (Arithmetic Operators)	13
2.7.2 รูปแบบการเขียนการคำนวณทางคณิตศาสตร์แบบย่อ	14
2.7.3 การจัดการข้อความด้วยเครื่องหมายทางคณิตศาสตร์	15
2.8 Expressions และ Statements	15
2.9 การเขียนข้อความอธิบายโปรแกรมโดยใช้ Comment	16
2.10 Source Code	16
2.11 คำสั่ง print (ตัวแปรหรือข้อมูล)	17
2.12 การใช้คำสั่ง input() รับค่าจากแป้นพิมพ์	17

2.13	แบบฝึกหัด	18
3	ประโยคเงื่อนไขในภาษา Python (Conditional Statements)	21
3.1	การเปรียบเทียบค่า (Boolean Expressions)	21
3.2	ตัวดำเนินการทางตรรกศาสตร์ (Logical Operators)	22
3.3	การใช้คำสั่ง if เพื่อเลือกเงื่อนไข	24
3.4	การใช้ if กับ else	24
3.5	Chained Expressions	25
3.6	Nested Expressions	25
3.7	แบบฝึกหัด	26
4	การเขียนและใช้งานฟังก์ชัน (Functions)	27
4.1	การเรียกใช้ฟังก์ชัน	27
4.2	การเรียกใช้โมดูล (Modules)	27
4.3	การสร้างฟังก์ชันในภาษา Python	28
4.4	ขอบเขตของตัวแปร	29
4.5	ฟังก์ชัน return	30
4.6	การคืนค่าจากฟังก์ชัน	30
4.7	การเขียนโปรแกรมเชิงฟังก์ชัน (Functional Programming)	31
4.8	ฟังก์ชันที่เรียกตัวเอง (Recursion)	32
4.9	แบบฝึกหัด	33
5	การใช้ประโยคสั่งทำงานวนซ้ำ	35
5.1	ฟังก์ชัน range()	35
5.2	คำสั่ง for	35
5.3	คำสั่ง while	36

5.4	แบบฝึกหัด	39
6	การใช้งาน String	41
6.1	ความหมายของ String	41
6.2	ฟังก์ชัน <code>len()</code>	41
6.3	การเดินทางตามตัวชี้ของ String	42
6.4	การตัดคำใน String	42
6.5	โครงสร้างข้อมูลที่ไม่เปลี่ยนแปลงไม่ได้	43
6.6	การค้นหาคำใน String	44
6.7	เมธอดของ String (String Methods)	45
6.8	<code>in</code> โอเปอเรเตอร์	45
6.9	การเปรียบเทียบ String	45
6.10	การจัดวางรูปแบบของ String (String Formatting)	46
6.11	แบบฝึกหัด	47
7	ลิสต์ (Lists)	49
7.1	ความหมายของลิสต์	49
7.2	การเข้าถึงค่าในลิสต์	50
7.3	การแบ่งข้อมูลในลิสต์ (List Slicing)	50
7.4	Lists เปลี่ยนแปลงค่าได้	51
7.5	การใช้ <code>in</code> กับลิสต์	51
7.6	การเดินทางไปในลิสต์ (List Traversal)	52
7.7	ตัวดำเนินการของลิสต์ (List Operators)	52
7.8	เมธอดของลิสต์ (List Methods)	53
7.9	Map, reduce, and filter	54
7.10	List กับ String	55

7.11	Objects กับ values	56
7.12	แบบฝึกหัด	57
8	ดิกชันนารี (Dictionary)	59
8.1	ความหมายของดิกชันนารี	59
8.2	การอ่านค่าในดิกชันนารี	59
8.3	การหาค่าของคีย์ (Key) ใน Dictionary	60
8.4	Dictionary และ List	60
8.5	ฟังก์ชันที่รับ Parameters ได้ไม่จำกัดจำนวน	60
8.6	แบบฝึกหัด	61
9	ทูเปิล (Tuple)	63
9.1	ความหมายของ Tuple	63
9.2	การสลับค่าของ Tuple	64
9.3	การเก็บค่าการดำเนินการใน Tuple	64
9.4	ฟังก์ชัน <code>list()</code> เปลี่ยน tuple ให้เป็น list	65
9.5	Dictionary และ Tuple	65
9.6	แบบฝึกหัด	66
10	การจัดการไฟล์ (Files)	67
10.1	ความหมายของไฟล์	67
10.2	การทำงานกับ Directories	67
10.3	การเปิดไฟล์	68
10.4	การอ่านไฟล์	69
10.5	การจัดการข้อผิดพลาด (Error)	69
10.6	ฐานข้อมูลแบบ Key-Value	70

10.7	การให้ Python เรียกใช้โปรแกรมอื่น	72
10.8	แบบฝึกหัด	73
11	Object-Oriented Programming (OOP)	75
11.1	ความหมายของ OOP (Object-Oriented Programming)	75
11.2	คลาส (Classes) และ ออบเจกต์ (Objects)	75
11.3	การสร้างคลาส	75
11.4	การสร้างออบเจกต์	76
11.5	ฟังก์ชัน <code>__init__()</code>	76
11.6	การสร้างเมธอดของออบเจกต์	76
11.7	การแก้ไขค่าของแอตทริบิวต์ของออบเจกต์	77
11.8	การลบแอตทริบิวต์ของออบเจกต์	77
11.9	การลบออบเจกต์	78
11.10	การสืบทอดคลาส (Class Inheritance)	78
11.11	แบบฝึกหัด	79
	บรรณานุกรม	80

สารบัญรูป

1.1	TIOBE Index for Python ในปี พ.ศ. 2562	2
1.2	สัญลักษณ์ผังงาน (Flowchart)	4
1.3	ตัวอย่างการเขียนผังงาน	5
1.4	การ Download Python 3.7	6
1.5	เลือก Add Python 3.7 to PATH	6
1.6	ไฟล์เตอร์ Python 3.7	7
1.7	ตัวอย่างหน้าโปรแกรม Idle หน้า Editor	7
1.8	ตัวอย่างหน้าโปรแกรม Idle หน้า Python Shell	8
2.1	ลำดับในการคำนวณ	13

สารบัญตาราง

2.1	คำสั่งวนในภาษา Python	11
2.2	สัญลักษณ์การคำนวณทางคณิตศาสตร์	14
2.3	สัญลักษณ์การคำนวณทางคณิตศาสตร์แบบย่อ	14
3.1	ตัวดำเนินการเปรียบเทียบในภาษา Python	21
3.2	ตารางผลการใช้ and	23
3.3	ตารางผลการใช้ or	23
3.4	ตารางผลการใช้ not	23

บทที่ 1

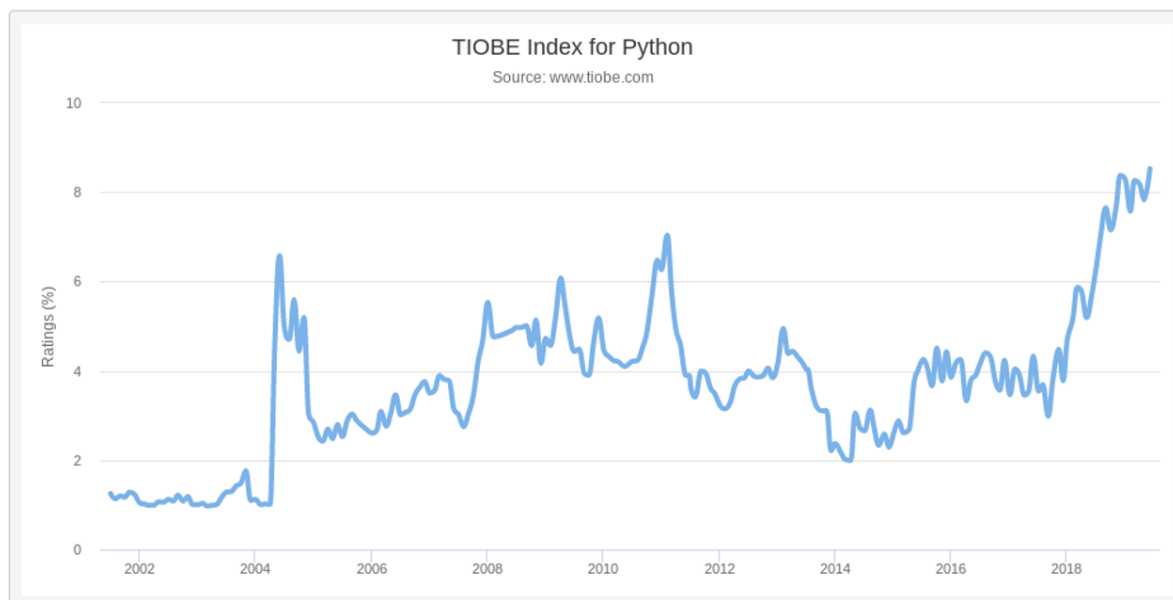
ความรู้เบื้องต้นเกี่ยวกับ Python

1.1 Python คืออะไร

ในปี ค.ศ. 1980 Mr. Guido van Rossum ได้พัฒนาภาษาโปรแกรมมิ่งขึ้นมาและให้ชื่อว่าภาษา Python และเผยแพร่ให้ใช้งานสู่สาธารณะชนในปี ค.ศ. 1991 (Guido, 2019) Python เป็นภาษาโปรแกรมคอมพิวเตอร์ระดับสูง ซึ่งไวยากรณ์ของภาษาระดับสูงนี้จะใกล้เคียงคำในภาษาอังกฤษทั่วไป (Downey, 2015) Python ถูกใช้ในการสร้างโมบายแอปพลิเคชัน เว็บไซต์ เว็บแอปพลิเคชัน ออนไลน์เซอร์วิส รวมทั้งใช้ในการวิเคราะห์ข้อมูล และคำนวณ ทางคณิตศาสตร์และวิทยาศาสตร์อย่างแพร่หลาย ตัวอย่างออนไลน์เซอร์วิสที่พัฒนาขึ้นด้วยภาษา Python ได้แก่ Instagram, Uber, Pinterest, Reddit, Spotify และ Dropbox (Shuup, 2019) โดยในระยะหลายปีที่ผ่านมา Python ได้รับความนิยมสูงขึ้นเรื่อยๆ โดยในเดือนมิถุนายน 2562 ดัชนีความนิยมภาษาโปรแกรมมิ่ง TIOBE ได้แสดงให้เห็นว่า Python เป็นภาษาโปรแกรมมิ่งที่ได้รับความนิยมเป็นอันดับที่ 3 เทียบกับภาษาโปรแกรมมิ่งอื่นๆ และมีความนิยมสูงสุดในรอบ 19 ปี (TIOBE, 2019)

หากเปรียบเทียบกับภาษาโปรแกรมมิ่งอื่นๆ แล้ว Python มีไวยากรณ์ภาษา (Syntax) ที่สามารถอ่านง่าย เข้าใจได้ง่าย และเรียนรู้ง่าย Python จึงเป็นภาษาที่เหมาะสมสำหรับการสอนการเขียนโปรแกรมโดยเฉพาะอย่างยิ่งในระดับเบื้องต้น อีกทั้งยังเป็นภาษาที่ยืดหยุ่นสามารถพัฒนาได้บนระบบปฏิบัติการที่หลากหลาย อาทิ Windows, Linux, OS/2, MacOS, iOS และ Android นอกจากนี้โปรแกรมเมอร์ทั่วโลกได้พัฒนาไลบรารี (Libraries) ขึ้นมาจำนวนมากสำหรับต่อยอด การทำงานของภาษา Python พื้นฐาน เช่น Django, Numpy, Pandas, Matplotlib, Flask, Web2py เป็นต้น (Foundation, 2019)

รูปที่ 1.1: TIOBE Index for Python ในปี พ.ศ. 2562



1.2 Python ทำงานอย่างไร

ภาษาโปรแกรมมิ่งระดับสูงจะต้องถูกโปรแกรมแปลภาษา เช่น คอมไพเลอร์ (Compiler) หรือ อินเทอร์พรีเตอร์ (Interpreter) ทำการแปลภาษาระดับสูงให้กลายเป็นภาษาเครื่องที่คอมพิวเตอร์เข้าใจก่อน (Lutz, 2013) ภาษาตระกูลที่ต้องใช้ Compiler เพื่อแปลงเป็นภาษาคอมไพเลอร์ซึ่งเป็นภาษาที่มนุษย์อ่านไม่ออกแล้วจึงจะทำงานได้ เช่น ภาษา Java ภาษา C หรือภาษา C++ ภาษาพวกนี้จะได้โปรแกรมที่ทำงานรวดเร็วมาก แต่ก็ยากที่จะเรียนรู้ในช่วงการฝึกฝนการเขียน Programming ใหม่ๆ (Barry, 2016)

แต่สำหรับภาษา Python เมื่อได้ Source code ที่เป็นนามสกุลไฟล์ **.py** แล้ว โปรแกรมจะถูกคอมไพล์โดยคอมไพเลอร์ของ Python เพื่อแปลคำสั่ง Python ให้เป็นคำสั่งแบบ Bytecode และบันทึกไว้ในไฟล์นามสกุล **.pyc** ต่อมาเมื่อผู้ใช้ต้องการ Run ไฟล์นี้ อินเทอร์พรีเตอร์ (Interpreter) ก็จะแปลง Bytecode เป็นภาษาเครื่องสำหรับการดำเนินการโดยตรงบนฮาร์ดแวร์ (Beazley & Jones, 2013) อาจเรียกได้ว่า Python เป็นภาษาลูกครึ่งและเรียนรู้ได้ง่าย เหตุผลที่ Python ทำการคอมไพล์เป็น Bytecode เป็นรหัสกลางไว้ก่อนนั้น นั่นก็เพราะ Python ถูกออกแบบมาให้เป็นภาษาการเขียนโปรแกรมที่ไม่ขึ้นกับแพลตฟอร์ม ซึ่งหมายความว่า

ว่ามีการเขียนโปรแกรมหนึ่งครั้ง แต่สามารถเรียกใช้งานบนอุปกรณ์ใดก็ได้ แต่จะต้องติดตั้ง Python เวอร์ชันที่เหมาะสม

1.3 อัลกอริทึม (Algorithm) และ ผังงาน (Flowchart)

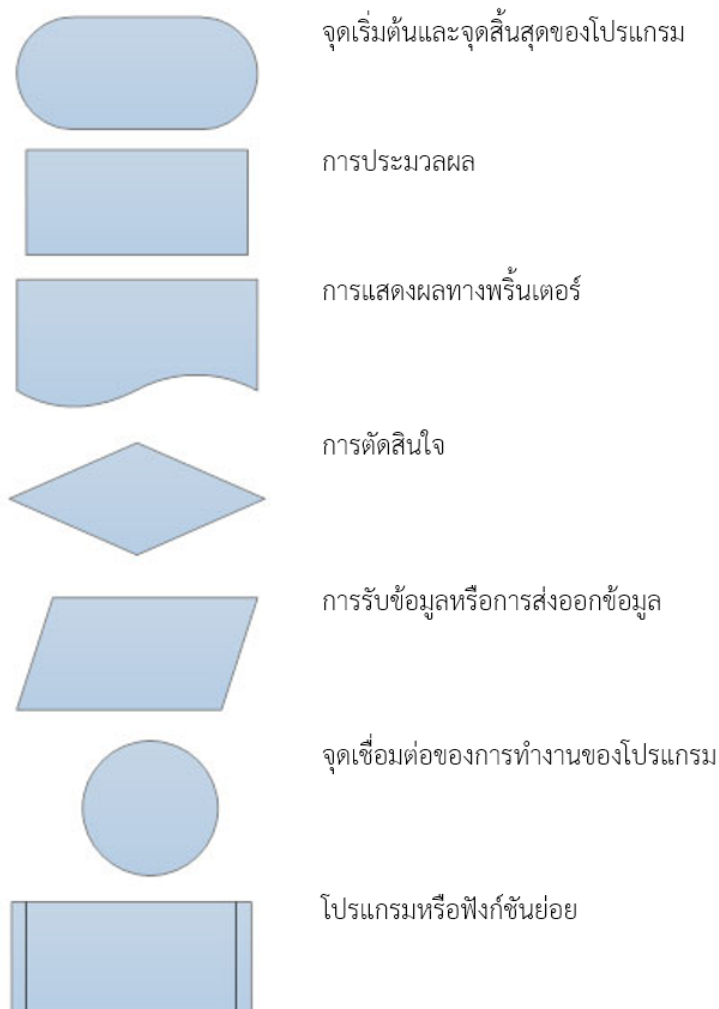
อัลกอริทึม (Algorithm) หมายถึง กระบวนการที่ละขั้นตอนเพื่อแก้ไขปัญหาที่กำหนดอย่างชัดเจน โดยทั่วไปจะมีสามขั้นตอนหลัก คือ มีการนำเข้าข้อมูลหรืออินพุต แล้วนำมาประมวลผล และแสดงผลลัพธ์ออกมา (Bouras, 2019) ตัวอย่างเช่น โจทย์ให้หาค่าเฉลี่ยของตัวเลขที่รับมาจากผู้ใช้งานสามค่า จะสามารถเขียนเป็นขั้นตอนได้ดังนี้คือ

1. ขั้นตอนที่หนึ่ง การนำเข้าข้อมูล
 - แจ้งให้ผู้ป้อนหมายเลขที่หนึ่ง
 - แจ้งให้ผู้ป้อนหมายเลขที่สอง
 - แจ้งให้ผู้ป้อนหมายเลขที่สาม
2. ขั้นตอนที่สอง การประมวลผลข้อมูล
 - คำนวณผลรวมของเลขทั้งสามจำนวน
 - ทหารผลรวมด้วยสาม
3. ขั้นตอนที่สาม การแสดงผลลัพธ์
 - แสดงผลลัพธ์ออกทางหน้าจอ

ส่วนผังงาน (Flowchart) เป็นการนำเสนอการไหลของอัลกอริทึมในรูปแบบของสัญลักษณ์จากคำสั่งหนึ่งไปยังอีกต่อไปจนถึงจุดสิ้นสุดของอัลกอริทึม (Cormen, Leiserson, Rivest, & Stein, 2009) สัญลักษณ์ที่ใช้บ่อยสำหรับผังงานมีดังต่อไปนี้

ตัวอย่างการเขียนผังงานจากอัลกอริทึมด้านบนสามารถเขียนได้ดังนี้

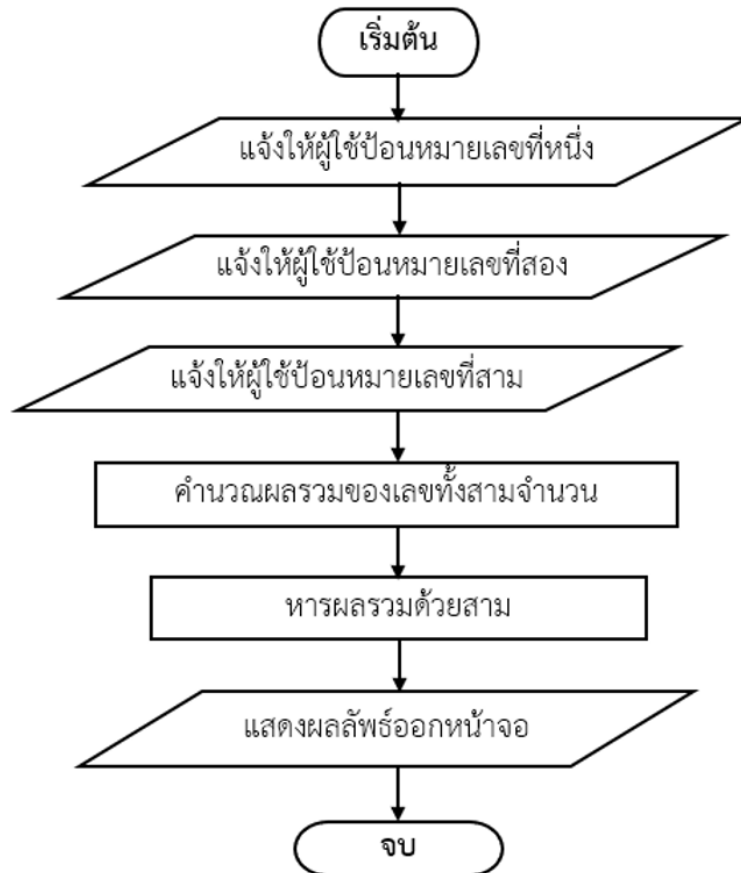
รูปที่ 1.2: สัญลักษณ์ผังงาน (Flowchart)



1.4 การติดตั้งโปรแกรม Python Runtime

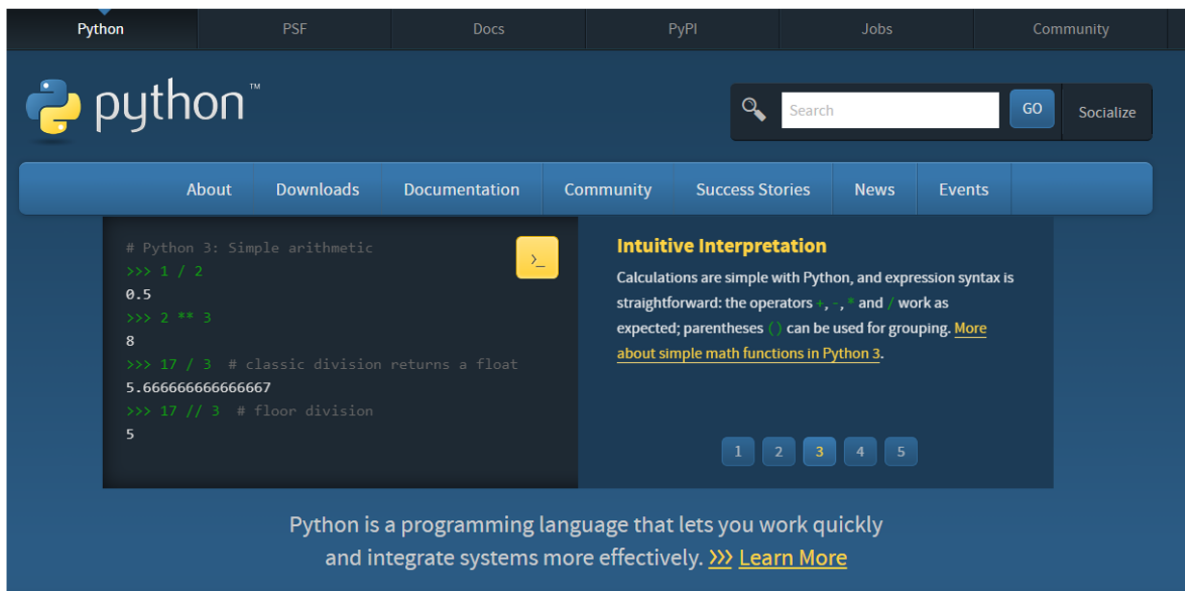
การติดตั้งโปรแกรม Python Runtime คือการติดตั้งโปรแกรมที่ทำให้นักพัฒนาซอฟต์แวร์สามารถใช้งานโปรแกรมที่เขียนขึ้นด้วยภาษา Python เองได้ ให้เข้าไปที่เว็บไซต์ <https://www.python.org/> (Foundation, 2019) ไปที่ Download for Windows แล้วเลือก Python 3.7.0 แล้วทำการติดตั้งให้เรียบร้อยลงในเครื่อง และให้เลือก Add Python 3.7 To Path เพื่อที่จะสามารถใช้ Python ได้ที่ Command Line หลังจากนั้นจะเห็นได้ว่าที่สตาร์ทเมนูโปรแกรม Python 3.7 จะถูกสร้างขึ้น ในโฟลเดอร์นี้จะมีโปรแกรมชื่อว่า Idle

รูปที่ 1.3: ตัวอย่างการเขียนผังงาน

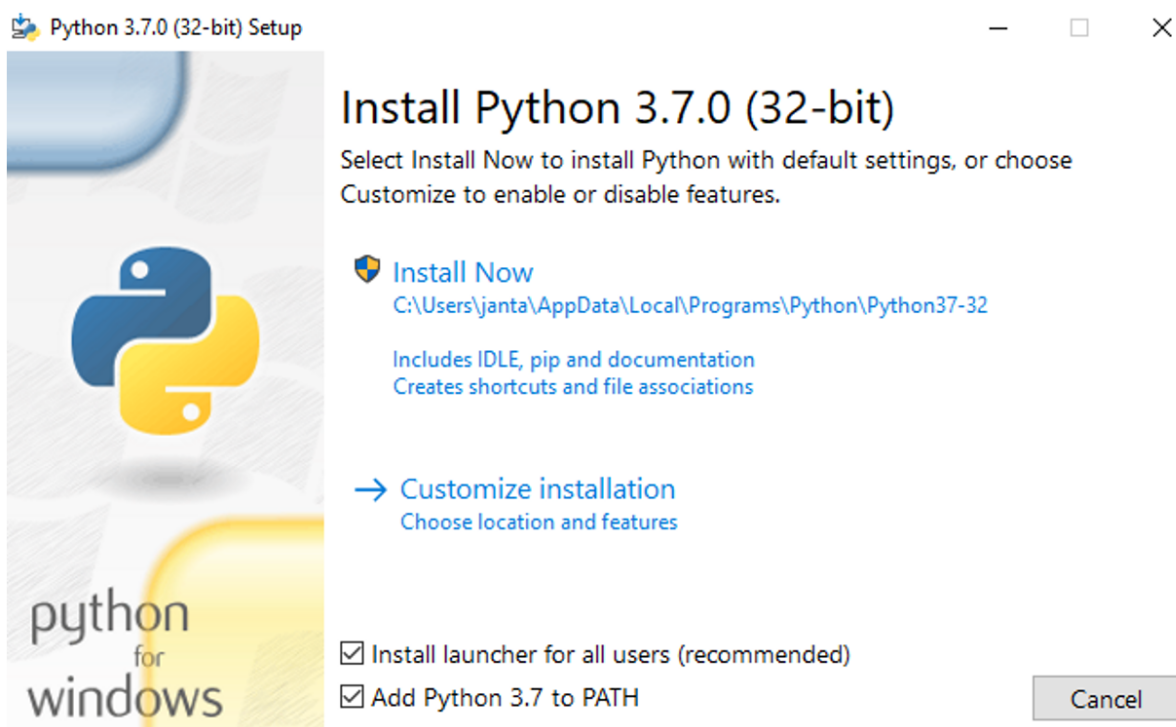


ซึ่งเป็น Integrated Development Environment หรือ เครื่องมือที่ช่วยในการพัฒนาโปรแกรมที่ใช้งานง่าย ๆ เหมาะแก่การเขียนโปรแกรมเบื้องต้น โดยจะมีทั้ง Text Editor และ Interactive Shell เวลาใช้งานควรเปิดไว้ 2 หน้าต่าง ด้านซ้ายมือเป็น Source code ด้านขวามือเป็น Python Shell เพื่อใช้ดูผลลัพธ์ในการ Run โปรแกรม ที่เขียนขึ้น

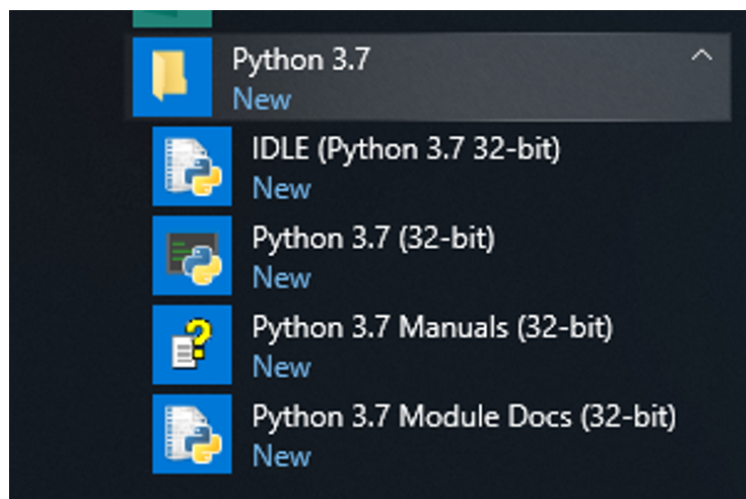
รูปที่ 1.4: การ Download Python 3.7



รูปที่ 1.5: เลือก Add Python 3.7 to PATH



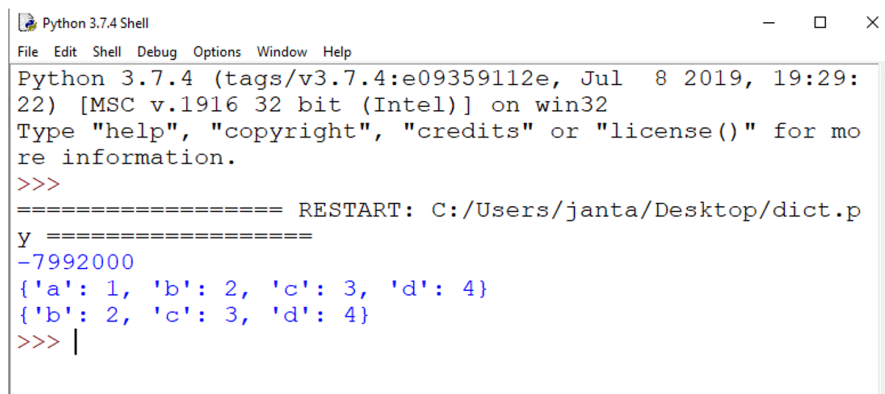
รูปที่ 1.6: โฟลเดอร์ Python 3.7



รูปที่ 1.7: ตัวอย่างหน้าโปรแกรม Idle หน้า Editor

```
*dict.py - C:\Users\janta\Desktop\temp4\Python dict\dict.py (3.7.4)*
File Edit Format Run Options Window Help
for x in range(1,n+1): d[x]=x*x
print(d)
#หาผลรวมของ values
pets={'cats':100,'dogs':60,'pigs':300}
print(sum(pets.values()))
#ตรวจสอบว่ามี key นี้อยู่ใน dict หรือไม่
d = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
def is_key_present(x):
    if x in d:
        print('Key is present in the dictionary')
    else:
        print('Key is not present in the dictionary')
is_key_present(5)
is_key_present(9)
```

รูปที่ 1.8: ตัวอย่างหน้าต่างโปรแกรม Idle หน้า Python Shell



```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/janta/Desktop/dict.py =====
>>>
-7992000
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
{'b': 2, 'c': 3, 'd': 4}
>>> |
```

บทที่ 2

ส่วนประกอบต่างๆ ของภาษา Python

2.1 ตัวแปร (Variables)

ตัวแปร (Variables) คือชื่อที่ผู้เขียนโปรแกรมกำหนดขึ้นมาเอง เพื่อใช้สำหรับการเก็บค่าข้อมูลในการเขียนโปรแกรมไว้ในหน่วยความจำของเครื่องคอมพิวเตอร์ โดยในภาษา Python ไม่ต้องระบุประเภทของตัวแปรไว้ในตอนที่ประกาศการตั้งชื่อตัวแปร ดังตัวอย่างต่อไปนี้

```
1 >>> a = 1
2 >>> a
3 1
4 >>> b = 2
5 >>> b
6 2
7 >>> a + b
8 3
9 >>> vat = 7
10 >>> vat
11 7
```

2.2 การตั้งชื่อตัวแปร

การตั้งชื่อตัวแปรสำหรับภาษา Python มีเงื่อนไขดังนี้

1. ให้ขึ้นต้นด้วยอักษรตัวภาษาอังกฤษตัวใหญ่หรือตัวเล็กตั้งแต่ Aa ถึง Zz เท่านั้น
2. ประกอบด้วยตัวอักษรหรือตัวเลข 0 ถึงเลข 9 หรือตัวขีดกลาง Underscore (_) แต่ห้ามมีช่องว่าง
3. ตัวเลข 0-9 จะนำหน้าชื่อตัวแปรไม่ได้
4. ตัวพิมพ์เล็กและตัวพิมพ์ใหญ่เป็นตัวแปรคนละตัวกัน (Case-Sensitive) เช่น Name ไม่ใช่ตัวแปรเดียวกันกับ name

5. ใช้ใส่เครื่องหมาย = ในการตั้งตัวแปรหรือให้ค่าแก่ตัวแปร
6. การตั้งชื่อตัวแปรควรตั้งอย่างสมเหตุสมผล
7. ภาษา Python จะมีคำที่ถูกสงวนไว้ในการเขียนโปรแกรม หรือ Keywords ซึ่งห้ามนำมาใช้ในการตั้งชื่อตัวแปร ชื่อฟังก์ชัน หรือ ชื่อคลาส

2.3 การตั้งชื่อตัวแปรพร้อมกันหลายตัวแปร

ในการเขียนโปรแกรมภาษา Python ผู้เขียนโปรแกรมสามารถตั้งชื่อตัวแปรพร้อมกันได้หลายตัวแปร โดยพิมพ์ตัวแปรแต่ละตัวในบรรทัดเดียวกันและคั่นแต่ละตัวแปรด้วยเครื่องหมายคอมม่า (,) ตามด้วยเครื่องหมาย (=) และกำหนดค่าตามลงไปตามลำดับการวางตัวแปร ดังตัวอย่างต่อไปนี้

```
1 >>> a, b, c = 1, 'Jan', 2.36
2 >>> a
3 1
4 >>> b
5 'Jan'
6 >>> c
7 2.36
8 >>>
```

2.4 คำสงวน (Keywords)

คำสงวน (Keywords) ในภาษา Python จะมีการสงวนคำบางคำไว้เฉพาะเพื่อใช้เป็นคำสั่งของภาษา โดยผู้เขียนโปรแกรมไม่ควรนำมาใช้ในการตั้งชื่อตัวแปร โดยคำสงวนของภาษา Python มีดังต่อไปนี้ (Lutz, 2014)

ตารางที่ 2.1: คำสงวนในภาษา Python

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

2.5 เลขประจำตัวตำแหน่งของตัวแปร

ตัวแปรจะชี้ไปที่หน่วยความจำในเครื่องคอมพิวเตอร์ซึ่งเก็บค่าของตัวแปรหรือ Value นั้นๆ อยู่ ฉะนั้นเมื่อเราพิมพ์ `a` ดังในตัวอย่าง คอมพิวเตอร์จึงแสดงเลข 1 ออกมา นอกจากนี้พื้นที่ที่เก็บค่านั้นนั้นจะมีที่อยู่อยู่บนหน่วยความจำมีหมายเลขประจำตำแหน่งอีกด้วย โดยใช้คำสั่ง `id()` เพื่อแสดงเลขประจำตำแหน่ง

```

1 >>> a
2 1
3 >>> id(a)
4 1538021648

```

2.6 ชนิดของข้อมูล (Types)

สิ่งที่อยู่ในหน่วยความจำมีชนิดของข้อมูลหรือ Type อยู่ด้วย โดยใช้คำสั่ง `{type() }` เพื่อดูประเภทของข้อมูล ในภาษา Python มีประเภทของข้อมูลหลายๆ แบบ (Ramalho, 2015) ทั้งแบบที่เป็นตัวเลขแบบจำนวนเต็ม ตัวเลขแบบมีจุดทศนิยม ตัวเลขที่มีค่าเป็นบวกหรือลบ ตัวอักษร ข้อความ และตรรกศาสตร์

1. none คือ Nothing ไม่มีอะไร

2. int หรือ Integer คือตัวเลข เช่น 50 หรือ 630 เป็นต้น
3. bool หรือ Boolean คือค่าถูกผิด เช่น True หรือ False เป็นต้น
4. float หรือ floating Point คือจำนวนทศนิยม เช่น 5.6 หรือ 4.23 เป็นต้น
5. str หรือ String หรือข้อความ ซึ่งจะอยู่ภายใต้เครื่องหมายฟันทู (" ") หรือ ฟันทอง (' ') เช่น `"This is my dog."` หรือ `'Jantawan'`

ตัวอย่างการแสดงผลประเภทของข้อมูลมีดังต่อไปนี้

```
1 >>> a = 1
2 >>> a
3 1
4 >>> type(a)
5 <class 'int'>
6 >>> firstname = 'Jantawan'
7 >>> firstname
8 'Jantawan'
9 >>> lastname = 'Piyawat'
10 >>> lastname
11 'Piyawat'
12 >>> id(firstname)
13 67626832
14 >>> type(firstname)
15 <class 'str'>
```

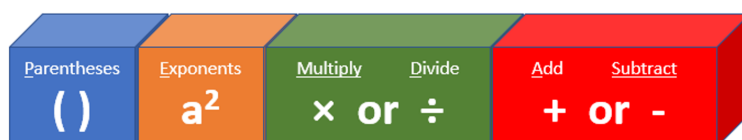
```
1 >>> n = None
2 >>> n
3 >>> id(n)
4 263420692
5 >>> type(n)
6 <class 'NoneType'>
7 >>> yes = True
8 >>> no = False
9 >>> type(yes)
10 <class 'bool'>
11 >>> degree = 1.1
12 >>> id(degree)
13 72213072
14 >>> type(degree)
15 <class 'float'>
```

2.7 เครื่องหมายสำหรับการคำนวณทางคณิตศาสตร์

2.7.1 การคำนวณทางคณิตศาสตร์ (Arithmetic Operators)

เครื่องหมายสำหรับการคำนวณเรียกว่า Arithmetic Operators เช่น เครื่องหมายบวก ลบ คูณ หาร การยกกำลัง การหารเอาเศษ การหารเอาจำนวนเต็ม เป็นต้น การคำนวณทางคณิตศาสตร์แบบซับซ้อนจะต้องมีลำดับในการคำนวณซึ่งเหมือนกับการคำนวณคณิตศาสตร์ทั่วไป คือ ในการแก้สมการทางคณิตศาสตร์จะต้องทำในวงเล็บก่อน ตามด้วยเลขยกกำลัง แล้วจึงตามด้วย คูณหรือหารโดยคำนวณจากซ้ายไปขวา แล้วตามด้วยบวกหรือลบโดยคำนวณจากซ้ายไปขวาเช่นกัน โดยให้จำคำว่า **PEMDAS** ซึ่งเป็นตัวอักษรภาษาอังกฤษตัวแรกของคำว่า Parentheses (วงเล็บ), Exponents (ยกกำลัง), Multiply (คูณ), Divide (หาร), Add (บวก), และ Subtract (ลบ) โดยภาษา Python ใช้สัญลักษณ์คำนวณทางคณิตศาสตร์ดังตารางต่อไปนี้ (Lubanovic, 2015)

รูปที่ 2.1: ลำดับในการคำนวณ



ตัวอย่างคำนวณทางคณิตศาสตร์ในภาษา Python มีดังต่อไปนี้

```

1 >>> a=1
2 >>> b=2
3 >>> a+b
4 3
5 >>> a-b
6 -1
7 >>> c = b-a
8 >>> c
9 1

```

ตารางที่ 2.2: สัญลักษณ์การคำนวณทางคณิตศาสตร์

สัญลักษณ์คำนวณ	ชื่อการคำนวณ	ตัวอย่าง
+	บวก	$a + b$
-	ลบ	$a - b$
*	คูณ	$a * b$
/	หาร	a / b
//	หารปัดเศษทิ้ง	$a // b$
%	เศษของการหาร	$a \% b$
**	ยกกำลัง	$a ** b$

2.7.2 รูปแบบการเขียนการคำนวณทางคณิตศาสตร์แบบย่อ

ในภาษา Python ผู้เขียนโปรแกรมสามารถเขียนการคำนวณทางคณิตศาสตร์แบบลดรูปหรือแบบย่อได้ ดังตารางสัญลักษณ์การคำนวณทางคณิตศาสตร์แบบย่อต่อไปนี้

ตารางที่ 2.3: สัญลักษณ์การคำนวณทางคณิตศาสตร์แบบย่อ

การคำนวณ	ตัวอย่าง	เทียบเท่ากับ
+=	$c += a$	$c = c + a$
-=	$c -= a$	$c = c - a$
*=	$c *= a$	$c = c * a$
/=	$c /= a$	$c = c / a$
//=	$c //= a$	$c = c // a$
%=	$c \% = a$	$c = c \% a$
**=	$c ** = a$	$c = c ** a$

2.7.3 การจัดการข้อความด้วยเครื่องหมายทางคณิตศาสตร์

เครื่องหมายที่ใช้ในการคำนวณทางคณิตศาสตร์เมื่อถูกนำมาใช้กับข้อความ (String) จะเป็นอีกความหมายหนึ่ง เช่น การใช้เครื่องหมายบวกเชื่อมต่อระหว่างสตริง 2 ตัว หรือ การใช้เครื่องหมายดอกจันเป็นการเพิ่มสตริงเดียวกันตามจำนวนครั้งของการคูณ

```
1 >>> firstname
2 'Jantawan'
3 >>> lastname
4 'Piyawat'
5 >>> firstname + lastname
6 'JantawanPiyawat'
7 >>> firstname + ' ' + lastname
8 'Jantawan Piyawat'
9 >>> firstname * 3
10 'JantawanJantawanJantawan'
```

2.8 Expressions และ Statements

Expression หมายถึงการใช้เครื่องหมายคำนวณและการใช้ตัวแปรและค่าของตัวแปรเพื่อหาผลลัพธ์ออกมา เอา Expression มาประกอบกันจะเรียกว่า Statement ดังนั้น Statement ก็คือคำสั่งเรียงต่อกันนั่นเอง เพื่อใช้ในการสั่งงานคอมพิวเตอร์ด้วยภาษาคอมพิวเตอร์

ตัวอย่าง Expression เป็นดังต่อไปนี้

```
1 >>> 1+2
2 3
```

ตัวอย่าง Statement เป็นดังต่อไปนี้

```
1 >>> c = a + b
2 >>> c
3 3
4 >>> print('hello world.')
5 hello world.
```

2.9 การเขียนข้อความอธิบายโปรแกรมโดยใช้ Comment

Comment คือสิ่งที่เราเขียนใน Source Code ของโปรแกรมแต่คอมพิวเตอร์ไม่ต้องแปลผล เพื่อใช้ในการเขียนข้อความประกอบคำอธิบายในการสื่อสารระหว่างผู้เขียนโปรแกรมด้วยกัน หรือเป็นการเตือนความจำของผู้เขียนโปรแกรมเอง โดย Comment ในภาษา Python นำหน้าด้วยเครื่องหมายชาร์ป (#) แล้วหลังจากนั้นตามด้วยข้อความอะไรก็ได้ ถ้าจะเขียน Comment หลายๆ บรรทัดจะต้องใช้เครื่องหมายฟันหนู (" - ") หรือฝนทอง (' - ') 3 ชุด ข้างในใส่ค่า ผลที่ได้จะมีเครื่องหมาย Backslash n (\n) หมายถึงการขึ้นบรรทัดใหม่

ตัวอย่างการใช้ Comment ในบรรทัดเดียว เป็นดังต่อไปนี้

```
1 >>> print('Hello world!')
2 Hello world!
3 >>> # Hello how are you doing?
```

ตัวอย่างการใช้ Comment ในหลายบรรทัด เป็นดังต่อไปนี้

```
1 >>> x = '''
2 hello
3 1
4 2
5 3
6 '''
7 >>> x
8 '\nhello\n1\n2\n3\n'
9 >>> print(x)
10 hello
11 1
12 2
13 3
```

2.10 Source Code

ที่ผ่านมาเป็นการเขียนโปรแกรมแบบ Interactive คือเขียนบน Python Shell แล้วโปรแกรมจะแสดงผลออกมาได้เลย ซึ่งเรียกว่าการทำงานแบบ Interpreter เป็นการใส่คำสั่งไปที่ Prompt และ Python จะแสดงผล

ของคำสั่งนั้นออกมาเลย แต่ในความเป็นจริงแล้วจะเขียนโปรแกรมหลายๆ บรรทัดแล้วสั่งโปรแกรมทำงานทีเดียวพร้อมกัน เราจะเขียนไว้ในไฟล์นั้นเรียกว่า Source Code โดยที่ Source Code ของภาษา Python นามสกุลจะเป็น **.py** เวลาใช้โปรแกรม Idle ให้กดที่เมนู File เลือก New เขียน Source Code แล้วให้กด Run ถ้าหากจะกดรันโปรแกรมอีกครั้งให้กด F5

ตัวอย่าง Python Source Code เป็นดังต่อไปนี้

```
1 x = 7
2 y = 6
3 if x == y: print('x and y are equal.')
4 else:
5     if x < y: print('x is less than y.')
6     else: print('x is greater than y.')
```

ตัวอย่างผลลัพธ์ที่ได้จากการประมวลผล Source Code เป็นดังต่อไปนี้

```
1 x is greater than y.
```

2.11 คำสั่ง print (ตัวแปรหรือข้อมูล)

`print()` เป็นฟังก์ชันที่ใช้ในการแสดงผลตัวแปรหรือข้อมูลออกทางหน้าจอ ดังตัวอย่างต่อไปนี้

```
1 >>> print('Hello world!')
2 Hello world!
3 >>>
```

2.12 การใช้คำสั่ง input() รับค่าจากแป้นพิมพ์

คำสั่ง `input(Prompt)` เป็นคำสั่งสำหรับรับข้อมูลจากผู้ใช้ในการพิมพ์ผ่านแป้นพิมพ์ ดังตัวอย่างต่อไปนี้

```
1 >>> name=input('What is your name? ')
2 What is your name? Jantawan
3 >>> print('Hello, ', name, end='.')
4 Hello, Jantawan.
```

2.13 แบบฝึกหัด

1. จงหาเลขประจำตำแหน่งของข้อมูลต่อไปนี้

- love = 2
- mom = "Jan"
- wed = True
- fah = 39.2

2. จงหาประเภทของข้อมูลต่อไปนี้

- love = 2
- mom = 'Jan'
- wed = True
- fah = 39.2
- money = '22'

3. จงแสดงผลต่อไปนี้

- ตั้งค่าตัวแปร dog, cat
- แสดงข้อความ I have 3 dogs and 2 cats.

4. จงรับค่าจากผู้ใช้และแสดงผลต่อไปนี้

- ตั้งตัวแปร name
- รับค่าด้วยข้อความว่า กรุณาใส่ชื่อของคุณ
- แสดงข้อความ สวัสดีค่ะคุณ

5. จงคำนวณหาค่าตัวเลขต่อไปนี้

- หาค่าพื้นที่สี่เหลี่ยม กว้าง 5 เมตร ยาว 3 เมตร

- หาค่าพื้นที่สามเหลี่ยม สูง 5 เมตร ฐาน 3 เมตร

6. ให้ $a = 3$, $b = 4$, $c = 5$ จงหาค่าต่อไปนี้

- $a == a*1$
- $a != b$
- $a > b$
- $b < c$
- $a+1 >= c$
- $c <= a+b$

บทที่ 3

ประโยคเงื่อนไขในภาษา Python (Conditional Statements)

3.1 การเปรียบเทียบค่า (Boolean Expressions)

Boolean Expressions คือ การดำเนินการเปรียบเทียบค่าเพื่อให้ได้ผลลัพธ์ออกมาเป็นถูก (**True**) หากเงื่อนไขเป็นจริง หรือผลลัพธ์เป็นผิด (**False**) หากเงื่อนไขเป็นเท็จหรือไม่เป็นจริง การเปรียบเทียบค่าใช้กับคำสั่งตรวจสอบเงื่อนไขการตัดสินใจ **if** และคำสั่งการทำงานวนซ้ำทั้ง **for** และ **while** เช่น ค่าของ x มากกว่าค่าของ y ใช่หรือไม่ ซึ่งผลลัพธ์จะออกมาเป็น True หรือ False โดยตัวดำเนินการเปรียบเทียบในภาษา Python ได้แสดงไว้ในตารางต่อไปนี้

ตารางที่ 3.1: ตัวดำเนินการเปรียบเทียบในภาษา Python

สัญลักษณ์	ความหมาย
$x == y$	x เท่ากับ y
$x != y$	x ไม่เท่ากับ y
$x > y$	x มากกว่า y
$x < y$	x น้อยกว่า y
$x >= y$	x มากกว่าหรือเท่ากับ y
$x <= y$	x น้อยกว่าหรือเท่ากับ y

ส่วนตัวอย่างการใช้สัญลักษณ์เปรียบเทียบค่าเป็นดังนี้

```
1 >>> a = 5
2 >>> b = 6
```

```
3 >>> a == b
4 False
5 >>> 5 > 6
6 False
7 >>> 5 < 6
8 True
9 >>> 5 >= 6
10 False
11 >>> print('5 == 5 : ', 5 == 5)
12 5 == 5 : True
13 >>> print('12 >= 13.47 : ', 12 >= 13.47)
14 12 >= 13.47 : False
15 >>>
```

3.2 ตัวดำเนินการทางตรรกศาสตร์ (Logical Operators)

การเปรียบเทียบค่ามากกว่าหนึ่งครั้งเชื่อมต่อกันสามารถดำเนินการได้โดยใช้ตัวดำเนินการทางตรรกศาสตร์ (Logical Operators) ในภาษา Python นั้นมีตัวดำเนินการทางตรรกศาสตร์ 3 ชนิด ซึ่งได้แก่ และ (**and**) หรือ (**or**) ไม่ (**not**) เช่น `a > 2 or c > b and c > 2` ตัวอย่างการใช้ตัวดำเนินการทางตรรกศาสตร์ในภาษา Python เป็นดังต่อไปนี้

```
1 >>> a = -4
2 >>> b = -2
3 >>> a > 2 or c > b and c > 2
4 False
```

ตัวดำเนินการทางตรรกศาสตร์ **and** ใช้เชื่อมสอง Expression ถ้าทั้งสอง Expression มีค่าเป็น True จะได้ผลลัพธ์เป็น **True** ส่วนตัวดำเนินการทางตรรกศาสตร์ **or** ใช้เชื่อมสอง Expression ถ้าทั้งสอง Expression มีค่าเป็น False เท่านั้น ผลลัพธ์จึงจะเป็น **False** และตัวดำเนินการทางตรรกศาสตร์ **not** จะใช้ในการกลับค่าจาก **True** เป็น **False** และจาก **False** เป็น **True** ซึ่งผลการใช้ตัวดำเนินการทางตรรกศาสตร์ทั้งสามตัว แสดงไว้ในตารางดังต่อไปนี้

ตารางที่ 3.2: ตารางผลการใช้ and

Boolean Expression 1 (BE1)	Boolean Expression 1 (BE2)	BE1 and BE2
False	False	False
False	True	False
True	False	False
True	True	True

ตารางที่ 3.3: ตารางผลการใช้ or

Boolean Expression 1 (BE1)	Boolean Expression 1 (BE2)	BE1 or BE2
False	False	False
False	True	True
True	False	True
True	True	True

ตารางที่ 3.4: ตารางผลการใช้ not

Boolean Expression	Not BE1
False	True
True	False

3.3 การใช้คำสั่ง if เพื่อเลือกเงื่อนไข

เงื่อนไขที่ใช้ในภาษา Python คือ if Statement สิ่งที่มาหลัง if คือ Boolean Expression เรียกว่า Statement ใหญ่ และใน Statement ใหญ่ ก็มี Statement ย่อย การดูว่า Statement ย่อยอยู่ใน if Statement ใดให้ดูที่การย่อหน้าหรือ Indentation ในภาษา Python การย่อหน้าสำคัญมากจะเป็นการบอกว่าอะไรอยู่ภายในอะไร

รูปแบบของการใช้งานคำสั่ง if ในภาษา Python โดยถ้าหากเงื่อนไขเป็นจริง ตัวโปรแกรมจะประมวลผลในคำสั่ง if หลังเครื่องหมาย : ดังต่อไปนี้

```
1 if expression:
2     #statements
```

ตัวอย่างเช่น ให้แสดงข้อความว่า อายุต่ำกว่าเกณฑ์ ถ้าหากค่าอายุที่รับเข้ามาต่ำกว่า 18 ปี การเขียน Source Code และผลลัพธ์ของไจทช์เป็นดังนี้

```
1 age = int(input('Enter your age: `'))
2 if age < 18:
3     print('You are underage.')

1 >>>
2 Enter your age: 15
3 You are underage.
4 >>>
```

3.4 การใช้ if กับ else

โครงสร้างคำสั่ง if...else จะดำเนินในบล็อกคำสั่ง else ถ้าหากเงื่อนไขในคำสั่ง if นั้นเป็นเท็จ โดยมีรูปแบบการเขียนดังนี้

```
1 if expression:
2     # statements
3 else:
4     # statements
```

ตัวอย่างการใช้ `if...else` และผลลัพธ์เป็นดังต่อไปนี้

```
1 x = 15
2 y = 6
3 if x > y: print('x is greater than y.')
4 else: print('x is less than or equal to y.')

1 >>>
2 x is greater than y.
3 >>>
```

3.5 Chained Expressions

การใช้ Chained Expressions คือ การใช้ `elif` ไปเรื่อยๆ และเงื่อนไขสุดท้ายจะต้องใช้ `else` โดยไม่ต้องการระบุ Boolean Expressions ใดๆ อีกแล้วหลังจากที่ใส่ `else` ดังตัวอย่างและผลลัพธ์ดังต่อไปนี้

```
1 x = 6
2 y = 6
3 if x > y: print('x is greater than y.')
4 elif x < y: print('x is less than y.')
5 else: print('x and y are equal.')

1 >>>
2 x and y are equal.
3 >>>
```

3.6 Nested Expressions

if มี Statement อยู่ข้างในได้ และ else ก็มี Statement อยู่ข้างในได้เช่นกัน เรียกว่า Nested Expressions ดังตัวอย่างและผลลัพธ์ดังต่อไปนี้

```
1 x = 7
2 y = 6
3 if x == y: print('x and y are equal.')
4 else:
5     if x < y: print('x is less than y.')
6     else: print('x is greater than y.')

```

```
1 >>>
2 x is greater than y.
3 >>>
```

3.7 แบบฝึกหัด

1. จงเขียน code ต่อไปนี้
 - ให้ถามว่า Are you bored? และให้ตอบว่า y หรือ n
 - ถ้าตอบ y ให้พิมพ์ข้อความว่า Let's go outside.
2. จงเขียน code ต่อไปนี้
 - ตั้งค่าตัวแปร var รับค่าเป็นตัวเลขจำนวนเต็มจากผู้ใช้
 - ถ้า $var > 100$ ให้แสดงผลว่า "The value is over 100."
 - ถ้า เป็นกรณีอื่นๆ ให้แสดงผลว่า "The value is less than or equal 100."
3. จงเขียน code ต่อไปนี้
 - รับค่า a, b เป็นจำนวนเต็ม
 - แสดงผลว่า $a > b$ หรือ $a < b$ หรือ $a = b$
4. จงเขียน code ต่อไปนี้
 - รับค่า score เป็นจุดทศนิยม
 - ถ้าคะแนน 81-100 แสดงผลว่า เกรด A
 - ถ้าคะแนน 61-80 แสดงผลว่า เกรด B
 - ถ้าคะแนน 41-60 แสดงผลว่า เกรด C
 - ถ้าคะแนน 0-40 แสดงผลว่า เกรด F
 - เมื่อแสดงผลดังกล่าวแล้ว ให้แจ้งด้วยว่า "ตัดเกรดแล้ว"

บทที่ 4

การเขียนและใช้งานฟังก์ชัน (Functions)

4.1 การเรียกใช้ฟังก์ชัน

ฟังก์ชัน (Function) คือชุดคำสั่งส่วนหนึ่งของโปรแกรมที่ทำงานเพื่อวัตถุประสงค์เฉพาะอย่าง ฟังก์ชันในภาษา Python มีทั้ง Built-in ฟังก์ชันของภาษา Python เอง และฟังก์ชันที่ผู้เขียนโปรแกรมเขียนขึ้นมา การใช้ฟังก์ชันในภาษา Python จะคล้ายกับฟังก์ชันทางคณิตศาสตร์ คือ กำหนดชื่อฟังก์ชันตามด้วยสิ่งที่อยู่ในวงเล็บ ซึ่งเรียกว่า Arguments ซึ่งอาจจะมีได้มากกว่า 1 และในภาษา Python มีการกำหนดฟังก์ชันมาให้เรียกใช้ได้เลย อยู่บ้างแล้ว เช่น `type(42)` คือ การแสดงค่าประเภทของเลข 42 หรือ `id(42)` คือการแสดงตำแหน่งที่อยู่ของเลข 42 ในหน่วยความจำ ดังตัวอย่างต่อไปนี้

```
1 >>> type(42)
2 <class 'int'>
3 >>> a = 1
4 >>> id(a)
5 1538021648
```

4.2 การเรียกใช้โมดูล (Modules)

โมดูล (Modules) คือ ฟังก์ชันที่รวมกันไว้เป็นหมวดหมู่ และสามารถดึงมาใช้ได้ในโปรแกรมได้ด้วยการ `import` เช่น `import math` และหากเรียกใช้ฟังก์ชัน `dir(math)` จะแสดงฟังก์ชันในโมดูล `math` ออกมา ดังตัวอย่างต่อไปนี้

```
1 >>> import math
2 >>> type(math)
3 <class 'module'>
4 >>> id(math)
5 56337632
```

บทที่ 4. การเขียนและใช้งานฟังก์ชัน (Functions)

การใช้งานโมดูลจะมีการใช้งานแบบ Dot Notation หากเห็นการเขียนโมดูล `math` ในลักษณะนี้ เช่น `math.pi` ตัว `pi` เรียกว่าเป็นตัวแปรที่อยู่ในโมดูล `math` ซึ่งไม่ใช่ฟังก์ชัน แต่ถ้าเขียน `math.pow(2,2)` คือ สองยกกำลังสอง ลักษณะนี้จะเป็นการเรียกใช้ฟังก์ชันที่อยู่ในโมดูล ดังตัวอย่างต่อไปนี้

```
1 >>> math.pi
2 3.141592653589793
3 >>> math.pow(2,2)
4 4.0
```

การใช้ฟังก์ชันไม่จำเป็นต้องใช้ฟังก์ชันแบบฟังก์ชันเดียว ฟังก์ชันใช้ฟังก์ชันซ้อนกันได้ คือ การรวมฟังก์ชันหลายๆ อันซ้อนกัน เรียกต่อๆ กันไปได้ ดังตัวอย่างต่อไปนี้

```
1 >>> math.exp(math.log(3+2))
2 4.999999999999999
```

4.3 การสร้างฟังก์ชันในภาษา Python

การเขียนฟังก์ชันหรือสร้างฟังก์ชันขึ้นมาเองจะกระทำเพื่อต้องการให้โปรแกรมที่เขียนขึ้นมาทำงานเพื่อวัตถุประสงค์เฉพาะอย่าง ต้องใช้ `def` Statement ซึ่งย่อมาจาก Define เมื่อเขียนฟังก์ชันไว้ใน Source Code แล้วทำการ Run ฟังก์ชันนั้นจะถูก Define ไว้ในระบบแล้วถูกเรียกใช้ขึ้นมาได้เลย ด้วยการเรียกชื่อฟังก์ชันนั้นกี่ครั้งต่อกี่ครั้งก็ได้ ดังรูปแบบการเขียนต่อไปนี้ทั้งในแบบที่มีการคืนค่าและไม่มีการคืนค่า

```
1 def function_name(args...):
2     # statements
3
4 def function_name(args...):
5     # statements
6     return value
```

ส่วนวิธีการเรียกใช้ฟังก์ชันทำได้โดย เรียกชื่อฟังก์ชันนั้นตามด้วยวงเล็บ () ซึ่งจะมี Arguments หรือไม่ก็ได้ แต่ฟังก์ชันที่กำหนดไว้ ดังตัวอย่างการเรียกใช้ฟังก์ชันที่สร้างขึ้นมาเองต่อไปนี้

```
1 def happy_birthday_song():
2     print('Happy Birthday.')
3     print('Happy Birthday to you.')
```

```

1 >>> happy_birthday_song()
2 Happy Birthday.
3 Happy Birthday to you.

```

4.4 ขอบเขตของตัวแปร

ในวงเล็บ () เป็นการกำหนด Argument ของฟังก์ชัน ซึ่งจะกลายเป็นพารามิเตอร์ (Parameters) หรือตัวแปรที่ใช้ในฟังก์ชันนั้นๆ เท่านั้นและต้องประมวลผลให้เสร็จสิ้นในฟังก์ชัน หรือเรียกว่า Local Variables ตัวแปร Local นี้ จะไม่สามารถเรียกใช้ที่โปรแกรมหลักได้ และผู้เขียนโปรแกรมก็ไม่สามารถนำตัวแปรนี้ไปใช้ในฟังก์ชันอื่น ๆ ได้ ส่วนตัวแปร Global (Global Variables) จะประกาศไว้ในส่วนหลักของโปรแกรมที่เขียนขึ้น ฟังก์ชันย่อยและตัวโปรแกรมหลักสามารถเรียกใช้ตัวแปร Global ได้

```

1 def newage(age):
2     a = 50 #local variable
3     x = age + a #local variable
4     return x
5
6 age = int(input('Enter your age: ')) #global variable
7 print('In 50 years from now, you will be ', newage(age), end='.')

```

```

1 Enter your age: 18
2 In 50 years from now, you will be 68.
3 >>>

```

ตัวแปรใดก็ตามที่จะใช้เป็น Local ให้ใส่คำว่า **global** ไปด้านหน้าตัวแปรที่ถูกเรียกใช้ในฟังก์ชัน อันที่จริงแล้วจะไม่เขียนคำว่า **global** ก็ได้หากชื่อตัวแปรไม่ซ้ำกันเลย

```

1 x = 5
2 def happy_birthday_song(name):
3     global x
4     print('Happy Birthday.')
5     print('Happy Birthday.')
6     print('Happy Birthday.')
7     print('Happy Birthday to ' , name)
8     print(x)
9 happy_birthday_song('Mike')

```

```
1 >>>
2 Happy Birthday.
3 Happy Birthday.
4 Happy Birthday.
5 Happy Birthday to Mike
6 5
7 >>>
```

4.5 ฟังก์ชัน return

ฟังก์ชันทุกอันจะต้อง return คือสิ้นสุดการทำงาน แต่ได้เว้นไว้ในฐานที่เข้าใจ เมื่อโปรแกรมใส่การทำงานมาถึงจุด return โปรแกรมจะหยุดทำงานทันทีที่ฟังก์ชันยังมีคำสั่งอื่นตามมาหลังจาก return อีกก็ตาม ฟังก์ชัน **return** ซึ่งไม่ได้คืน ค่าอะไรออกมาเรียกว่า Void

```
1 x = 5
2 def happy_birthday_song(name):
3     print('Happy Birthday.')
4     return
5     print('Happy Birthday.')
6     print('Happy Birthday.')
7     print('Happy Birthday to ' , name)
8     print(x)
9 happy_birthday_song('Mike')

1 >>>
2 Happy Birthday.
3 >>>
```

4.6 การคืนค่าจากฟังก์ชัน

ฟังก์ชันสามารถ return ค่าได้ด้วย ดังเช่นตัวอย่างการหาพื้นที่สี่เหลี่ยม โดยกำหนดฟังก์ชันชื่อ `rectangle_area(width, height)` และฟังก์ชันมีพารามิเตอร์สองตัวสำหรับความกว้างและความยาวของสี่เหลี่ยม และฟังก์ชันทำการคืนค่า ผลลัพธ์ที่เป็นพื้นที่กลับไปด้วยคำสั่ง **return**

ตัวอย่างการหาพื้นที่สี่เหลี่ยม เป็นดังต่อไปนี้

```

1 def happy_birthday_song(name):
2     print('Happy Birthday.')
3     print('Happy Birthday.')
4     print('Happy Birthday.')
5     print('Happy Birthday to ' , name)
6     return name
7
8 def rectangle_area(width, height):
9     return width * height #
10
11 x = rectangle_area(4, 3)
12 print('The area of rectangle is', str(x))

```

ผลลัพธ์ของตัวอย่างการหาพื้นที่สี่เหลี่ยม เป็นดังต่อไปนี้

```

1 >>>
2 The area of rectangle is 12
3 >>>

```

อีกตัวอย่างของฟังก์ชันที่สร้างขึ้นมาเองโดยมีการใช้ return

```

1 def newage(age):
2     x = age+50
3     return x #
4
5 age = int(input('Enter your age: '))
6 print('In 50 years from now, you will be ', newage(age), end='.')

```

ผลลัพธ์จากตัวอย่างฟังก์ชันที่สร้างขึ้นมาเองโดยมีการใช้ return เป็นดังต่อไปนี้

```

1 >>>
2 Enter your age: 6
3 In 50 years from now, you will be 56.
4 >>>

```

4.7 การเขียนโปรแกรมเชิงฟังก์ชัน (Functional Programming)

การเขียนโปรแกรมเชิงฟังก์ชัน (Functional Programming) เป็นรูปแบบการเขียนโปรแกรมที่เก่าแก่ที่สุด และเป็นรูปแบบที่กลับมาได้รับความนิยมมากในปัจจุบัน คือสร้างฟังก์ชันแล้วให้ฟังก์ชันทำงานร่วมกันโดยไม่มี

บทที่ 4. การเขียนและใช้งานฟังก์ชัน (Functions)

การใช้ Global Variables เลย ฟังก์ชันหนึ่งทำงานส่งผลลัพธ์แก่อีกฟังก์ชันหนึ่งต่อๆ กันไปเรื่อยๆ ซึ่งภาษา Python สามารถใช้เขียนโปรแกรมแบบนี้ได้ ฟังก์ชันแล้ว return ค่าเป็นผลลัพธ์แก่อีกฟังก์ชันหนึ่งไปเรื่อยๆ

จากรูป $x=f(g(h(x)))$ ทำงานเหมือนกันกับ $x = h(x)$ แล้ว $x = g(x)$ แล้ว $x = f(x)$

```
1 x = f(g(h(x)))
2 x = h(x)
3 x = g(x)
4 x = f(x)
```

4.8 ฟังก์ชันที่เรียกตัวเอง (Recursion)

Recursion คือการเรียกใช้ฟังก์ชันซ้อนฟังก์ชันนั้นๆ เอง หรือ ฟังก์ชันเรียกใช้ตัวมันเอง จากฟังก์ชัน `countdown()` ในตัวอย่าง เมื่อมีการเรียกฟังก์ชัน `countdown(5)` โปรแกรมจะทำงานลดหลั่นลงไปเรื่อยๆ คือตั้งแต่ 5, 4, 3, 2, 1 ตราบเท่าที่ n ยังมากกว่า 0 แต่เมื่อเงื่อนไขเป็นเท็จแล้ว โปรแกรมจะแสดงคำว่า Go! แทน ซึ่งมีลักษณะการเขียนดังนี้

```
1 def countdown(n):
2     if x > 0:
3         print(n)
4         countdown(n-1)
5     else:
6         print('Go.')
```

ซึ่งจะได้ผลลัพธ์ดังนี้

```
1 >>> countdown(5)
2 5
3 4
4 3
5 2
6 1
7 Go.
```

4.9 แบบฝึกหัด

1. ตั้งชื่อฟังก์ชัน `hello` เพื่อแสดงผลว่า สวัสดีคุณ
2. สร้างฟังก์ชันคำนวณพื้นที่ รับค่า ความกว้าง และ ความยาว
3. สร้างฟังก์ชันชื่อ `maximal_2` รับ arguments 2 ค่า และ return ค่าที่มากที่สุดออกมา
4. สร้างฟังก์ชันชื่อ `maximal_3` รับ arguments 3 ค่า และ return ค่าที่มากที่สุดออกมา
5. สร้างฟังก์ชันรับ arguments 3 ค่า และ return ผลคูณออกมา
6. สร้างฟังก์ชันรับค่าตัวเลขในหน่วยเมตรต่อวินาที แล้ว return ผลในหน่วยกิโลเมตรต่อชั่วโมง
7. สร้างฟังก์ชันหาผลต่างของรายรับกับรายจ่าย และส่งผลกลับมา

บทที่ 5

การใช้ประโยคสั่งทำงานวนซ้ำ

5.1 ฟังก์ชัน range()

ฟังก์ชัน `range()` คือ ระบุตั้งแต่เริ่มต้นถึงก่อนระยะสิ้นสุด มักจะใช้ในการควบคุมการทำงานของโปรแกรม เป็นจำนวนรอบ มีวิธีการเขียนดังนี้ `range(start, stop[, step])`: โดย start เป็นค่าเริ่มต้น stop เป็นตำแหน่งค่าระยะสิ้นสุด และ step คือ ระยะห่างการเพิ่มดังตัวอย่างต่อไปนี้

```
1 >>> for i in range(5): print(i, end=' ')
2 0 1 2 3 4
3 >>> for i in range(1,5): print(i, end=' ')
4 1 2 3 4
5 >>> for i in range(1,5,2): print(i, end=' ')
6 1 3
7 >>> for i in range(5,1,-2): print(i, end=' ')
8 5 3
9 >>>
```

5.2 คำสั่ง for

คำสั่ง `for` statement เป็นการทำงานซ้ำๆ ตามจำนวนครั้งที่ระบุไว้อย่างแน่นอน เช่น การใช้ `for` statement ร่วมกับ `range()` โดยมีรูปแบบการเขียนดังนี้

```
1 for var in sequence:
2     # statements
```

ตัวอย่างการใช้ `for` statement เป็นดังนี้

```
1 for x in range(0,5): print(x, end=' ')
```

ซึ่งจะได้ผลลัพธ์ดังนี้

```
1 >>>
2 0 1 2 3 4
3 >>>
```

ตัวอย่างการใช้ **for** statement ใน Python shell จะได้ผลลัพธ์ดังนี้

```
1 >>> name = 'Jan'
2 >>> for char in name: print(char, end=' ')
3 J a n
4 >>> mylist = ['Jan', 200, 4.5]
5 >>> for item in mylist: print(item, end = ' ')
6 Jan 200 4.5
7 >>>
```

ตัวอย่างการใช้ **for** statement ที่มีการผสมของการใช้ **if** ในการกำหนดเงื่อนไขการตัดสินใจ เป็นดังนี้

```
1 message = input('Enter a message: ')
2 vowels = 'AEIOU'
3 counter = 0
4 for char in message:
5     if char.upper() in vowels:
6         counter += 1
7 print('Vowels: ', counter)
```

ซึ่งจะได้ผลลัพธ์ดังนี้

```
1 Enter a message: Jantawan
2 Vowels:  3
3 >>>
```

5.3 คำสั่ง while

while Statement เป็นคำสั่งให้โปรแกรมทำงานวนซ้ำในขณะที่เงื่อนไขของการวนซ้ำนั้นยังคงเป็นจริงอยู่ และเมื่อเงื่อนไขเป็นเท็จจะสิ้นสุดการทำงานวนซ้ำนั้นทันที ดังนั้นจึงต้องมีตัวควบคุมในการเพิ่มค่าไปเรื่อยๆ จนเงื่อนไขเป็นเท็จ

while Statement มีลักษณะการเขียน 3 รูปแบบ คือ แบบ Pre-Test แบบ Post-Test และแบบ Mid-Test

รูปแบบการเขียน **while** statement แบบ Pre-Test เป็นดังนี้คือ

```
1 while expression:
2     # statements
```

และมีตัวอย่างการใช้ **while** statement เป็นดังนี้

```
1 x = 0
2 while x < 10:
3     print(x)
4     x = x + 1
```

ซึ่งได้ผลลัพธ์เช่นนี้

```
1 0
2 1
3 2
4 3
5 4
6 5
7 6
8 7
9 8
10 9
```

รูปแบบการเขียน **while** statement แบบ Post-Test เป็นดังนี้คือ

```
1 while True:
2     # statements
3     if Boolean_Expression: break
```

และมีตัวอย่างการใช้ **while** statement เป็นดังนี้

```
1 i = 10
2 while True:
3     if i \% 8 == 0: print( i, " 8  ")
4     if i \% 4 == 0: print( i, " 4  ")
5     if i \% 2 == 0: print( i, " 2  ")
6     i -= 2
7     if i <= 0: break
```

บทที่ 5. การใช้ประโยคสั่งทำงานวนซ้ำ

ซึ่งได้ผลลัพธ์เช่นนี้

```
1 10 2
2 8 8
3 8 4
4 8 2
5 6 2
6 4 4
7 4 2
8 2 2
9 >>>
```

รูปแบบการเขียน **while** statement แบบ Mid-Test เป็นดังนี้คือ

```
1 while expression:
2     # statements 1
3     if Boolean_Expression: break
4     # statements 2
```

และมีตัวอย่างการใช้ **while** statement เป็นดังนี้

```
1 x = 0
2 while True:
3     print(x)
4     if x == 10: break
5     x = x + 1
```

ซึ่งได้ผลลัพธ์เช่นนี้

```
1 0
2 1
3 2
4 3
5 4
6 5
7 6
8 7
9 8
10 9
11 10
```

5.4 แบบฝึกหัด

1. สร้างฟังก์ชันหาค่าของ $3^1 + 3^2 + 3^3 + 3^4 + 3^5$ โดยใช้ For loop
2. สร้างฟังก์ชันให้ผู้ใช้ป้อนค่า x แล้วนำค่า x มาคำนวณ $x^1 + x^2 + x^3 + x^4 + x^5$
3. สร้างฟังก์ชันให้ผู้ใช้ป้อนค่า n และให้แสดงเลขเริ่มที่ n โดยลดลงทีละหนึ่ง โดยใช้ while loop
4. สร้างฟังก์ชันให้ผู้ใช้ป้อนตัวเลข แล้วหาว่ามีเลขใดที่สามารถหารเลขที่ผู้ใช้ป้อนได้ลงตัว เช่น ผู้ใช้ป้อนตัวเลข 4 จะมี 1 2 4 ที่หารเลข 4 ลงตัว
5. สร้างฟังก์ชันคำนวณปีเกิด คศ. เป็น 12 ราศีปีนักษัตร
6. สร้างฟังก์ชันรับจำนวนเงินมาหนึ่งค่า แล้วแลกเปลี่ยนธนบัตร 100 บาท 50 บาท 20 บาท เหรียญ 10 บาท 5 บาท และ 1 บาท

บทที่ 6

การใช้งาน String

6.1 ความหมายของ String

String คือ ข้อความหรือตัวอักษรที่เรียงต่อกันที่อยู่ในเครื่องหมายคำพูดแบบ Double Quotes (ฟั่นหนู) เช่น “How are you?” หรือ Single Quotes (ฝนทอง) เช่น ‘How are you?’

6.2 ฟังก์ชัน len()

มีฟังก์ชันสำหรับ String อยู่หลายฟังก์ชัน เช่น ฟังก์ชัน `len()` มีวัตถุประสงค์เพื่อหาความยาวของ String นั้น และสำหรับการระบุตำแหน่งของตัวอักษรแต่ละตัวใน String จะใช้สัญลักษณ์ก้ามปู `[]` โดยตัวชี้หรือ Index จะเริ่มต้นจาก 0 เช่น `fruit[0]` ตัวอย่างการใช้ฟังก์ชัน `len()` และการใช้สัญลักษณ์ก้ามปู `[]` มีดังต่อไปนี้

```
1 >>> fruit = 'banana'
2 >>> fruit
3 'banana'
4 >>> len(fruit)
5 6
6 >>> type(fruit)
7 <class 'str'>
8 >>> fruit[0]
9 'b'
10 >>> fruit[1]
11 'a'
12 >>> fruit[2]
13 'n'
```

6.3 การเดินทางตามตัวชี้ของ String

วิธีการเดินทางไปเรื่อยๆ ตามตัวชี้ของ String เป็นดังตัวอย่างต่อไปนี้ โดยในผลลัพธ์ชุดแรกจะเกิดจากการใช้ **while** Statement ส่วนในผลลัพธ์ถัดมาจะใช้ตัวแปร String เป็น Iterator โดยการใช้ **for** Statement ซึ่งผลลัพธ์จะออกมาเหมือนกัน

```
1 fruit = 'banana'
2 i = 0
3 while i < len(fruit):
4     print(fruit[i])
5     i += 1
6 for character in fruit: print(character)
```

ผลลัพธ์เป็นดังนี้

```
1 >>>
2 b
3 a
4 n
5 a
6 n
7 a
8 b
9 a
10 n
11 a
12 n
13 a
14 >>>
```

6.4 การตัดคำใน String

การตัด คำ ใน String ด้วย ตัวชี้ (Index) โดยการ ตัด คำ เป็น ส่วน ย่อยๆ จะมี รูป แบบ การ เขียน เป็น **[start:end]** โดยที่ start เป็นตำแหน่งของ Index เริ่มต้นที่ต้องการ และ end นั้นเป็นตำแหน่งก่อนหน้าตำแหน่งสุดท้ายของตัวอักษรที่ต้องการ ตัวอย่างการทำ String slices เป็นดังต่อไปนี้


```
1 >>> s = 'Monty Python'
2 >>> s
3 'Monty Python'
4 >>> len(s)
5 12
6 >>> s[0:1]
7 'M'
8 >>> s[0]
9 'M'
10 >>> s[0:2]
11 'Mo'
12 >>>
```

หากเขียนเป็น [start:] ระบุจุดเริ่มต้นที่ start ผลลัพธ์จะแสดงยาวไปจนถึงจุดสิ้นสุด และหากเขียนเป็น [:end] ผลลัพธ์ที่ได้จะแสดงอักษรตั้งแต่ตัวแรกหรือตัวชี้ที่ศูนย์ไปจนถึงตัวสิ้นสุดที่ระบุไว้ ตัวอย่างการเขียน String slices แบบไม่ระบุต้นหรือปลาย เป็นดังต่อไปนี้

```
1 >>> s[1:]
2 'onty Python'
3 >>> s[:5]
4 'Monty'
5 >>>
```

6.5 โครงสร้างข้อมูลที่ไม่เปลี่ยนแปลงไม่ได้

String เป็นโครงสร้างข้อมูลที่ไม่เปลี่ยนแปลงไม่ได้ ดังนั้นถ้าหากต้องการสร้าง String ใหม่ก็ต้องสร้างเป็น Object ใหม่เท่านั้น ตัวอย่างการสร้าง String ใหม่เป็นดังต่อไปนี้

```
1 >>> s
2 'Monty Python'
3 >>> s[0] = 'J'
4 Traceback (most recent call last):
5   File "<pyshell#19>", line 1, in <module>
6     s[0] = 'J'
7   TypeError: 'str' object does not support item assignment
8 >>> id(s)
9 62723320
10 >>> a = 'J' + s[1:]
```

```
11 >>> a
12 'Jonty Python'
13 >>> id(a)
14 62737048
```

6.6 การค้นหาตัวอักษรใน String

การเขียนโปรแกรมเพื่อค้นหาตัวอักษรใน String แล้วส่งค่าออกมาเป็นค่าตัวชี้ตัวอักษรใน String มีผังงานดังต่อไปนี้

- ให้ฟังก์ชันชื่อว่า find มีการส่งค่า str เป็น string และค่า char เป็น character
- ตั้งตัวนับ i เริ่มต้นที่ 0
- ขณะที่ i ยังน้อยกว่าจำนวนตัวอักษรใน string ที่ชื่อว่า str ให้ดำเนินการดังนี้คือ
 - ตรวจสอบว่า ถ้าตัวชี้ของตัวอักษรเท่ากับตัวอักษรที่ต้องการแล้ว ให้ส่งค่าตัวนับออกมา
- เมื่อ while เป็นเท็จแล้ว หรือเมื่อค้นจนครบ character ใน string แล้วให้ส่งค่ากลับคือ -1

จากผังงานดังกล่าวสามารถเขียนเป็นฟังก์ชันในภาษา Python ได้ดังนี้

```
1 def find(str, char):
2     i = 0
3     while i < len(str):
4         if str[i] == char: return i
5         i += 1
6     return -1
```

ซึ่งจะได้ผลลัพธ์ดังนี้

```
1 >>> find('Mike', 'c')
2 -1
3 >>> find('Mike', 'e')
4 3
5 >>>
```

6.7 เมธอดของ String (String Methods)

การดำเนินการกับ String สามารถศึกษาฟังก์ชันได้ที่ <https://docs.python.org/2.4/lib/string-methods.html> เช่น `str.upper()` ใช้ทำงานเพื่อแปลงตัวอักษรภาษาอังกฤษเป็นตัวพิมพ์ใหญ่ ตัวอย่างการใช้ String methods เป็นดังต่อไปนี้

```
1 >>> s = 'Monty Python'
2 >>> s
3 'Monty Python'
4 >>> s.count('t')
5 2
6 >>> s.capitalize()
7 'Monty python'
8 >>> s.upper()
9 'MONTY PYTHON'
```

6.8 in โอเปอเรเตอร์

`in` โอเปอเรเตอร์ใช้ในการพิสูจน์ค่าแบบ Boolean Expression เช่น `'t' in s` แปลว่า ตัวอักษร `t` อยู่ใน String ชื่อว่า `s` หรือไม่ หรือในทางตรงข้ามเพื่อตรวจสอบว่าไม่มีหรือไม่ให้ใส่ `not in` ดังตัวอย่างต่อไปนี้

```
1 >>> 't' in s
2 True
```

6.9 การเปรียบเทียบ String

การเปรียบเทียบ String สามารถใช้สัญลักษณ์ (`>` , `<` , `<=` , `>=` , `==` , `!=`) เพื่อเปรียบเทียบค่าของ String สองชุด โดยดูผลลัพธ์ของการเปรียบเทียบค่าของ ASCII value นั้นๆ ดังตัวอย่างต่อไปนี้

```
1 >>> 'banana' == 'banana'
2 True
3 >>> 'banana' != 'banana'
4 False
5 >>> 'banana' > 'banana'
6 False
7 >>> 'banana' < 'banana'
8 False
9 >>> 'banana' > 'Banana'
10 True
11 >>> 'banana' > 'bazooka'
12 False
```

6.10 การจัดวางรูปแบบของ String (String Formatting)

การเปลี่ยนการจัดวางรูปแบบของ String มีสองวิธีคือ แบบ Classic ซึ่งทำได้โดยใช้สัญลักษณ์ + แต่สัญลักษณ์นี้จะมีข้อจำกัดคือไม่สามารถแทรกข้อความระหว่างกันได้ แต่หากใช้สัญลักษณ์ % จะช่วยแก้ไขข้อจำกัดนี้ได้ เช่น %s สำหรับ string และ %d สำหรับตัวเลข ตัวอย่างการเปลี่ยนการจัดวางของ String โดยใช้ %s เป็นดังต่อไปนี้

```
1 >>> 'Hello' + 'Mike'
2 'HelloMike'
3 >>> 'hello %s' % 'Mike'
4 'hello Mike'
5 >>>
```

และตัวอย่างการเปลี่ยนการจัดวางของ String โดยใช้ %d เป็นดังต่อไปนี้

```
1 >>> 'Total is \%d baht' \%12
2 'Total is 12 baht'
3 >>>
```

ส่วนการเปลี่ยนการจัดวางของ String แบบ Modern คือ ใช้ **.format** และระบุ Parameters ได้มากกว่า 1 ตัว อีกทั้งยังสามารถจัดเรียงลำดับ Parameters สลับก่อนหลังได้ตามสะดวก สามารถศึกษาเพิ่มเติมได้ที่ <https://docs.python.org/3/library/string.html#format-examples> ตัวอย่างการจัดวาง String โดยใช้ .format() เป็นดังต่อไปนี้

```
1 >>> x = 12
2 >>> 'Total is {0} baht' .format(x)
3 'Total is 12 baht'
4 >>> 'Total is {1} baht. Mr. {0}' .format('Mike', x)
5 'Total is 12 baht. Mr. Mike'
```

6.11 แบบฝึกหัด

1. เขียนฟังก์ชันต่อไปนี้

- รับค่าข้อความ 'James had had had the cat.'
- นับจำนวนคำว่า had

2. เขียนฟังก์ชันต่อไปนี้

- รับค่าข้อความ 'I intend to live forever, or die trying.'
- แทนค่าคำว่า to ด้วย three

3. เขียนฟังก์ชันต่อไปนี้

- คำนวณความยาวของ String ที่รับมาจากผู้ใช้

4. เขียนฟังก์ชันต่อไปนี้

- รับ String มาแล้วเปลี่ยนค่ากลับกันระหว่างตัวอักษรตัวแรกกับตัวสุดท้ายของ String นั้น

บทที่ 7

ลิสต์ (Lists)

7.1 ความหมายของลิสต์

ลิสต์ (List) เป็นโครงสร้างข้อมูลที่สำคัญมากของภาษา Python คล้ายคลึงกับ Array ในภาษาอื่นๆ คือ ชุดของข้อมูลที่เรียงลำดับต่อกัน กัน จะเป็นค่าของอะไรก็ได้ การสร้าง list โดยกำหนดสัญลักษณ์ [] เช่น `t = []` หรือ `t=list()` ดังตัวอย่างต่อไปนี้

```
1 >>> t = []
2 >>> t
3 []
4 >>> type(t)
5 <class 'list'>
6 >>> id(t)
7 55105976
8 >>> t = list()
```

ตัวอย่างของการกำหนดค่าใน List เช่น

- `t = [1, 2, 3, 4]` เป็น list ของตัวเลข
- `t = [1, "yes", "no", 1.1]` เป็นลิสต์ผสมของตัวเลขและข้อความ
- `t = [1, 2, 3, ['yes', 'no'], []]` เป็นลิสต์ที่มีลิสต์เป็นองค์ประกอบอยู่ด้วย

กล่าวได้ว่าเราสามารถเอาค่าของหลายๆ ประเภทมาอยู่รวมกันในลิสต์เดียวกันได้ ดังตัวอย่างต่อไปนี้

```
1 >>> t = [1,2,3,4,5]
2 >>> len(t)
3 5
4 >>> x = [1,'Mike',1.1,True]
5 >>> x
6 [1,'Mike',1.1,True]
7 >>> len(x)
8 4
9 >>> y = [1,['a','b'],True]
```

```
10 >>> y
11 [1, ['a', 'b'], True]
12 >>> len(y)
13 3
14 >>>
```

7.2 การเข้าถึงค่าในลิสต์

ค่าในลิสต์จะเรียงตามตัวชี้หรือ Index ที่เริ่มต้นที่ 0 เช่น `t=[0]` และหากมีลิสต์ซ้อนอยู่ในลิสต์จะสามารถแสดงตัวชี้ได้ด้วยการกำหนดตัวชี้หลักตามด้วยตัวชี้ย่อย เช่น `t=[1][0]` ดังตัวอย่างต่อไปนี้

```
1 >>> y = [1, ['a', 'b'], True]
2 >>> y
3 [1, ['a', 'b'], True]
4 >>> len(y)
5 3
6 >>> t
7 [1, 2, 3, 4, 5, 6]
8 >>> t[0]
9 1
10 >>> t[1]
11 2
12 >>> t[2]
13 3
14 >>> y[0]
15 1
16 >>> y[1]
17 ['a', 'b']
18 >>> y[1][0]
19 'a'
```

7.3 การแบ่งข้อมูลในลิสต์ (List Slicing)

List Slicing หรือการแบ่งข้อมูลในลิสต์เป็นชุดข้อมูลย่อยๆ จะเขียนในรูปแบบ `[a:b]` เมื่อ `a` เป็น Index เริ่มต้นและ `b` เป็น Index ก่อนสมาชิกตัวสุดท้ายที่ต้องการตัด ดังตัวอย่างต่อไปนี้


```
1 >>> t
2 [1, 2, 3, 4, 5, 6]
3 >>> t[1:3]
4 [2,3]
5 >>> t[2:]
6 [3, 4, 5, 6]
7 >>> t[:4]
8 [1, 2, 3, 4]
9 >>>
```

7.4 Lists เปลี่ยนแปลงค่าได้

ลิสต์สามารถเปลี่ยนแปลงค่าได้ ดังตัวอย่างต่อไปนี้

```
1 >>> t
2 [True, 2, 3, 4, 5, 6]
3 >>> t[1] = 'Hello World'
4 >>> t
5 [True, 'Hello World', 3, 4, 5, 6]
6 >>> t[1]
7 'Hello World'
```

7.5 การใช้ in กับลิสต์

การดำเนินการด้วย **in** สามารถใช้กับลิสต์ได้ ดังตัวอย่างต่อไปนี้

```
1 >>> t
2 [True, 2, 3, 4, 5, 6]
3 >>> t[1] = 'Hello World'
4 >>> t
5 [True, 'Hello World', 3, 4, 5, 6]
6 >>> t[1]
7 'Hello World'
8 >>> 3 in t
9 True
```

7.6 การเดินทางไปในลิสต์ (List Traversal)

การใช้ **for** loop และตัวดำเนินการ **in** ในการเดินทางไปในลิสต์ เป็นดังตัวอย่างต่อไปนี้

```
1 fruits = ['banana', 'orange', 'mango']
2 for fruit in fruits: print(fruit)
```

ซึ่งผลลัพธ์ที่ได้เป็นดังนี้

```
1 >>>
2 banana
3 orange
4 mango
5 >>>
```

การใช้ฟังก์ชัน **range()** กับลิสต์ สำหรับ **range()** ของความยาวของตัวแปร จะถูกนำมาใช้ในการเดินทางด้วย index ในลิสต์ เป็นดังตัวอย่างต่อไปนี้

```
1 for i in range(len(fruits)):
2     print('{0} = {1}'.format(i, fruits[i]))
```

ซึ่งผลลัพธ์ที่ได้เป็นดังนี้

```
1 >>>
2 0 = banana
3 1 = orange
4 2 = mango
5 >>>
```

7.7 ตัวดำเนินการของลิสต์ (List Operators)

ถ้าต้องการเอาลิสต์มารวมกันให้ใช้เครื่องหมายบวก (+) ถ้าต้องการขยายลิสต์ให้มีค่าชุดเดิมเพิ่มเป็นกี่เท่าตัวให้ใช้เครื่องหมายคูณ (*) เป็นดังตัวอย่างต่อไปนี้

```
1 >>> a = [1,2,3]
2 >>> b = [4,5,6]
3 >>> a
4 [1, 2, 3]
5 >>> b
6 [4, 5, 6]
7 >>> a + b
8 [1, 2, 3, 4, 5, 6]
9 >>> a * 2
10 [1, 2, 3, 1, 2, 3]
11 >>>
```

7.8 เมธอดของลิสต์ (List Methods)

การขยายลิสต์ด้วยลิสต์หนึ่ง ให้ใช้คำสั่ง `list.extend()` เป็นดังตัวอย่างต่อไปนี้

```
1 >>> a
2 [1, 2, 3]
3 >>> b
4 [4, 5, 6]
5 >>> a.extend(b)
6 >>> a
7 [1, 2, 3, 4, 5, 6]
```

การเพิ่มค่าในลิสต์ทำได้ด้วยคำสั่ง `list.append()` ค่าใหม่ที่ได้จะต่อท้ายตัวสุดท้ายในลิสต์ เป็นดังตัวอย่างต่อไปนี้

```
1 >>> a.append(7)
2 >>> a
3 [1, 2, 3, 4, 5, 6, 7]
```

การเพิ่มค่าในลิสต์โดยกำหนดตำแหน่งของ index ให้ใช้คำสั่ง `list.insert()` เป็นดังตัวอย่างต่อไปนี้

```
1 >>> a.insert(0, 0)
2 >>> a
3 [0, 1, 2, 3, 4, 5, 6, 7]
```

การลบค่าในลิสต์ทำได้ด้วยคำสั่ง `list.remove()` เป็นดังตัวอย่างต่อไปนี้

```
1 >>> a.remove(7)
2 >>> a
3 [0, 1, 2, 3, 4, 5, 6]
```

นอกจากนี้สามารถลบค่าในลิสต์ได้ด้วยคำสั่ง `del` โดยจะต้องระบุค่า Index ของตัวที่ต้องการลบ เช่น `del a[0]` เป็นการลบค่าลิสต์ที่มี index 0 หรือถ้าลบเป็นช่วงให้ระบุตำแหน่งเริ่มต้นที่จะลบจนถึงตำแหน่งตัวก่อนสุดท้ายที่จะลบ `del a[2:4]` จะลบตัวที่ 2 และ 3 ในลิสต์ และถ้าลบค่าในลิสต์ออกทั้งหมดให้ใช้คำสั่ง `del a[:]`

ศึกษาเรื่อง list methods เพิ่มเติมได้ที่ <https://docs.python.org/3/tutorial/datastructures.html>

7.9 Map, reduce, and filter

การสร้าง Map ฟังก์ชัน เป็นการทำให้ลิสต์หนึ่งเป็นอีกลิสต์หนึ่ง ให้ฟังก์ชันชื่อ `capitalize()` รับลิสต์ `t` มา แล้ว return ลิสต์ `r` แล้วทำการเดินทางในค่าแต่ละค่า เมื่อเจอค่าก็ให้ทำการประมวลผลเป็นตัวพิมพ์ใหญ่แล้วเก็บไว้ในลิสต์ `r` เป็นดังตัวอย่างต่อไปนี้

```
1 def capitalize(t):
2     r = []
3     for s in t:
4         r.append(s.capitalize())
5     return r
```

ซึ่งผลลัพธ์ที่ได้เป็นดังนี้

```
1 >>> capitalize(['a', 'b', 'c', 'd'])
2 ['A', 'B', 'C', 'D']
3 >>>
```

การสร้าง Reduce ฟังก์ชันเป็นการประมวลผลลิสต์เพื่อการผลสรุป ให้ฟังก์ชันชื่อ `sum()` รับลิสต์ `t` มา แล้ว return ค่า `sum` โดยกำหนดตัวแปรชื่อ `sum` ให้ค่าเป็น 0 แล้วเดินทางไปในลิสต์ทีละค่า แล้วนำค่าเมื่อบวกกัน เมื่อบวกจนครบทุกตัวแล้วให้ส่งค่ากลับมาเป็น `sum` เป็นดังตัวอย่างต่อไปนี้

```

1 def sum(t):
2     sum = 0
3     for x in t: sum += x
4     return sum

```

ซึ่งผลลัพธ์ที่ได้เป็นดังนี้

```

1 >>> sum([1,2,3,4,5])
2 15

```

ฟังก์ชันอีกประเภทหนึ่งเรียกว่า filter ฟังก์ชัน คือการ search แบบรับลิสต์มาแล้ว return ค่า คือมีการค้นหาแล้วทำการประมวลผลค่านั้นๆ หรืออาจจะ return มาเป็นลิสต์ก็ได้ แล้วก็ทำการประมวลผลกับข้อมูลที่อยู่ในลิสต์นั้น เช่น ฟังก์ชันรับลิสต์ t เข้าไป แล้ว return ลิสต์ที่เป็น integer เท่านั้น โดยตั้งต้นสร้างลิสต์ r แล้วเดินทางไปในค่าแต่ละค่าของลิสต์ t ถ้าเจอว่าประเภทของค่าเป็น int ให้ทำการแทรกค่าในลิสต์ r เมื่อทำครบแล้วให้ส่งค่าลิสต์ r ออกมา เป็นดังตัวอย่างต่อไปนี้

```

1 def only_int(t):
2     r = []
3     for x in t:
4         if type(x) == int: r.append(x)
5     return r

```

ซึ่งผลลัพธ์ที่ได้เป็นดังนี้

```

1 >>> only_int([1,2,3,True,'hello',4,'abcdef',1.1])
2 [1, 2, 3, 4]
3 >>>

```

7.10 List กับ String

String เปลี่ยนแปลงค่าไม่ได้ แต่ List เปลี่ยนแปลงค่าได้ ทั้ง String และ List เป็นการเรียงลำดับและเปลี่ยนแปลงค่ากลับกันไปมาได้ด้วยการใช้ฟังก์ชันของลิสต์ เช่น `split()`, `join()` ดังตัวอย่างต่อไปนี้

```
1 >>> s = 'Mink is a cat.'
2 >>> s
3 'Mink is a cat.'
4 >>> t = s.split()
5 >>> t
6 ['Mink', 'is', 'a', 'cat.']
7 >>> ' '.join(t)
8 'Mink is a cat.'
9 >>> p = '081-123-4567'
10 >>> p
11 '081-123-4567'
12 >>> t = p.split('-')
13 >>> t
14 ['081', '123', '4567']
15 >>> '-'.join(t)
16 '081-123-4567'
```

7.11 Objects กับ values

ภาษา Python เพื่อประหยัดพื้นที่ในหน่วยความจำ สำหรับ String ซึ่งไม่สามารถเปลี่ยนแปลงค่าได้ หรือ Immutable Data Structure ภาษา Python จะชี้ชื่อตัวแปรไปที่ที่เดียวกันสำหรับค่าที่เหมือนกัน ดังตัวอย่างต่อไปนี้

```
1 >>> a = 'banana'
2 >>> b = 'banana'
3 >>> id(a)
4 67486272
5 >>> id(b)
6 67486272
7 >>> a is b
8 True
```

แต่สำหรับลิสต์ซึ่งเปลี่ยนแปลงค่าได้ หรือ Mutable Data Structure ภาษา Python จะเก็บไว้คนละที่ในหน่วยความจำ แต่สามารถสร้างชื่อตัวแปรต่อๆ กันมาได้ สำหรับตำแหน่งหนึ่งๆ เรียกว่า การทำ Aliasing

Objects อยู่ในหน่วยความจำมี Values แต่ไม่มีชื่อ แต่มี id หรือหมายเลขกำกับ สร้างตัวแปรเพื่อชี้ไปยัง Objects เหล่านั้น ตั้งตัวแปรหลายตัวหรือตัวเดียวก็ได้ ดังตัวอย่างต่อไปนี้

```
1 >>> a = [1,2,3]
2 >>> b = [1,2,3]
3 >>> a is b
4 False
5 >>> id(a)
6 67462368
7 >>> id(b)
8 67464088
9 >>> c = a
10 >>> id(c)
11 67462368
12 >>> c is a
13 True
```

7.12 แบบฝึกหัด

- กำหนดคะแนนของนักเรียน 5 คน เก็บไว้ใน list คือ 75 80 68 82 62 ต้องการหาคะแนนรวม คะแนนเฉลี่ย คะแนนมากที่สุด คะแนนน้อยสุด
- ให้ข้อมูลเป็น list มีค่าคือ 3 4 12 31 ให้หาจำนวนสมาชิกใน list และพิมพ์สมาชิกทุกตัวตัวละบรรทัด
- ให้ข้อมูลเป็น list มีค่าคือ 25 4 3 15 21 นำมาเรียงจากน้อยไปหามาก พร้อมหาผลคูณของสมาชิกทุกตัว
- ให้ข้อมูลเป็น list มีค่าคือ 6 9 8 7 10 ให้ลบข้อมูลตัวแรกทิ้งแล้วเพิ่มข้อมูลตัวแรกไปที่ตำแหน่งสุดท้าย
- จงสร้างข้อมูลเป็น list ชื่อ a มีค่าคือ 1-10 และ list ชื่อ b มีค่าคือ 11-20 โดยใช้ for และใช้ฟังก์ชัน append เพื่อเพิ่มสมาชิกใน list แล้วหาค่า a+b

บทที่ 8

ดิกชันนารี (Dictionary)

8.1 ความหมายของดิกชันนารี

ดิกชันนารี (Dictionary) คือประเภทข้อมูลที่เก็บข้อมูลในเป็นคู่ๆ ของ Key และ Value โดยที่ Key ใช้สำหรับเป็น Index ในการเข้าถึงข้อมูล Value ของ Key นั้นๆ การสร้าง Dictionary เปล่าจะเขียนอยู่ในวงเล็บปีกกา หรือ `dict()` ส่วนการใส่ค่าใน Dictionary จะเขียนอยู่ในวงเล็บปีกกา แต่ละคู่จะขึ้นต้น key ตามด้วย : แล้วตามด้วย value และคั่นคู่ด้วยเครื่องหมาย comma (,) ดังตัวอย่างต่อไปนี้

```
1 >>> stocks = {'scb':160, 'ptt':360}
2 >>> stocks
3 {'scb':160, 'ptt':360}
```

8.2 การอ่านค่าในดิกชันนารี

การอ่านค่าใน Dictionary ด้วยคำสั่ง `for` หรือการใช้ Dictionary เป็น Iterators เมื่อมีการประกาศค่าของ Dictionary แล้ว `for` จะวนอ่านค่าใน Dictionary ทีละตัว เช่น

```
for key in stocks: print(key, stocks[key])
```

ตัวอย่างการใช้ Dictionary เป็น Iterator

```
1 stocks = {'scb':160, 'ptt':360}
2 for key in stocks:
3     print('{0} = {1}'.format(key, stocks[key]))
```

ซึ่งได้ผลลัพธ์ดังนี้

```
1 ptt = 360
2 scb = 160
```

8.3 การหาค่าของคีย์ (Key) ใน Dictionary

การหาค่าของคีย์ในดิกชันนารีเมื่อรู้ value เช่น สร้างฟังก์ชัน `reverse_lookup()` มีการส่ง Parameters สองตัวคือ Dictionary กับ Value สำหรับ Key แต่ละตัวใน Dictionary นี้ ถ้า ค่าของ Dictionary เท่ากับ Value ที่ต้องการ ให้ส่งค่า Key กลับมา แต่ถ้าไม่มีก็ไม่ต้องส่งอะไรออกมา ดังตัวอย่างต่อไปนี้

```
1 def reverse_lookup(d,v):
2     for key in d:
3         if d[key] == v: return key
4     return None
```

ซึ่งได้ผลลัพธ์ดังนี้

```
1 >>> reverse_lookup(stocks, 160)
2 'scb'
```

8.4 Dictionary และ List

`keys()` method จะแสดง keys ออกมา ส่วน `values()` method จะแสดง values ออกมา ดังตัวอย่างต่อไปนี้

```
1 >>> stocks.keys()
2 dict_keys(['ppt', 'scb'])
3 >>> stocks.values()
4 dict_values([360,160])
```

ดังนั้น `stocks.keys()` และ `stocks.values()` คือ ลิสต์สองตัว ตัวหนึ่งเป็น keys อีกตัวเป็น values ที่ map เข้าหากัน จำไว้ว่า ค่าของลิสต์หรือ dictionary สามารถเป็นได้ทั้งลิสต์และ dictionary ด้วย

8.5 ฟังก์ชันที่รับ Parameters ได้ไม่จำกัดจำนวน

เมื่อไรก็ตามที่ต้องการสร้างฟังก์ชันที่รับ Parameters ได้ไม่จำกัดจำนวนและบอก Keywords ได้ด้วย ให้ใส่เครื่องหมายดอกจัน (*) สองครั้งไว้หน้า Parameters หมายความว่าสิ่งที่ผ่านเข้ามาทาง Parameters จะเป็น

Dictionary ดังตัวอย่างต่อไปนี้

```
1 def printall(**kwargs):
2     for key in kwargs:
3         print(key + ' ' + str(kwargs[key]))
```

ซึ่งได้ผลลัพธ์ดังนี้

```
1 >>> printall(x=1, y=2, z=3)
2 y 2
3 z 3
4 x 1
```

แต่หากต้องการให้ Arguments หรือ Parameters ที่รับเข้ามาเป็น list ให้ใส่เครื่องหมายดอกจัน (*) หนึ่งครั้งไว้หน้า Parameters ดังตัวอย่างต่อไปนี้

```
1 def printall(*args):
2     for x in args:
3         print(x)
```

ซึ่งได้ผลลัพธ์ดังนี้

```
1 >>> printall(1,2,3,4,5)
2 1
3 2
4 3
5 4
6 5
```

8.6 แบบฝึกหัด

1. ให้ dictionary มีค่าคือ {0: 10, 1: 20} เขียนโปรแกรมให้เพิ่มค่าอีกคู่เข้าไป คือ 2:30
2. ทำการรวมค่าทั้งหมดใน dictionary นี้ ('cats':100, 'dogs':60, 'pigs':300)
3. ทำการเชื่อมต่อ dictionary ทั้ง 3 ชุดนี้ให้เป็นชุดเดียว

- `dic1={1:10, 2:20}` `dic2={3:30, 4:40}` `dic3={5:50,6:60}`

4. จงดำเนินการต่อไปนี้

- สร้าง dictionary ชื่อ stock มีค่าคือ

`'banana': 40, 'apple': 10, 'orange': 15, 'pear': 33`

- สร้าง dictionary ชื่อ prices มีค่าคือ

`'banana': 4, 'apple': 2, 'orange': 1.5, 'pear': 3`

- ให้คำนวณว่าถ้าขายผลไม้ได้ทั้งหมดจะได้เงินเท่าไร

5. เขียนฟังก์ชันตรวจสอบว่ามีค่าตัวเลขที่รับมาให้ key ของ dictionary หรือไม่ โดยให้ d คือ dictionary

มีค่าคือ `{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}`

บทที่ 9

ทูเปิล (Tuple)

9.1 ความหมายของ Tuple

Tuple จะคล้ายกับ List แต่สิ่งที่แตกต่างกันคือ Tuple นั้นเป็นประเภทข้อมูลที่ไม่สามารถเปลี่ยนแปลงได้ เมื่อไม่ต้องการให้ส่วนใดส่วนหนึ่งของโปรแกรมพลอไปเปลี่ยน Value ก็ควรใช้ Tuple การสร้าง Tuple นั้น จะอยู่ภายในวงเล็บ () และคั่นค่าแต่ละตัวด้วยเครื่องหมายคอมมา (,) ส่วนการเข้าถึงค่าใน Tuple ใช้ index เหมือนกับ list การสร้าง Tuple เป็นดังตัวอย่างต่อไปนี้

```
1 >>> t = 'a', 'b', 'c'
2 >>> t
3 ('a', 'b', 'c')
4 >>> t = 'a',
5 >>> t
6 ('a',)
7 >>>
```

ตัวอย่างที่แสดงให้เห็นว่า Tuple ไม่สามารถเปลี่ยนแปลงได้

```
1 >>> t = ('a', 'b', 'c')
2 >>> t[0]
3 'a'
4 >>> t[1]
5 'b'
6 >>> t[1:]
7 ('b', 'c')
8 >>> t[0] = 'z'
9 Traceback (most recent call last):
10   File '<pyshell#16>', line 1, in <module>
11     t[0] = 'z'
12   TypeError: 'tuple' object does not support item assignment
13 >>>
```

9.2 การสลับค่าของ Tuple

การสลับค่าของ Tuple สามารถเขียน $a, b = b, a$ และสามารถกำหนดค่าจาก String มาเป็น Tuple ได้ด้วยคำสั่ง `split()` เช่น `username, domain = 'support@classstart.org'.split('@')` เป็นดังตัวอย่างต่อไปนี้

```
1 >>> username, domain = 'support@classstart.org'.split('@')
2 >>> username
3 'support'
4 >>> domain
5 'classstart.org'
6 >>>
```

9.3 การเก็บค่าการดำเนินการใน Tuple

เราสามารถใส่ Tuple ในการเก็บค่าที่ได้จากการดำเนินการได้โดยตรง เช่น `floor, remainder = divmod(7, 3)` โดยที่ `floor` คือ ค่าจำนวนเต็มที่ได้จากการหาร ส่วน `remainder` คือค่าของเศษที่ได้จากการหาร เป็นดังตัวอย่างต่อไปนี้

```
1 >>> t = divmod(7,3)
2 >>> t
3 (2, 1)
4 >>> floor, remainder = divmod(7,3)
5 >>> floor
6 2
7 >>> remainder
8 1
```

ตัวอย่างการเขียนฟังก์ชันเพื่อให้ค่ากลับมาเป็น Tuple เป็นดังนี้ Source code:

```
1 def split_email(email):
2     return email.split('@')
```

ซึ่งได้ผลลัพธ์ดังนี้

```
1 >>> username, domain = split_email('support@classstart.org')
2 >>> username
```

```

3 'support'
4 >>> domain
5 'classstart.org'
6 >>>

```

9.4 ฟังก์ชัน `list()` เปลี่ยน tuple ให้เป็น list

ตัวอย่างการใช้ฟังก์ชัน `list()` เปลี่ยน tuple ให้เป็น list

```

1 >>> t = (1,2,3,4,5)
2 >>> t
3 (1,2,3,4,5)
4 >>> type(t)
5 <class 'tuple'>
6 >>> mylist = list(t)
7 >>> mylist
8 [1,2,3,4,5]
9 >>>

```

9.5 Dictionary และ Tuple

เมธอด `items()` ของ Dictionary จะให้ค่าเป็น List ของ Tuples โดย Tuples แต่ละตัวคือ Key และ Value ดังตัวอย่างต่อไปนี้

```

1 >>> d = {'ppt' : 360, 'scb' : 160}
2 d
3 {'ppt' : 360, 'scb' : 160}
4 >>> d.items()
5 dict_items([('ppt', 360), ('scb', 160)])

```

สรุปความแตกต่างในการใช้ Data Types คือถ้าหากต้องการลำดับของอักษร จะใช้ String ถ้าต้องการลำดับของค่าที่เปลี่ยนแปลงได้จะใช้ List ถ้าต้องการลำดับของค่าที่เปลี่ยนแปลงไม่ได้จะใช้ Tuple ถ้าต้องการคู่ลำดับของ Key กับ Value จะใช้ Dictionary

9.6 แบบฝึกหัด

1. จงเขียนโปรแกรมสร้าง Tuple มีค่าดังต่อไปนี้ ("tuple", False, 3.2, 1) และแสดงค่าออกมา
2. จงเขียนโปรแกรมสร้าง Tuple มีค่าดังต่อไปนี้ 4 8 3 แล้วทำการแยกค่าแต่ละค่าให้เป็นตัวแปรแต่ละตัว แล้วให้คำนวณผลรวมของตัวแปรทั้งหมด
3. จงเขียนโปรแกรมสร้าง Tuple มีค่าดังต่อไปนี้ 4 6 2 8 3 1 แล้วให้แปลง Tuple เป็น List และการเพิ่มเลข 30 เข้าไป แล้วให้แปลง list กลับมาเป็น Tuple ทำการ Print Tuple นั้น
4. จงเขียนโปรแกรมแปลง Tuple ให้เป็น String โดยให้ Tuple มีค่าคือ ('e', 'x', 'e', 'r', 'c', 'i', 's', 'e', 's')
5. จงเขียนโปรแกรมตรวจสอบว่ามีข้อมูลอยู่ใน Tuple หรือไม่ โดยให้ Tuple มีค่าคือ ("w", 3, "r", "e", "s", "o", "u", "r", "c", "e")

บทที่ 10

การจัดการไฟล์ (Files)

10.1 ความหมายของไฟล์

ไฟล์คือพื้นที่เก็บข้อมูลบนคอมพิวเตอร์ ไฟล์มีหลายประเภทตามการใช้งาน เช่น ไฟล์ Microsoft Word ไฟล์เพลง และไฟล์ VDO เป็นต้น โดยทั่วไปไฟล์มี 2 ประเภท คือ Text files และ Binary files โดย Text files เป็นไฟล์ที่เก็บชุดข้อความซึ่งเปิดอ่านได้ ส่วน Binary files จะอยู่ในรูปแบบของ Binary form เพื่อให้คอมพิวเตอร์ทำงาน

การเขียนโปรแกรมเพื่อใช้งานไฟล์ เช่น อ่านไฟล์ เขียนไฟล์ สร้างไฟล์ และแก้ไขไฟล์ โดยในภาษา Python จะต้องเรียกใช้โมดูล os คือ ระบบปฏิบัติการ (Operating Systems) os module มีความสามารถหลายอย่าง ศึกษาเพิ่มเติมที่ <https://docs.python.org/3.5/library/os.html>

10.2 การทำงานกับ Directories

การเรียกใช้โมดูล os ต้องใช้คำสั่ง import os ก่อนแล้วจะสามารถใช้คำสั่งในโมดูลได้ เช่น `os.getcwd()` ใช้แสดง Directory ที่กำลังทำงานอยู่ ส่วนคำสั่ง `os.chdir()` คือการเปลี่ยนตำแหน่งการทำงานของ Directory ดังตัวอย่างต่อไปนี้

```
1 >>> import os
2 >>> os.getcwd()
3 'c:\Users\janta\AppData\Local\Programs\Python\Python37-32'
4 >>> os.chdir('c:\Users\janta\Desktop')
```

คำสั่งอื่นๆ เช่น

- `os.mkdir('dirname')`
- `os.rmdir('dirname')`

- `os.remove('filename')`
- `os.path.abspath('file.txt')`
- `os.path.exists('file.txt')`
- `os.path.isdir('file.txt')`
- `os.path.isdir(os.getcwd())`
- `os.listdir(os.getcwd())`

```
1 >>> os.path.isdir(os.getcwd())
2 True
3 >>> os.listdir(os.getcwd())
4 ['DLLs', 'Doc', 'include', 'Lib', 'libs', 'LICENSE.txt', 'NEWS.txt',
5 'python.exe', 'python3.dll', 'python38.dll', 'pythonw.exe',
6 'Scripts', 'tcl', 'Tools', 'vcruntime140.dll']
7 >>>
```

10.3 การเปิดไฟล์

ฟังก์ชัน `open()` มีไว้เพื่อเปิดไฟล์ ก่อนที่จะเริ่มทำงานกับไฟล์ทุกครั้งจะต้องทำการเปิดไฟล์ก่อน โดยมีรูปแบบการเขียนคือ `fout = open(filename, flag)` ซึ่ง Filename คือ ชื่อไฟล์ที่ต้องการเปิด ส่วน Flag คือ รูปแบบในการเปิดไฟล์ ซึ่งมีหลายแบบ เช่น

- `r` ใช้อ่านข้อมูลอย่างเดียว
- `w` ใช้เขียนข้อมูลลงในไฟล์ใหม่ที่ไม่ได้สร้างไว้ก่อนหน้า
- `a` ใช้เขียนต่อท้ายไฟล์เดิม

และทุกครั้งหลังใช้งานไฟล์เสร็จแล้ว จะต้องทำการปิดไฟล์ด้วยคำสั่ง `close()` ดังตัวอย่างต่อไปนี้

```
1 import os
2 os.chdir('c:\textbackslash\Users\janta\Desktop')
3 fout = open('file.txt', 'w')
4 fout.write('The first line. \n')
5 fout.write('The second line. \n')
6 fout.write('The thrid line. \n')
7 fout.close()
```

10.4 การอ่านไฟล์

คำสั่ง `read()` จะทำการอ่านข้อมูลทั้งหมดในไฟล์เพียงครั้งเดียว และข้อมูลที่อ่านได้จะเป็น String ส่วนคำสั่ง `readline()` จะทำการอ่านข้อมูลที่ละบรรทัดแล้วเก็บไว้เป็น String ที่ละบรรทัดใน List ดังตัวอย่างต่อไปนี้

```
1 fin = open('file02.txt', 'r')
2 x = fin.read()
3 fin.close()
4 print(x)
```

10.5 การจัดการข้อผิดพลาด (Error)

การเขียนโปรแกรมเพื่อให้สามารถจัดการเหตุการณ์ที่คาดไม่ถึงได้ อาทิ เปิดไฟล์ไม่ได้เพราะ Hard disk มีปัญหา หรือ Network มีปัญหา สามารถเขียนโปรแกรมให้แสดง Error message ออกมาได้ โดยใช้ Try and Except statements สำหรับ **FileNotFoundError** เช่น ในการเปิดไฟล์ที่ยังไม่ได้สร้างไฟล์ไว้ก่อนหน้า แล้วต้องการให้ Error message แสดงออกมาว่า File not found! ดังตัวอย่าง FileNotFoundError ต่อไปนี้

```
1 try:
2     fin = open('abcdef.txt')
3 except FileNotFoundError:
4     print('File not found.')
```

ซึ่งได้ผลลัพธ์ดังนี้

```
1 >>>
2 File not found.
3 >>>
```

นอกจากนี้ยังมี **finally** statement จะทำงานท้ายสุดไม่ว่าจะประสบความสำเร็จหรือไม่ก็ตาม ดังตัวอย่างต่อไปนี้ อ่านเพิ่มเติมเรื่อง Exception Handling ได้ที่ <https://docs.python.org/3/library/exceptions.html#builtin-exceptions>

```
1 try:
2     fin = open('file.txt')
3 except:
4     print('File not found.')
5 finally:
6     print('Finally...everything is ok.')
```

ซึ่งได้ผลลัพธ์ดังนี้

```
1 >>>
2 Finally...everything is ok.
3 >>>
```

ผู้เขียนโปรแกรมสามารถสั่งให้เกิด Exception ขึ้นเองได้ โดยการใช้คำสั่ง **raise** ตัวอย่าง Raise exception เป็นดังนี้

Source code:

```
1 def print_james(name):
2     if name == 'James':
3         print('Hello James.')
4     else:
5         raise Exception('Name is not James.')
6
7 print_james('Jake')
```

ซึ่งได้ผลลัพธ์ดังนี้

```
1 Traceback (most recent call last):
2   File "C:\Users\janta\OneDrive\Documents\Python 1_63\code01.py",
3   line 30, in <module>
4     print_james('Jake')
5   File "C:\Users\janta\OneDrive\Documents\Python 1_63\code01.py",
6   line 28, in print_james
7     raise Exception('Name is not James.')
8 Exception: Name is not James.
```

10.6 ฐานข้อมูลแบบ Key-Value

Databases คือ ไฟล์แบบ Binary ที่เก็บข้อมูลในเครื่องคอมพิวเตอร์ที่เก็บในลักษณะ Key และ Value เหมือน Dictionary โดยจะต้องมีการ import โมดูล dbm ก่อนซึ่งเป็นการจัดการเกี่ยวกับ Database และ

ผลลัพธ์ที่ได้จากการอ่านไฟล์ Database จะมีการแสดงตัวอักษร b ไว้ด้านหน้าเพื่อแสดงความเป็น Binary

ตัวอย่าง Key/Value Databases เป็นดังต่อไปนี้

```

1 import dbm
2 db = dbm.open('stocks.db', 'c')
3 db['ptt']='360'
4 db['scb']='160'
5 print(db['ptt'])
6 print(db['scb'])
7 db.close()

```

ซึ่งทำให้ได้ผลลัพธ์ดังนี้

```

1 >>>
2 b'360'
3 b'160'
4 >>>

```

การ Pickling เซฟข้อมูลเป็นอย่างอื่นนอกจาก text ดังนั้นให้แปลงอย่างอื่นให้เป็น text แล้วแปลงเป็น object เพื่อดึงกลับขึ้นมา

เริ่มต้นด้วยการ `import pickle` เช่น ให้ list ชื่อว่า t1 แล้วทำการ `dumps(t1)` ให้กลายเป็น string ตอนนี้จะสามารถเอา string ไปเก็บในไฟล์ข้อมูลหรือเก็บใน key-value database ก็ได้ และเมื่อต้องการเรียกนำกลับมาใช้ในสภาพเดิมให้ใช้คำสั่ง `loads(s)` ดังตัวอย่างต่อไปนี้

```

1 >>> import pickle
2 >>> t = [1,2,3,4,5,6,7]
3 >>> t
4 [1,2,3,4,5,6,7]
5 >>> t1 = t
6 >>> t1
7 [1,2,3,4,5,6,7]
8 >>> s = pickle.dumps(t1)
9 >>> s
10 b'\x80\x04\x95\x13\x00\x00\x00\x00\x00\x00\x00'
11   \x94(K\x01K\x02K\x03K\x04K\x05K\x06K\x07e.'
12 >>> t2 = pickle.loads(s)
13 >>> t2
14 [1, 2, 3, 4, 5, 6, 7]

```

```
15 >>> id(t1)
16 66507840
17 >>> id(t2)
18 63748456
```

10.7 การให้ Python เรียกใช้โปรแกรมอื่น

การให้ Python เรียกใช้โปรแกรมอื่นได้ (Piping) โดยใช้โมดูล `os` และใช้ฟังก์ชัน `popen()` อาทิ ตั้งตัวแปรชื่อ `cmd` เรียกใช้คำสั่ง `dir` ของ Windows OS ด้วยคำสั่ง `popen` แล้วนำผลจากคำสั่งนั้นมาเก็บเป็น String ไว้ในตัวแปรชื่อ `result`

ตัวอย่างการให้ Python เรียกใช้โปรแกรมอื่นได้ เป็นดังต่อไปนี้

```
1 import os
2 os.chdir('c:\\users\\janta\\Desktop')
3 cmd = 'dir'
4 fp = os.popen(cmd)
5 result = fp.read()
6 fp.close()
7 print(result)
```

Result:

```
1 Volume in drive C has no label.
2 Volume Serial Number is 2E09-DC35
3 Directory of c:\users\janta\Desktop
4 07/05/2020  01:34 PM    <DIR>          .
5 07/05/2020  01:34 PM    <DIR>          ..
6 07/03/2020  05:46 PM                52,422 560gra01.png
7 07/05/2020  01:34 PM                29,887 5f83b4e260d668989b715e8.jpg
8 07/02/2020  07:04 PM                299,334 latexsheet-a4.pdf
9 07/03/2020  06:22 PM            14,832,375 Manual_MS-Teams_Full.pdf
10 07/03/2020  06:11 PM                2,394 Microsoft Teams.lnk
11                5 File(s)          15,216,412 bytes
12                2 Dir(s)   90,572,718,080 bytes free
```

10.8 แบบฝึกหัด

1. เขียนฟังก์ชันอ่าน text file ทั้งไฟล์
2. เขียนฟังก์ชันอ่านไฟล์ n บรรทัดแรกใน text file
3. เขียนฟังก์ชันอ่านไฟล์ที่ละบรรทัดแล้วเก็บไว้ใน list
4. สร้าง text file ขึ้นมา แล้วเพิ่มบรรทัดใหม่หนึ่งบรรทัดว่า Welcome to my class. แล้ว print เนื้อหาออกมาทั้งหมด

บทที่ 11

Object-Oriented Programming (OOP)

11.1 ความหมายของ OOP (Object-Oriented Programming)

OOP (Object-Oriented Programming) หมายถึงการเขียนโปรแกรมเชิงวัตถุหรือเขียนโปรแกรมแบบออบเจกต์ ซึ่งมีวิธีการของการจัดโครงสร้างโปรแกรมเพื่อให้คุณสมบัติและพฤติกรรมหรือการกระทำรวมอยู่ในแต่ละวัตถุ (Lutz, 2011) ตัวอย่างเช่น วัตถุอาจเป็นตัวแทนของบุคคลที่มีคุณสมบัติ คือ ชื่อ นามสกุล อายุ ที่อยู่ และมีพฤติกรรม เช่น การเดิน การพูด การหายใจ และการวิ่ง เป็นต้น

11.2 คลาส (Classes) และ ออบเจกต์ (Objects)

คลาส (Classes) เปรียบเสมือนแบบพิมพ์เขียวหรือแม่พิมพ์ที่ใช้สร้างออบเจกต์ (Objects) ในคลาสจะกำหนดรูปแบบของข้อมูลหรือแอตทริบิวต์หรือตัวแปร (Attributes/Properties/Characteristics) และเมธอด (Methods/Behaviors) ตัวอย่างเช่น คลาสชื่อ `pet` มีแอตทริบิวต์คือ `legs` ส่วน `wolf` คือ ออบเจกต์ที่สร้างขึ้นมาจากคลาส `pet` ดังตัวอย่างต่อไปนี้

```
1 class Pet: # define a class pet
2     legs = 0 # a property for the class
3     wolf = Pet() # create an object of the class
4     wolf.legs # access to a property of the object
```

11.3 การสร้างคลาส

การสร้างคลาสในภาษา Python ต้องใช้คำสั่ง `class` ตามด้วยชื่อคลาส ดังตัวอย่างต่อไปนี้

```
1 class Pet: #define a class pet
2     legs = 0 #a property for the class
```

11.4 การสร้างออบเจกต์

หลังจากที่สร้างคลาสแล้ว ก็จะสามารถสร้างตัวแปรออบเจกต์จากคลาสได้ เช่น `wolf = Pet()` โดยออบเจกต์นี้จะมีแอตทริบิวต์คือ `legs` และสามารถเข้าถึงแอตทริบิวต์ของออบเจกต์ได้โดยใช้เครื่องหมายจุด (.) ดังตัวอย่างต่อไปนี้

```
1 wolf = Pet() # create an object of the class
2 print(wolf.legs) # access to a property of the object
```

11.5 ฟังก์ชัน `__init__()`

ฟังก์ชัน `__init__()` เรียกว่าเป็นคอนสตรัคเตอร์ (Constructor) ซึ่งถูกเรียกใช้อัตโนมัติทุกครั้งในการกำหนดค่าให้แก่แอตทริบิวต์ของออบเจกต์และการดำเนินการต่างๆ ที่จำเป็นเมื่อต้องสร้างออบเจกต์ขึ้นมา และสำหรับพารามิเตอร์ `self` มีไว้เพื่อการเข้าถึงตัวแปรต่างๆ ที่เป็นของคลาส โดยที่ไม่จำเป็นต้องใช้คำว่า `self` ก็ได้ สามารถตั้งเป็นคำอื่นได้ ดังตัวอย่างต่อไปนี้

```
1 class Person:
2     name = None
3     gender = None
4
5     #Define the constructor
6     def __init__(self, n, g):
7         self.name = n
8         self.gender = g
9
10    #Main code starts here
11    p1 = Person('Mike', 'male') # create object p1
12    p2 = Person('Jan', 'female') # create object p2
```

11.6 การสร้างเมธอดของออบเจกต์

เมธอดในออบเจกต์ก็คือฟังก์ชันที่เป็นของออบเจกต์นั้น ในตัวอย่างเป็นการสร้างเมธอด `foo` เพื่อทำการ `print` คำว่า `Hello` แล้วตามด้วยชื่อ เมื่อออบเจกต์จะเรียกใช้เมธอดก็ให้เขียนชื่อออบเจกต์นั้นตามด้วยจุดแล้ว

จึงตามด้วยชื่อเมธอด ดังตัวอย่างต่อไปนี้

```

1 class Person:
2     name = None
3     gender = None
4     def __init__(self, n, g):
5         self.name = n
6         self.gender = g
7     def foo(self):
8         print('Hello' , self.name)
9
10    #Main code starts here
11    p1 = Person('Mike', 'male')
12    p2 = Person('Jan', 'female')
13    p1.foo()
14    p2.foo()

```

ซึ่งได้ผลลัพธ์ดังนี้

```

1 Hello Mike
2 Hello Jan

```

11.7 การแก้ไขค่าของแอตทริบิวต์ของออบเจกต์

การแก้ไขค่าของแอตทริบิวต์ของออบเจกต์สามารถทำได้เหมือนการกำหนดค่าของตัวแปร ดังตัวอย่างต่อไปนี้

```

1 >>> p1.name = 'Ton-mike'
2 >>> p1.foo()
3 Hello Ton-mike

```

11.8 การลบแอตทริบิวต์ของออบเจกต์

การลบแอตทริบิวต์ของออบเจกต์ให้ใช้คำสั่ง del ตามด้วยแอตทริบิวต์ของออบเจกต์นั้น ดังตัวอย่างต่อไปนี้

```
1 >>> del p1.name
2 >>> p1.foo()
3 Hello None
4 >>>
```

11.9 การลบออบเจกต์

การลบออบเจกต์ทำได้โดยใช้คำสั่ง `del` ตามด้วยชื่อของออบเจกต์ได้เลย ดังตัวอย่างต่อไปนี้

```
1 >>> del p1
2 >>> p1
3 Traceback (most recent call last):
4   File "<pyshell\#29>", line 1, in <module>
5     p1
6   NameError: name 'p1' is not defined
7 >>>
```

11.10 การสืบทอดคลาส (Class Inheritance)

การสืบทอดคลาส (Class Inheritance) คือการที่คลาสหนึ่งสามารถสืบทอดเมธอดและแอตทริบิวต์ ของคลาสนั้นไปยังคลาสอื่นได้ เรียกคลาสที่เป็นฐานหรือให้การสืบทอดนั้นว่าคลาสแม่ (Parent class) และ เรียกคลาสที่ได้รับการสืบทอด ว่าคลาสลูก (Child class) ตัวอย่าง คลาสแม่คือ `SchoolMember` ได้มีการ สืบทอดเมธอดและ แอตทริบิวต์ ไปยังคลาสลูกคือคลาส `Teacher` หลังจากนั้นคลาสลูกก็จะสามารถใช้เมธอด และแอตทริบิวต์ของคลาสแม่ได้ ดังตัวอย่างต่อไปนี้

```
1 #Define the class Teacher. It inherits from class SchoolMember
2 class Teacher(SchoolMember):
3     def __init__(self, name, age, salary):
4         SchoolMember.__init__(self, name, age)
5         self.salary = salary
6
7 #Main code
8 teacher1 = Teacher('Mr. Mike Piyawat', 47, 50000)
9 teacher2 = Teacher('Mrs. Jane Doe', 55, 120000)
10
```

```

11 print(teacher1.name)
12 print(teacher1.age)
13 print(teacher1.salary)
14 print('\n')
15 print(teacher2.name)
16 print(teacher2.age)
17 print(teacher2.salary)

```

ซึ่งได้ผลลัพธ์ดังนี้

```

1 >>>
2 Mr. Mike Piyawat
3 47
4 50000
5
6
7 Mrs. Jane Doe
8 55
9 120000
10 >>>

```

11.11 แบบฝึกหัด

1. เขียนคลาสชื่อว่า Pet ประกอบด้วย

- คอนสตรัคเตอร์ (Constructor)
- แอตทริบิวต์ genre
- แอตทริบิวต์ legs
- เมธอด **start_running** แสดงผลทางหน้าจอว่า “Start running”
- เมธอด **stop_running** แสดงผลทางหน้าจอว่า “Stop running”

2. จากข้อ 1 ให้เขียนโปรแกรมเพื่อสร้างออบเจกต์จากคลาส Pet สองออบเจกต์ แล้วเรียกใช้เมธอดที่มี

บรรณานุกรม

- Barry, P. (2016). *Head first python: A brain-friendly guide*. Sebastopol, CA, USA: O'Reilly Media.
- Beazley, D., & Jones, B. K. (2013). *Python cookbook*. Sebastopol, CA, USA: O'Reilly Media.
- Bouras, A. S. (2019). *Python and algorithmic thinking for the complete beginner (2nd edition): Learn to think like a programmer*. Independently published.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. Cambridge, MA, USA: The MIT Press.
- Downey, A. B. (2015). *Think python: How to think like a computer scientist*. Sebastopol, CA, USA: O'Reilly Media.
- Foundation, P. S. (2019, January). *Python*. Retrieved from <https://www.python.org/>
- Guido, V. R. (2019, January). *Guido van rossum - personal home page*. Retrieved from <https://gvanrossum.github.io//help.html>
- Lubanovic, B. (2015). *Introducing python: Modern computing in simple packages*. Sebastopol, CA, USA: O'Reilly Media.
- Lutz, M. (2011). *Programming python: Powerful object-oriented programming*. Sebastopol, CA, USA: O'Reilly Media.
- Lutz, M. (2013). *Learning python*. Sebastopol, CA, USA: O'Reilly Media.
- Lutz, M. (2014). *Python pocket reference: Python in your pocket*. Sebastopol, CA, USA: O'Reilly Media.
- Ramalho, L. (2015). *Fluent python: Clear, concise, and effective programming*. Sebastopol, CA, USA: O'Reilly Media.
- Shuup. (2019, April). *25 of the most popular python and django websites*. Retrieved from <https://www.shuup.com/django/25-of-the-most-popular-python-and-django-websites/>

TIOBE. (2019, August). *The python programming language*. Retrieved from [**https://www.tiobe.com/tiobe-index/python/**](https://www.tiobe.com/tiobe-index/python/)