

การเขียนโปรแกรมภาษาไพธอน

Python



ลาวัณย์ ดุลยชาติ

คำนำ

ตำราเล่มนี้เป็นส่วนหนึ่งของรายวิชา ED-032-104 การเขียนโปรแกรมคอมพิวเตอร์ เนื้อหาเหมาะสมสำหรับผู้เริ่มต้นเขียนโปรแกรม ซึ่งยังไม่มีพื้นฐานด้านนิ่มมาก่อน เนื้อหาถูกออกแบบให้ สอดคล้องกับหลักสูตรครุศาสตรบัณฑิต สาขาวิชานวัตกรรมการจัดการเรียนรู้ วิชาเอกคอมพิวเตอร์ ในกลุ่มรายวิชาเอกบังคับ เนื้อหาของตำราถูกแบ่งออกเป็น 7 บท ประกอบด้วย บทที่ 1 แนะนำ ความรู้เบื้องต้นเกี่ยวกับภาษาไพทอน บทที่ 2 พื้นฐานการเขียนโปรแกรมภาษาไพทอน บทที่ 3 ตัว ดำเนินการ บทที่ 4 ข้อมูลแบบลำดับรายการ บทที่ 5 คำสั่งควบคุมทิศทางการทำงานของโปรแกรม บทที่ 6 พังก์ชันและโมดูล และบทที่ 7 การสร้างกราฟด้วย Matplotlib

เมื่อนักศึกษาได้ศึกษาตำราเล่มนี้แล้ว จะสามารถถูกแบบ เขียนโปรแกรมแก้โจทย์ปัญหา ทางคอมพิวเตอร์ และสามารถนำความรู้ไปประยุกต์ใช้กับภาษาคอมพิวเตอร์อื่นได้

ในส่วนท้ายของแต่ละบทเรียน เป็นส่วนของกิจกรรม สรุปท้ายบท และแบบฝึกหัดที่เป็น ประโยชน์ในการทบทวนทำความเข้าใจในเนื้อหาแต่ละบท ปัญหานี้แบบฝึกหัดครอบคลุมลักษณะ ต่างๆ ที่สำคัญที่ได้อธิบายในบทเรียน หมายเหตุที่นักศึกษาจะใช้เป็นโจทย์ที่จะเรียนรู้เพิ่มเติมด้วยตนเอง

ลาวณย์ ดุลยชาติ
พฤษจิกายน 2564

สารบัญ

บทที่ 1 แนะนำความรู้เบื้องต้นเกี่ยวกับภาษาไพธอน.....	1
1. แนะนำภาษาไพธอน	1
2. การติดตั้งและใช้งาน Python IDLE.....	3
3. การติดตั้งและใช้งาน Jupyter Notebook.....	13
4. การใช้งาน Google Colab.....	30
5. การติดตั้งโปรแกรม Visual Studio Code หรือ VS Code	35
6. สรุป	40
7. แบบฝึกหัดท้ายบท	40
บทที่ 2 พื้นฐานการเขียนโปรแกรมภาษาไพธอน.....	41
1. คำสั่งที่เขียนในโปรแกรมไพธอน (Statement Python)	41
2. การแสดงผลข้อมูลด้วยฟังก์ชัน print()	41
3. การรับข้อมูลทางคีย์บอร์ดด้วยฟังก์ชัน input().....	42
4. การเขียนคำอธิบายโค้ด	42
5. การกำหนดขอบเขตของคำสั่ง (Indentation).....	44
6. ตัวแปร (Python Variables).....	45
7. ชนิดข้อมูลแบบ Built-in (Built-in Data Types)	50
8. ข้อมูลชนิดตัวเลข (Python Numbers).....	52
9. ชนิดข้อมูลสตริง (Python Strings)	59
10. บูลีน (Python Booleans).....	70
11. สรุป	73
12. แบบฝึกหัดท้ายบท	73
บทที่ 3 ตัวดำเนินการ	74
1. ตัวดำเนินการกำหนดค่า (Python Assignment Operators).....	74
2. ตัวดำเนินการคณิตศาสตร์ (Python Arithmetic Operators).....	75
3. ตัวดำเนินการเปรียบเทียบ (Python Comparison Operators)	76
4. ตัวดำเนินการตรรกศาสตร์ (Python Logical Operators)	77
5. ตัวดำเนินการเอกลักษณ์ (Python Identity Operators)	79

6. ตัวดำเนินการตรวจสอบสมาชิก (Python Membership Operators).....	81
7. ตัวดำเนินการระดับบิต (Python Bitwise Operators)	82
8. ลำดับการประมวลผลของตัวดำเนินการ.....	83
9. ฟังก์ชันเกี่ยวกับตัวเลข (Python Math).....	84
10. กิจกรรม	86
11. สรุปท้ายบท	88
12. แบบฝึกหัดท้ายบท	89
บทที่ 4 ข้อมูลแบบลำดับรายการ	91
1. รายการข้อมูลแบบ Lists	91
2. รายการข้อมูลแบบทูเพิล (Tuples).....	96
3. รายการข้อมูลแบบเซต (Sets).....	101
4. รายการข้อมูลแบบดิกชันนารี (Dictionaries).....	107
5. สรุปท้ายบท	122
6. กิจกรรม	122
7. แบบฝึกหัดท้ายบท	125
บทที่ 5 คำสั่งควบคุมทิศทางการทำงานของโปรแกรม	126
1. คำสั่งควบคุมแบบลำดับ	126
2. คำสั่งควบคุมแบบมีเงื่อนไข(Condition Control Statement)	127
3. คำสั่งควบคุมแบบทำซ้ำ (Iteration Control Statement)	133
4. กิจกรรม	140
5. แบบฝึกหัดท้ายบท	145
บทที่ 6 ฟังก์ชันและโมดูล	148
1. ลักษณะของฟังก์ชัน ในภาษาไพธอนแบ่งฟังก์ชันออกเป็น 2 ประเภทหลักๆ คือ	148
2. การสร้างฟังก์ชันแบบไม่ส่งค่ากลับ.....	148
3. พารามิเตอร์หรืออาร์กิวเมนต์.....	149
4. ฟังก์ชันแบบส่งค่ากลับ	155
5. การเรียกใช้ฟังก์ชันแบบ Recursion	155
6. ฟังก์ชันแบบ Lamda.....	156
7. การสร้างและใช้โมดูล.....	159
9. สรุปท้ายบท	168

10. แบบฝึกหัดท้ายบท	168
บทที่ 7 การสร้างกราฟด้วย Matplotlib.....	169
1. Matplotlib คืออะไร	169
2. Matplotlib Codebase อยู่ที่ไหน	169
3. การติดตั้ง Matplotlib (Installation of Matplotlib).....	170
4. การนำเข้า Matplotlib (Import Matplotlib)	170
5. การตรวจสอบเวอร์ชัน Matplotlib (Checking Matplotlib Version)	170
6. การสร้างกราฟด้วย Matplotlib.....	170
7. สรุปท้ายบท	217
8. แบบฝึกหัดท้ายบท	217
บรรณานุกรม	220
ด้วย	222
ประวัติผู้เขียน	227

สารบัญภาพ

ภาพที่	หน้า
1 แสดง Anaconda Powershell Prompt	19
2 แสดงการเข้าใช้งาน Anaconda Powershell Prompt.....	20
3 แสดงคำแนะนำไฟล์ Anaconda	24
4 แสดงผังงานรูปแบบคำสั่ง	126
5 แสดงผังงานคำสั่ง if - else	129
6 แสดงผังงานคำสั่งแบบทำซ้ำ.....	133

สารบัญตาราง

ตารางที่	หน้า
1 คำส่วน	46
2 ชนิดข้อมูลแบบ Built-in	50
3 การกำหนดค่าให้ชนิดข้อมูล	51
4 การระบุชนิดข้อมูล	52
5 สัญลักษณ์การแสดงรูปแบบชนิดข้อมูล	65
6 อักขระ escape อื่น ๆ ที่ใช้ในไฟฟอน	67
7 String Methods	68
8 แสดงตัวดำเนินการที่ใช้ในการกำหนดค่าให้กับตัวแปร	74
9 ตัวดำเนินการทางคณิตศาสตร์	75
10 ตัวดำเนินการเปรียบเทียบ	76
11 ตัวดำเนินการตรรกศาสตร์	77
12 ตัวดำเนินการเอกลักษณ์	79
13 ตัวดำเนินการในการตรวจสอบการเป็นสมาชิกในออบเจ็ค	81
14 ตัวดำเนินการระดับบิตในภาษาไฟฟอน	82
15 ลำดับการประมวลผลของตัวดำเนินการ	83
16 เมธอดและฟังก์ชันที่ใช้กับ List	91
17 เมธอดและฟังก์ชันที่ใช้งานกับชนิดข้อมูลทุกเพล	100
18 เมธอดและคำอธิบาย	107
19 ชุดของ built-in methods	118
20 เครื่องหมายอ้างอิง	176
21 รูปแบบลักษณะเส้น	182



บทที่ 1

แนะนำความรู้เบื้องต้นเกี่ยวกับภาษาไพทอน

ในยุคปัจจุบันคอมพิวเตอร์จะสามารถทำงานได้หลายอย่าง และมีประสิทธิภาพที่สูงมาก อย่างไรก็ตามคอมพิวเตอร์ไม่ได้มีความสามารถหรือความฉลาดได้ด้วยตัวเองแต่อย่างใด แต่สิ่งที่ทำให้คอมพิวเตอร์สามารถทำงานต่าง ๆ ได้ก็คือสิ่งที่เรียกว่า “โปรแกรม” ที่คอยทำงานอยู่เบื้องหลัง ซึ่งเป็นชุดคำสั่งที่มีการกำหนดขั้นตอนที่ชัดเจนเพื่อส่งงานให้คอมพิวเตอร์ทำงานตามที่เราต้องการ ไม่ว่าจะเป็นการจัดการกับข้อมูลที่นำเข้ามาในระบบ การตัดสินใจสำหรับเงื่อนไขต่าง ๆ การประมวลผลข้อมูล การสื่อสารกับระบบภายนอก การจัดการความผิดพลาดที่เกิดขึ้น และการแสดงผลในรูปแบบต่าง ๆ เป็นต้น เมื่อว่าในงานทางด้านปัญญาประดิษฐ์ ระบบคอมพิวเตอร์จะสามารถตัดสินใจเองจนสามารถแข่งขันกับมนุษย์ได้ แต่ยังมีความจำเป็นที่จะต้องเขียนโปรแกรมในการสร้างโมเดลเพื่อสอนให้คอมพิวเตอร์สามารถเรียนรู้จากข้อมูลเองได้ ดังนั้นหากเราต้องการให้คอมพิวเตอร์ทำงานได้ก็ตาม จะต้องอาศัยการเขียนโปรแกรมเพื่อส่งงานคอมพิวเตอร์ให้ทำงานตอบสนองความต้องการของเราได้ไพฑูรย์ จึงเป็นภาษาที่ดีที่สุดที่นิยมนำมาใช้งานอย่างกว้างขวาง เนื่องจากเป็นภาษาที่ใช้งานง่าย สามารถเรียนรู้และเข้าใจได้เร็ว สามารถทำงานได้หลายลักษณะ และรองรับการทำงานหลายอุปกรณ์หลายรูปแบบ อีกทั้งในปัจจุบันยังนิยมนำมาใช้ในภาษาที่ฝึกเขียนโปรแกรมทั้งในระดับ การศึกษาขั้นพื้นฐาน และระดับอุดมศึกษา

1. แนะนำภาษาไพทอน

โปรแกรมภาษาไพทอน (Python Programming) คือ ภาษาโปรแกรมคอมพิวเตอร์ระดับสูง โดยถูกออกแบบมาให้เป็นภาษาสคริปต์ที่อ่านง่าย โดยตัดความซับซ้อนของโครงสร้างและไวยกรณ์ของภาษาออกไป ในส่วนของการแปลงชุดคำสั่งที่เราเขียนให้เป็นภาษาเครื่องไพทอน มีการทำงานแบบ Interpreter คือเป็นการแปลงชุดคำสั่งที่ละเอียดทั้งหมด เพื่อบอกเข้าสู่หน่วยประมวลผลให้คอมพิวเตอร์ทำงานตามที่เราต้องการ นอกจากนั้นภาษาโปรแกรมไพทอน ยังสามารถนำไปใช้ในการเขียนโปรแกรมได้หลากหลายประเภท โดยไม่ได้จำกัดอยู่ที่งานเฉพาะทางใดทางหนึ่ง (General-purpose language) จึงทำให้มีการนำไปใช้กันแพร่หลายในหลายองค์กรใหญ่ระดับโลก เช่น Google, YouTube, Instagram, Dropbox และ NASA เป็นต้น

สำหรับประวัติของโปรแกรมภาษาไพทอน ได้เริ่มต้นขึ้นในเดือนธันวาคมปี 1989 โดยนาย Guido van Rossum โปรแกรมเมอร์ชาวดัตช์ ในตอนนั้นทำงานอยู่ที่สถาบันวิจัยแห่งชาติ Centrum Wiskunde & Informatica (CWI) ซึ่งเป็นสถาบันวิจัยทางด้านคณิตศาสตร์และวิทยาการคอมพิวเตอร์ ในเมืองอัมสเตอร์ดัม ประเทศเนเธอร์แลนด์ ในเวลานั้น Guido ต้องพัฒนาโปรแกรมสำหรับผู้ดูแลระบบ เพื่อใช้ในโครงการ Amoeba ซึ่งเป็นโครงการเกี่ยวกับระบบปฏิบัติการแบบกระจาย (Distributed operating system) อย่างไรก็ตามเขารู้สึกว่าภาษาโปรแกรม ABC, C และ Bourne shell มีข้อจำกัดมาก many ทั้งเรื่องใช้เวลาในการพัฒนานานมากและไม่สามารถตอบโจทย์หลายประการ ดังนั้น Guido จึงได้ตัดสินใจเริ่มพัฒนาภาษาโปรแกรมระดับสูงขึ้นมาใหม่เพื่อใช้งานเองเป็นงานอดิเรก โดยนำเอาสิ่งที่ชอบในภาษา ABC มาพัฒนาลงมาในภาษาโปรแกรมไพทอน รวมถึงได้



พัฒนาส่วนอื่น ๆ เพิ่มเติมเข้าไป และในเวลาต่อมาจึงได้เผยแพร่ไฟทอน 1.0 เวอร์ชันแรกในปี 1994 หากเทียบกับภาษา Java ที่ได้ทำการเผยแพร่เวอร์ชันแรกในปี 1996 จะเห็นได้ว่าภาษาไฟทอน มีอายุมากกว่าภาษา Java ถึง 2 ปี ซึ่งมีจุดเด่นของไฟทอน ดังนี้

- สามารถใช้บนเซิร์ฟเวอร์เพื่อสร้างเว็บแอปพลิเคชัน
- สามารถใช้ร่วมกับซอฟต์แวร์เพื่อสร้างเวิร์กโฟล์ว
- สามารถเชื่อมต่อกับระบบฐานข้อมูล นอกจากนี้ยังสามารถอ่านและแก้ไขไฟล์
- สามารถใช้ในการจัดการข้อมูลขนาดใหญ่และดำเนินการทางคณิตศาสตร์ที่ซับซ้อน
- สามารถใช้สำหรับการสร้างต้นแบบอย่างรวดเร็วหรือสำหรับการพัฒนาซอฟต์แวร์ที่พร้อมใช้งานจริง
- ทำงานบนแพลตฟอร์มที่หลากหลายได้ เช่น Windows, Mac, Linux, Raspberry Pi, ฯลฯ
- ออกแบบมาเพื่อให้สามารถอ่านได้ และมีความคล้ายคลึงกับภาษาอังกฤษที่มีอิทธิพลจากคณิตศาสตร์
- มีไวยากรณ์ที่ช่วยให้นักพัฒนาสามารถเขียนโปรแกรมที่มีบรรทัดน้อยกว่าภาษาโปรแกรมอื่น
- ทำงานบนระบบตัวแปลภาษาแบบอินเตอร์พรีเทอร์ ซึ่งหมายความว่าสามารถพิมพ์คำสั่งเป็นภาษาสคริปต์ ทำงานโดยต้องทิ้งคำสั่ง และแสดงผลลัพธ์แจ้งเตือนทันทีหากมีข้อผิดพลาด ซึ่งทำได้รวดเร็วมาก
- สามารถใช้งานได้ในแบบเพรซีเดอร์, แบบ object-orientated หรือแบบ functional

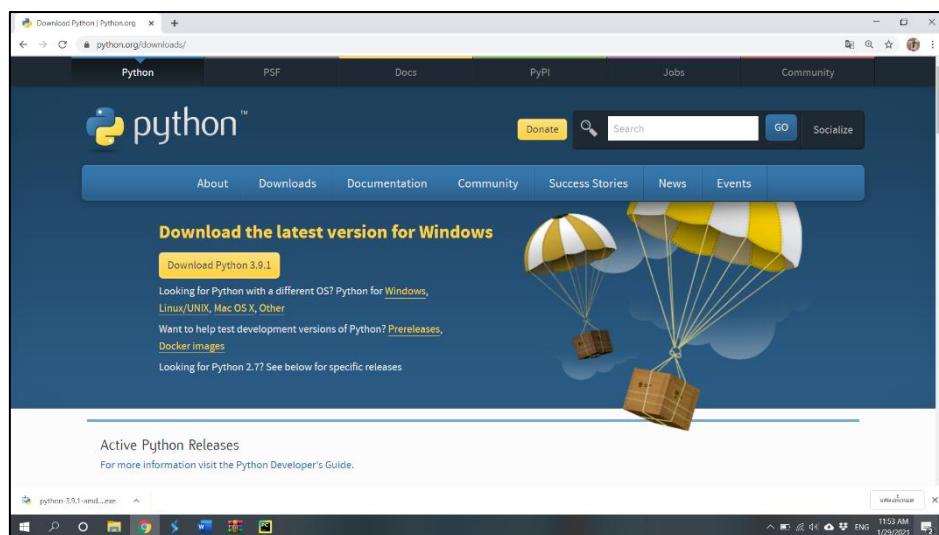
การเขียนโปรแกรมภาษาไฟทอนจะใช้เครื่องมือในการพัฒนาที่เรียกว่า ไอเดีย (Integrated Development Environment : IDE) ซึ่งประกอบด้วยเครื่องมือแก้ไขโปรแกรมต้นฉบับ (Source Code Editor) เครื่องมือแก้ไขจุดบกพร่อง (Debugger) และเครื่องมือช่วยให้โปรแกรมทำงาน (Run) โดยทั่วไปไฟทอน IDE จะทำงานตามคำสั่งได้ใน 2 โหมด คือ

1. โหมดอิมมีเดียท (immediate mode) เป็นโหมดที่ผู้ใช้จะพิมพ์คำสั่งลงไปในส่วนที่เรียกว่า เชลล์ (shell) หรือคอนโซล (console) ทิ้งคำสั่ง และตัวแปลภาษาจะแปลคำสั่ง หากไม่มีข้อผิดพลาดจะทำงานตามคำสั่งดังกล่าว
2. โหมดสคริปต์ (script mode) ในโหมดนี้ผู้เขียนโปรแกรมต้องพิมพ์คำสั่งหลายคำสั่งประกอบกันแล้วบันทึกเป็นไฟล์ไว้ก่อน เพื่อจะสั่งให้ตัวแปลภาษาทำงานตามคำสั่งตั้งแต่คำสั่งแรกจนถึงคำสั่งสุดท้าย ถ้าหากต้องการตรวจสอบความถูกต้องสามารถใช้โหมดอิมมีเดียทในการทดสอบได้

2. การติดตั้งและใช้งานไฟล์ Python IDLE

2.1 การติดตั้ง IDLE บนระบบปฏิบัติการ Windows

ขั้นที่ 1 ไปที่ <https://www.python.org> และคลิกที่ “Download” เพื่อเปิดหน้าดาวน์โหลดสำหรับ Windows



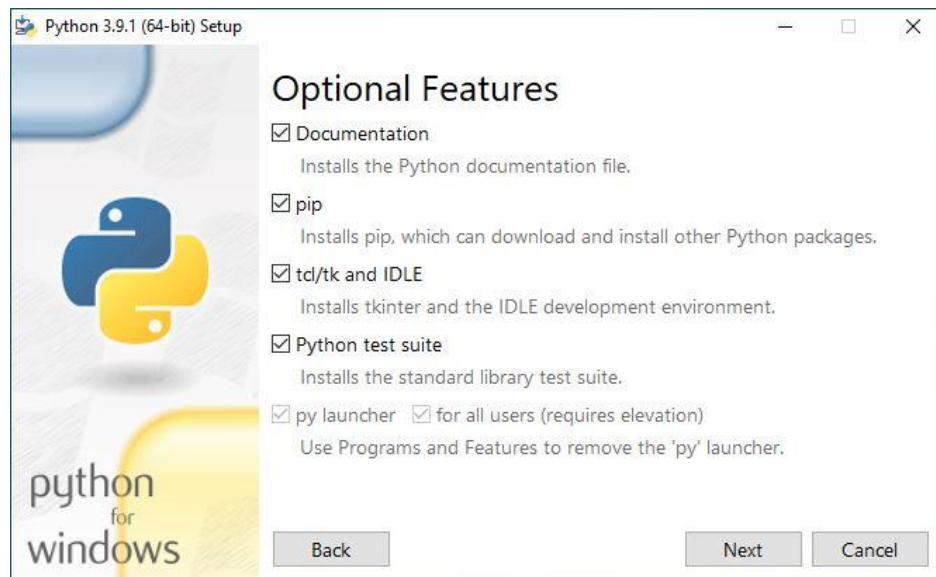
ขั้นที่ 2 หลังจากดาวน์โหลดเสร็จแล้วให้เปิดไฟล์. exe เพื่อติดตั้ง Python 3.9.1





การเขียนโปรแกรมภาษาไพทอน (Python Programming)

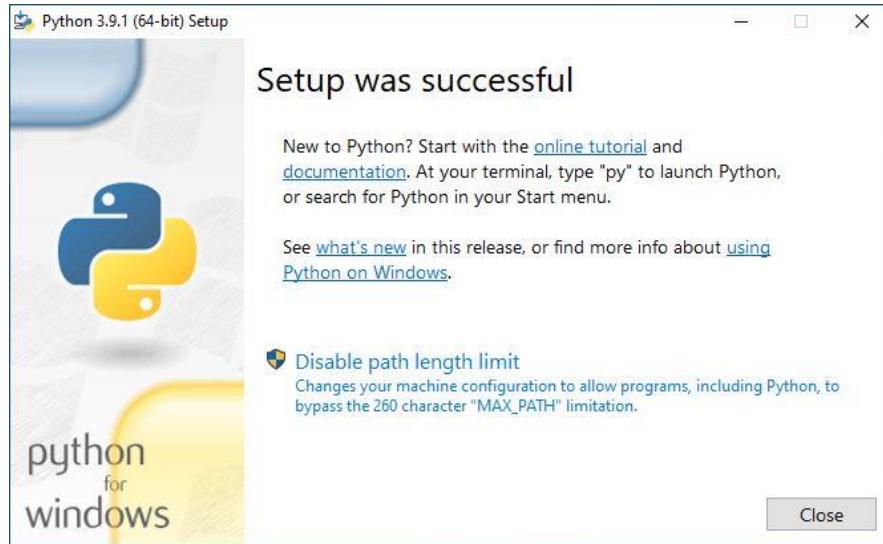
ขั้นที่ 3 เปิด “Set-up Window” พบตัวเลือกการติดตั้งสองตัวเลือก “Install Now” (ติดตั้งเดียว) และ “Customize Installation” (กำหนดค่าการติดตั้งเอง)



 หากต้องการเปลี่ยนตำแหน่งการติดตั้งและต้องการเพิ่มคุณสมบัติเฉพาะบางอย่างให้คลิกที่ “Customize Installation”

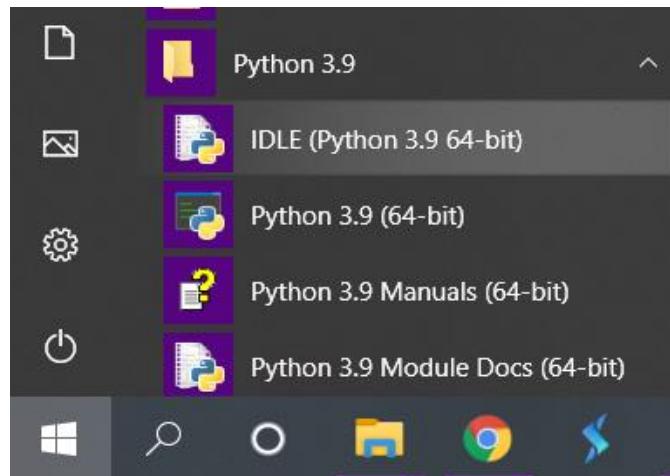


ขั้นที่ 4 ตอนนี้หากไม่ต้องการปรับแต่งการติดตั้งให้เลือก “Install Now” และก่อนที่จะเลือกติดตั้งทันทีต้องทำเครื่องหมายในช่องที่เรียกว่า “Add Python 3.9.1 to PATH” “เพิ่ม Python 3.9.1 ไปที่ PATH” จากนั้นรอเพื่อให้กระบวนการติดตั้งเสร็จสมบูรณ์



2.2 การใช้งาน IDLE เป็นต้น

เนื่องจาก IDLE ไม่ได้สร้าง Shortcut บนหน้า Desktop เพื่อเข้าใช้งานโปรแกรมให้คลิกดังภาพ



IDLE จะแบ่งเป็น 2 โหมดดังต่อไปนี้

- Shell Window ใช้สำหรับเขียนโค้ดสั้น และทดสอบผลลัพธ์
- Edit Window ใช้ในการเขียนโค้ดคำสั่งที่ยาวๆ แบบต่อเนื่องกันตามปกติ



IDLE Shell 3.9.1

File Edit Shell Debug Options Window Help

Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

>>> |

Ln: 3 Col: 4

ทดลองเขียนคำสั่งง่ายๆ ดังภาพ

IDLE Shell 3.9.1

File Edit Shell Debug Options Window Help

Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

```
>>> print ("This is my first python")
This is my first python
>>> x = 2
>>> y = 3
>>> z = x + y
>>> |
```

Ln: 8 Col: 4

2.3 การติดตั้งและใช้งาน Pycharm

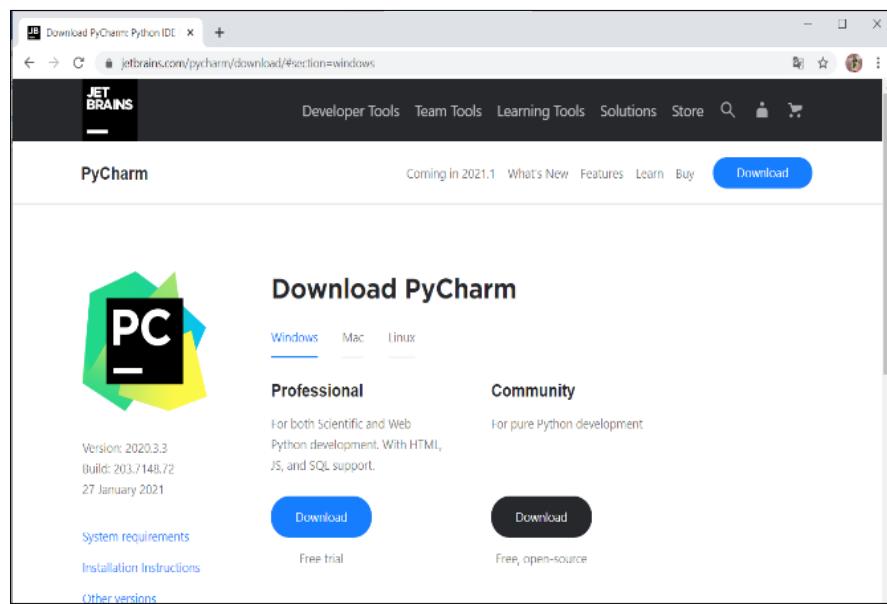
PyCharm มีให้เลือกสามรุ่น:

- **Community** (ฟรีและโอเพ่นซอร์ส): สำหรับนักพัฒนาไฟทอน มีความช่วยเหลือด้านโค้ดการปรับโครงสร้างใหม่การแก้ไขจุดบกพร่องด้วยภาพและการรวมการควบคุมเวอร์ชัน
- **Professional** (ชำระเงิน): สำหรับระดับมืออาชีพ พัฒนาเว็บ และวิทยาศาสตร์ข้อมูล รวมถึงความช่วยเหลือด้านโค้ดการปรับโครงสร้างการแก้ไขจุดบกพร่องด้วยภาพการรวมการควบคุมเวอร์ชันการกำหนดค่ารายละเอียดการปรับใช้การสนับสนุนเว็บเฟรมเวิร์กยอดนิยมเช่น Django และ Flask การสนับสนุนฐานข้อมูลทางวิทยาศาสตร์ เครื่องมือ (รวมถึงการรองรับโนํตบุ๊ก Jupyter) เครื่องมือข้อมูลขนาดใหญ่
- **Edu** (ฟรีและโอเพ่นซอร์ส): สำหรับการเรียนรู้ภาษาโปรแกรมและเทคโนโลยีที่เกี่ยวข้องด้วยเครื่องมือทางการศึกษาแบบบูรณาการ

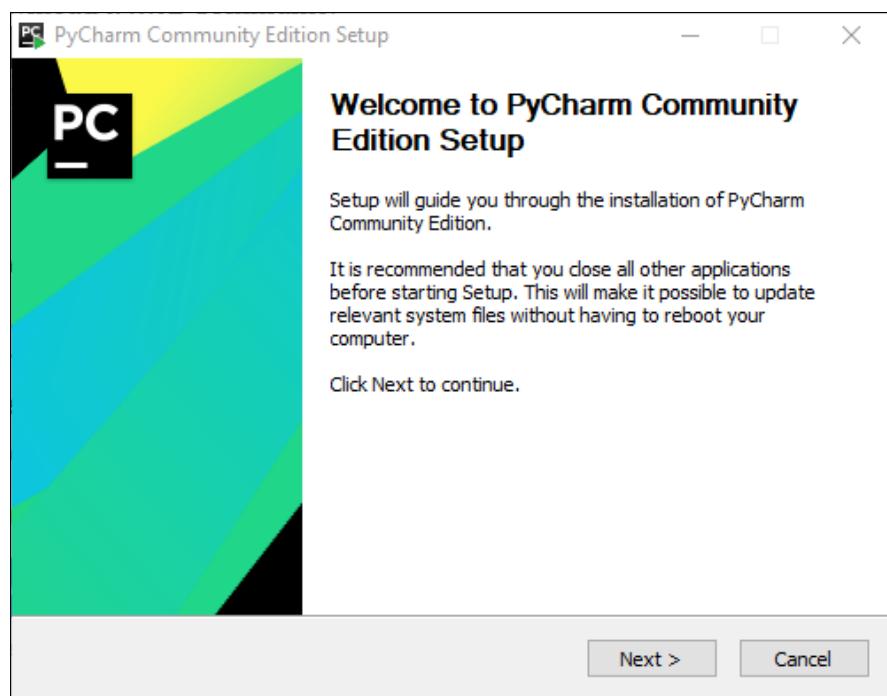
2.3.1 การติดตั้ง Pycharm

ขั้นที่ 1 ไปที่ <https://www.jetbrains.com/pycharm/download/#section=windows>

สามารถเลือกติดตั้ง Pycharm Edu หรือ Pycharm Community หรือติดตั้งทั้ง 2 โปรแกรม คลิกที่ “Download”

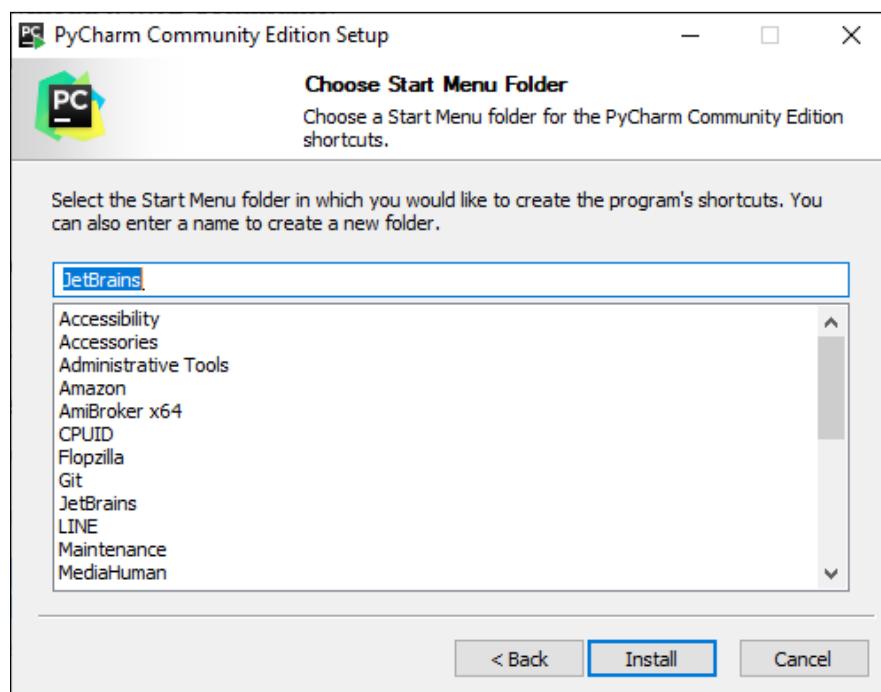


ขั้นที่ 2 หลังจากที่ Download เรียบร้อยแล้ว คลิก Run จะปรากฏหน้าต่างเพื่อติดตั้ง คลิก Next

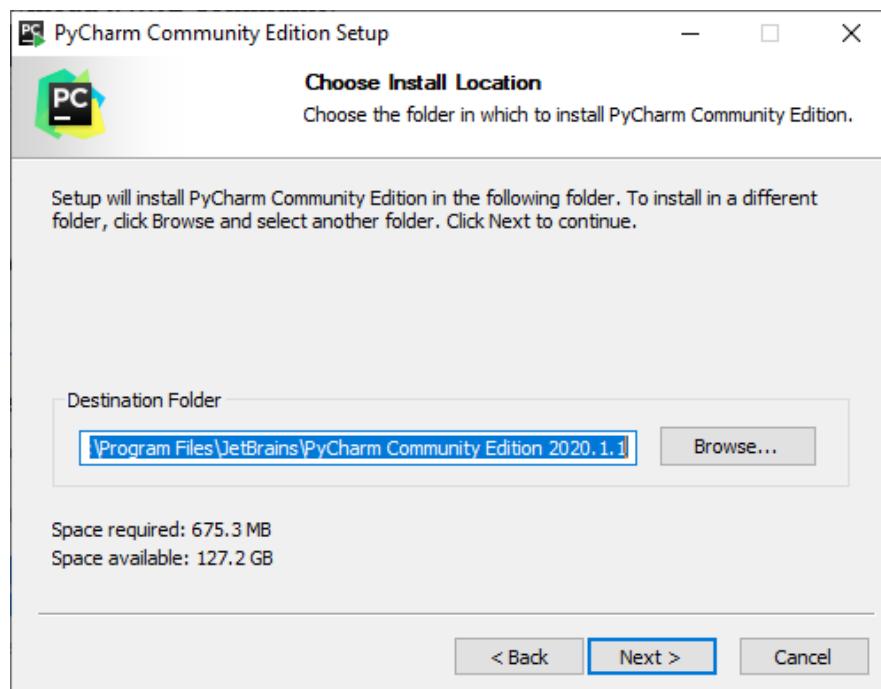




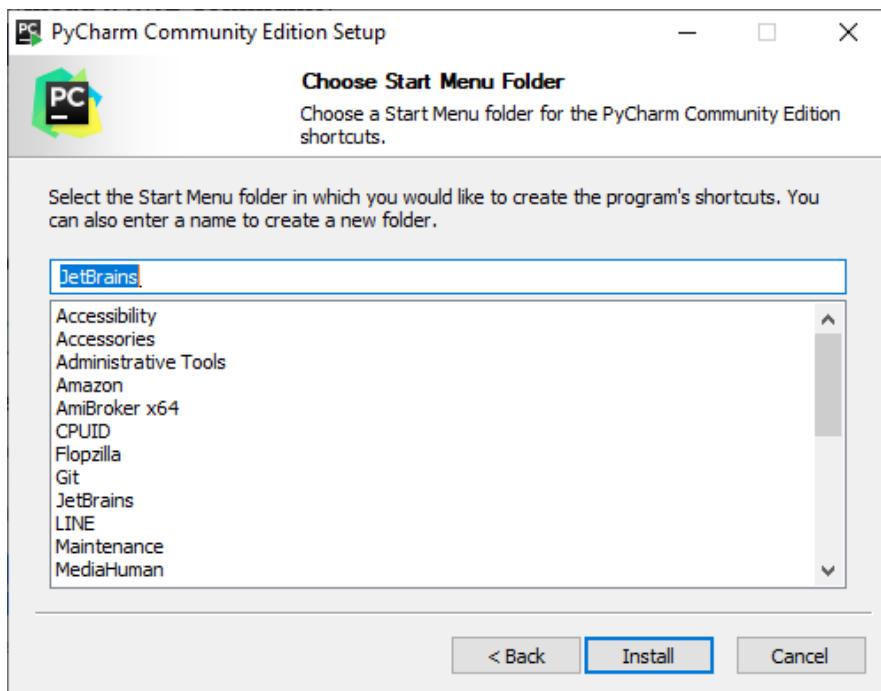
ขั้นที่ 3 คลิก Install



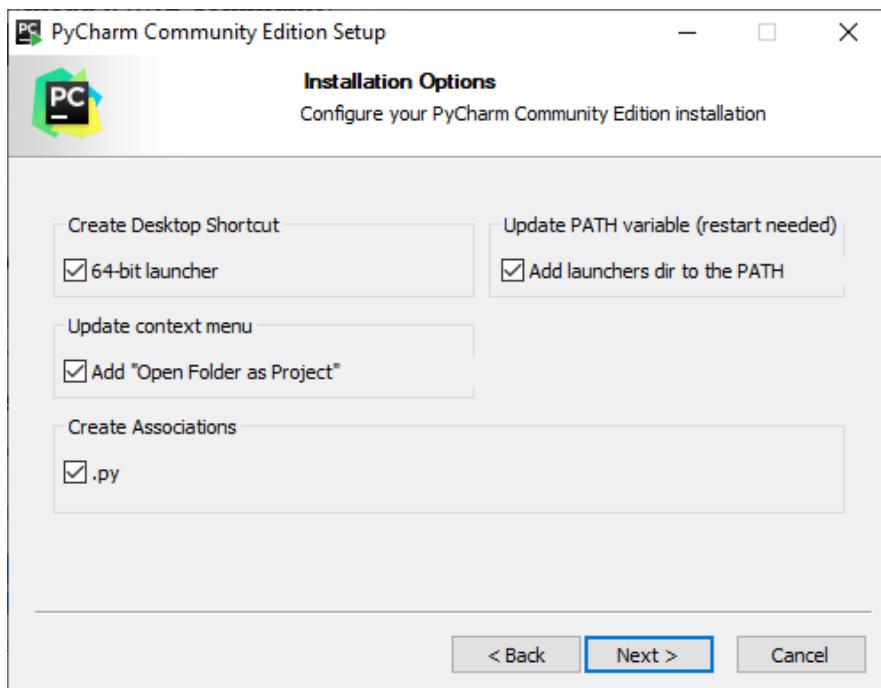
ขั้นที่ 4 เลือก folder ที่จะติดตั้ง โดยสามารถเลือก folder นอกเหนือจากโปรแกรมกำหนดมาให้โดยเดfault แล้วเลือกไฟเดอร์ที่ต้องการ หรือจะเลือกตามค่าเริ่มต้นที่โปรแกรมกำหนดมาให้ก็ได้จากนั้น กด คลิก Next



ขั้นที่ 5 คลิก Install

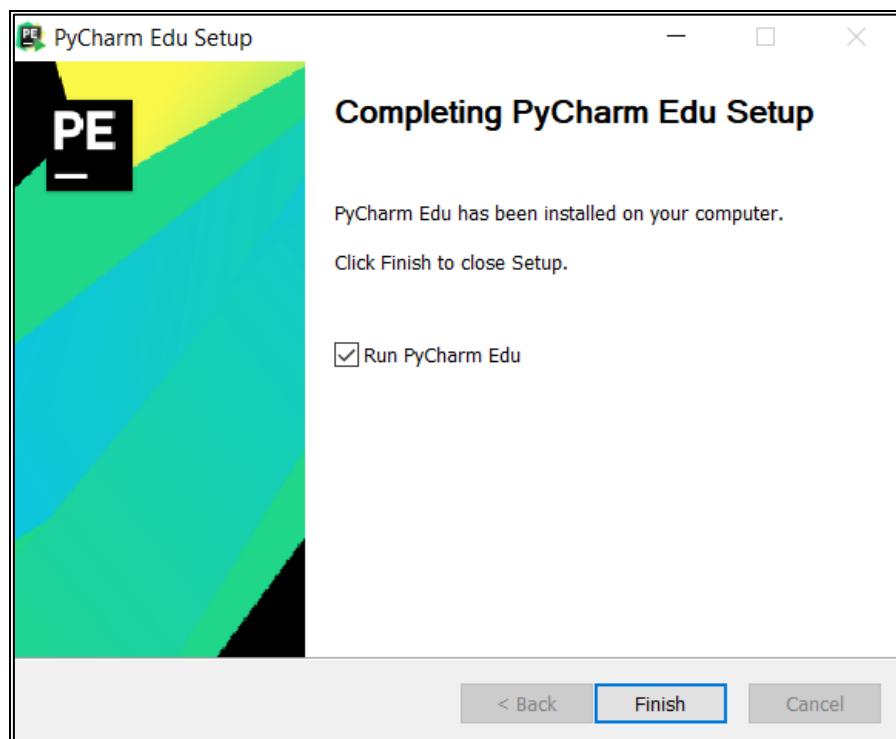


ขั้นที่ 6 คลิก Next

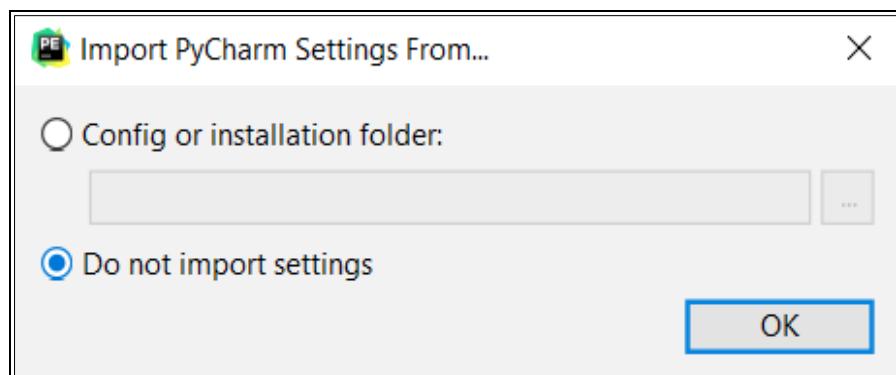




ขั้นที่ 7 ติดตั้งเสร็จสมบูรณ์ คลิก Finish

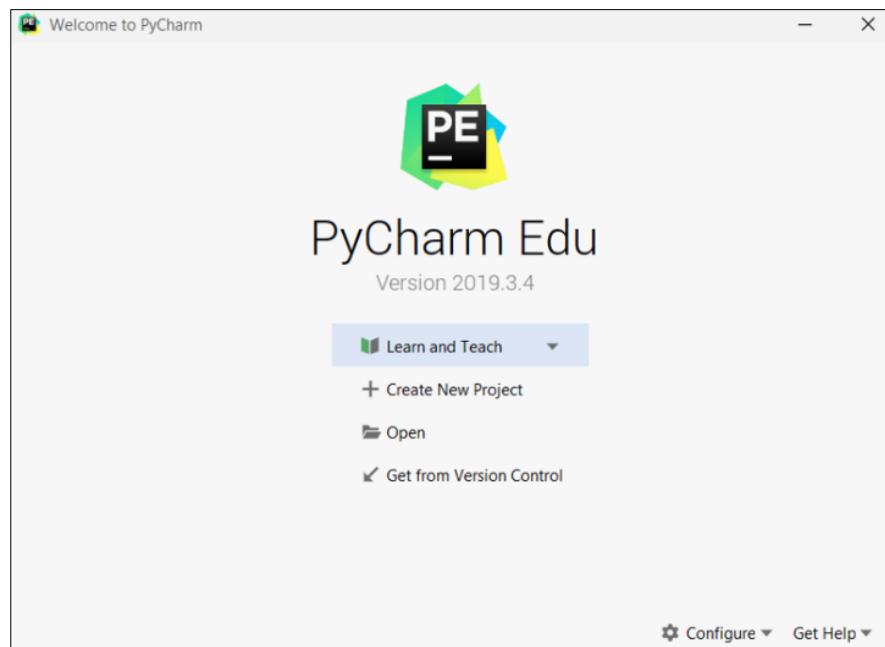


ขั้นที่ 8 ถ้าใช้งาน Pycharm ครั้งแรกให้เลือก Do not import settings

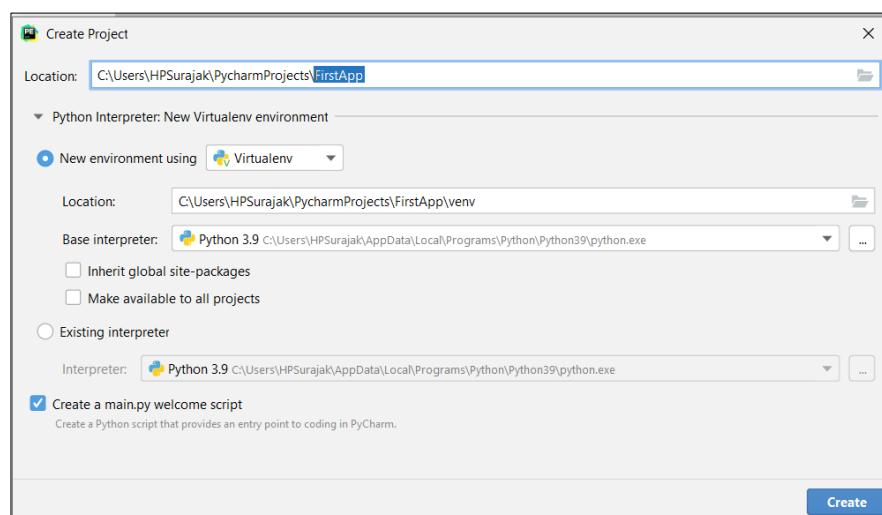


2.3.2 การใช้งาน Pycharm Edu

ขั้นที่ 1 เมื่อปรากฏหน้าจอ Welcome to PyCharm เลือก Create New Project

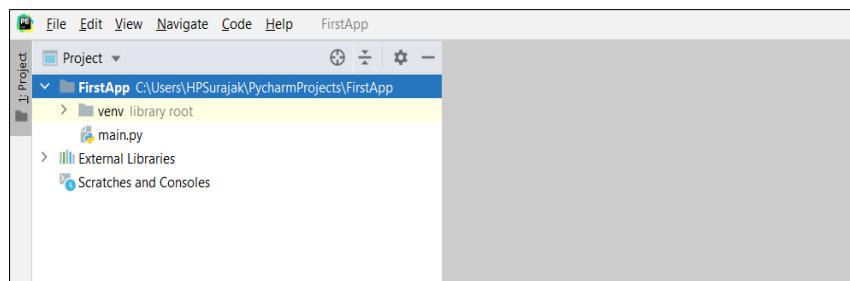


ขั้นที่ 2 เลือกทำแน่งที่เก็บไฟล์ project

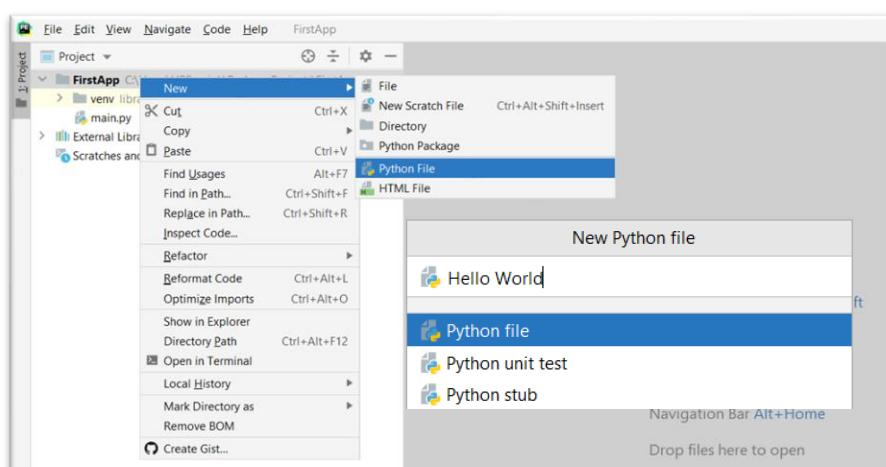




ขั้นที่ 3 หน้าจอโปรแกรม

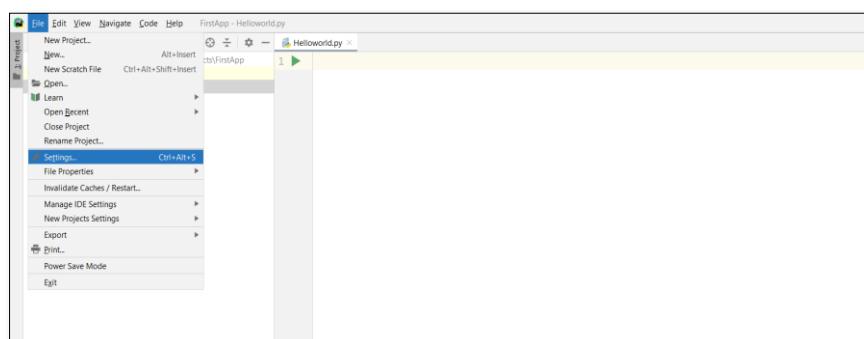


ขั้นที่ 4 เริ่มต้นใช้งานโปรแกรม คลิกขวาที่ไฟล์เดอร์โปรเจกต์ FirstAPP > New > Python File จากนั้นกำหนดชื่อไฟล์ที่จะสร้างลง เป

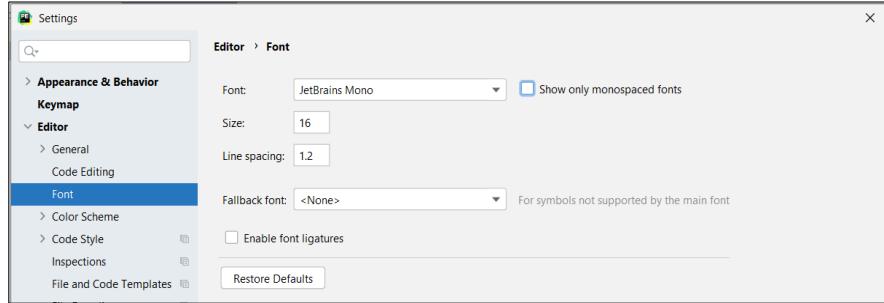


2.3.3 การกำหนดฟอนต์และธีม

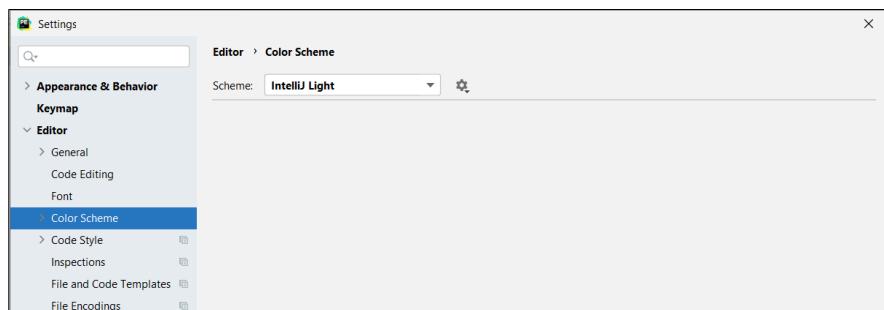
ขั้นที่ 1 เลือกเมนู File > Settings



ขั้นที่ 2 เลือกเมนู Editor > Font เอาเช็คต์ง Show only monospaced fonts ออกร



ขั้นที่ 3 เราสามารถเปลี่ยนธีม(Theme) เลือกเมนู Editor > Color Theme



3. การติดตั้งและใช้งาน Jupyter Notebook

3.1 แนะนำ Jupyter Notebook

Jupyter Notebook เป็น Open Source Web Application สำหรับพัฒนาโปรแกรมได้หลากหลายภาษาแต่ภาษาหลักๆ ที่นำไปใช้เป็นเครื่องมือพัฒนาโปรแกรม ได้แก่ Python, R, Julia, Scala

Jupyter Notebook คือ หน้าเว็บที่ประกอบด้วย ช่อง ๆ cell เรียงต่อกันลงมา โดยแต่ละ cell สามารถเป็นเนื้อหา static content ต่าง ๆ เช่น ข้อความ รูปภาพ กราฟ วิดีโอ เสียง หรือ เป็นโค้ดโปรแกรมคอมพิวเตอร์ ภาษาไพทอน ที่สามารถรันคำสั่งประมวลผล แสดงผลลัพท์ออกมาได้

3.2 การติดตั้ง Jupyter Notebook ผ่าน Anaconda

- **Anaconda** Anaconda Individual Edition คือ โปรแกรมฟรีที่ง่ายต่อการติดตั้ง ตัวจัดการแพ็คเกจ ตัวจัดการสภาพแวดล้อม และไฟทอน ที่ประกอบด้วย open source packages มากกว่า 1,500+ ชุมชนแลกเปลี่ยนเรียนรู้ Anaconda สำหรับ Windows, macOS, Linux.
- **Anaconda.org** เป็นบริการจัดการแพ็คเกจที่ทำให้ง่ายต่อการค้นหา เข้าถึง จัดเก็บ และแบ่งปันสมุดบันทึกและสภาพแวดล้อมสาธารณะ ตลอดจนแพ็คเกจ conda และ PyPI
- **Conda** คือ ตัวจัดการแพ็คเกจและตัวจัดการสภาพแวดล้อมที่ทรงพลังที่ใช้กับคำสั่งบรรทัดคำสั่งที่ Anaconda Prompt สำหรับ Windows หรือในหน้าต่างเทอร์มินัลสำหรับ macOS หรือ Linux



3.2.1 ติดตั้ง Anaconda สำหรับ Windows

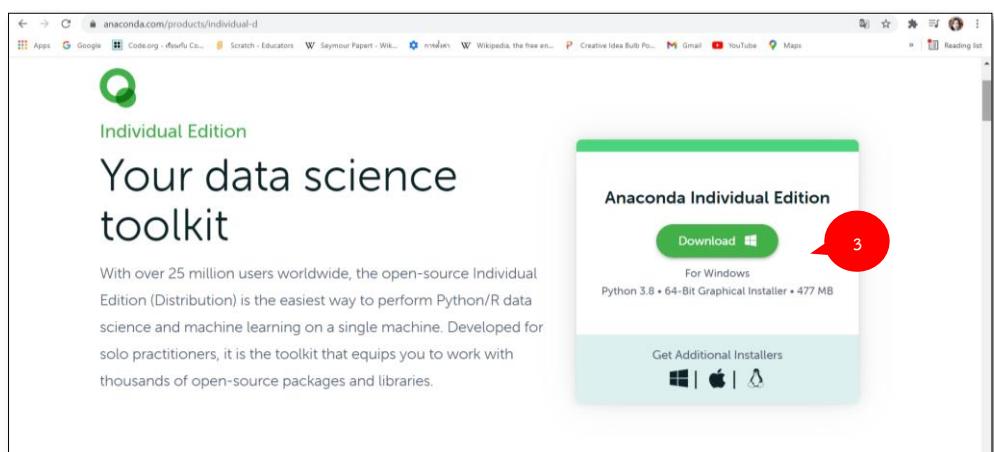
1. ไปที่ Google search พิมพ์ anaconda python



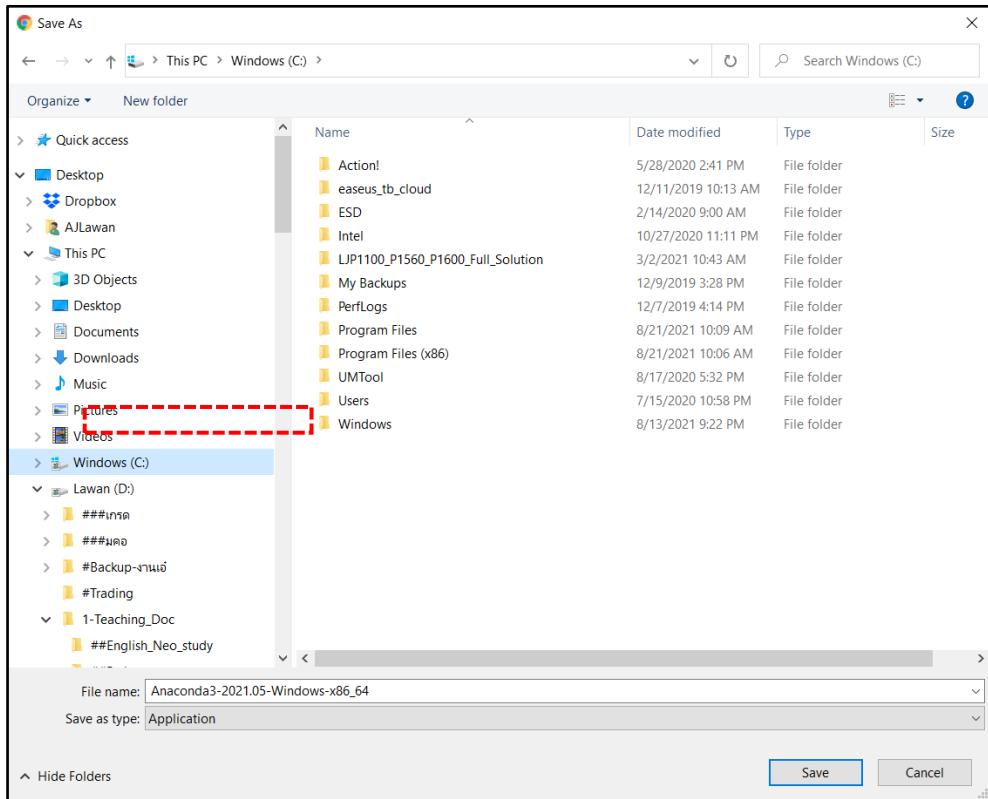
2. คลิก <https://www.anaconda.com/products/individual-d>



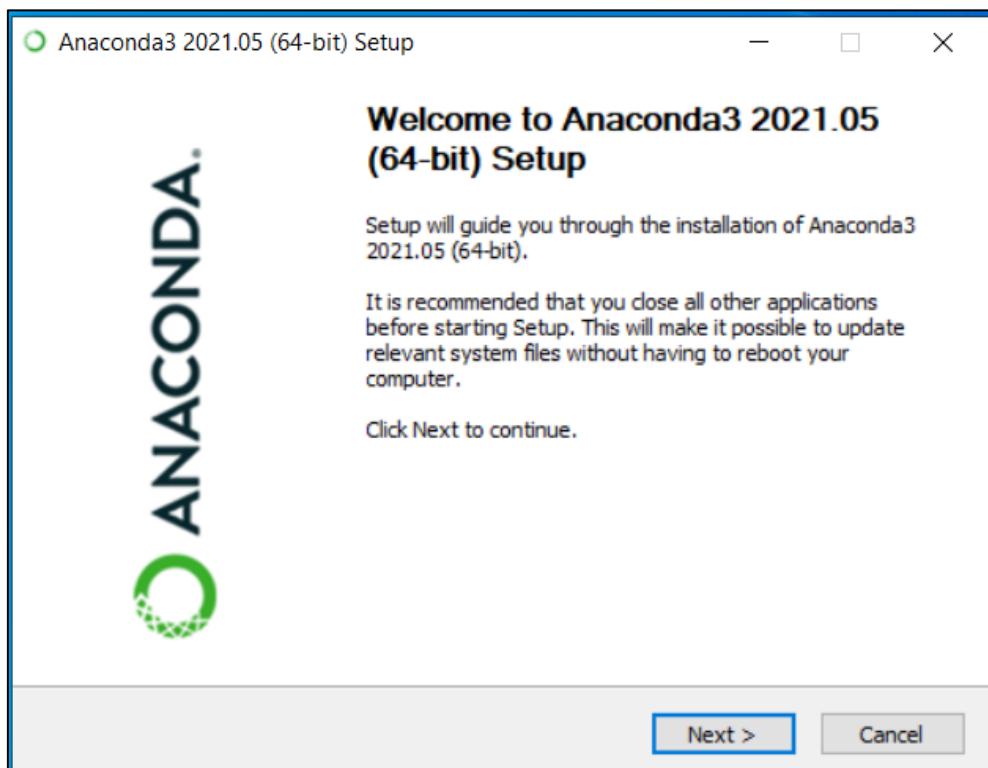
3. คลิก Download



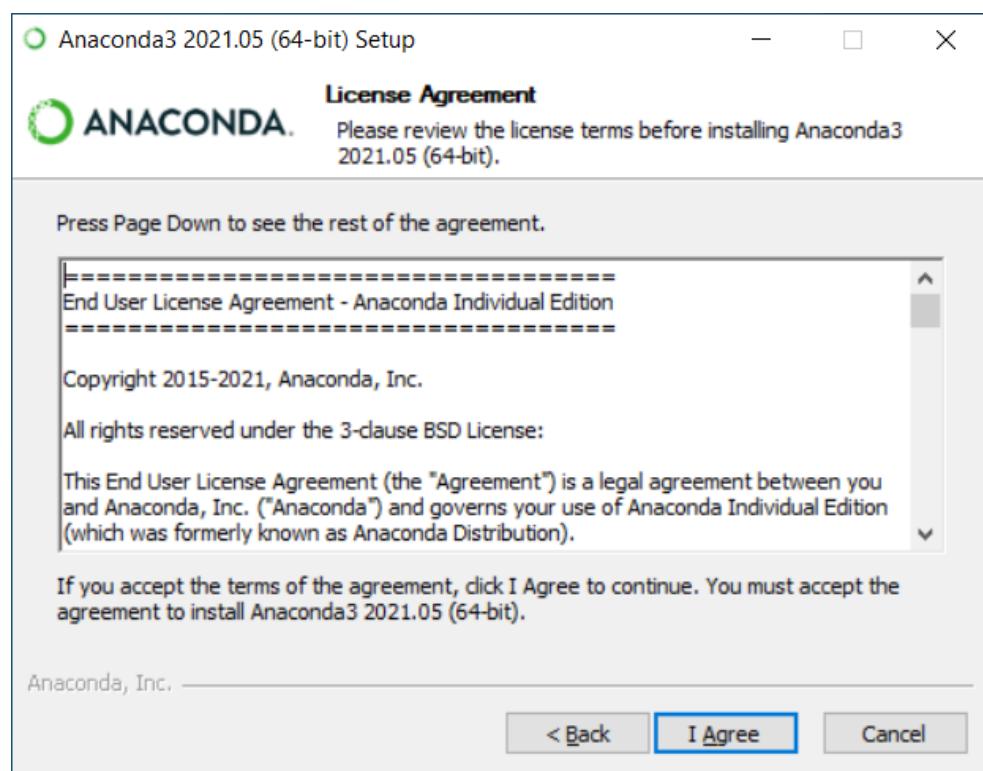
4. Save file โปรแกรมที่ดาวน์โหลดไปยัง Drive ที่ต้องการ



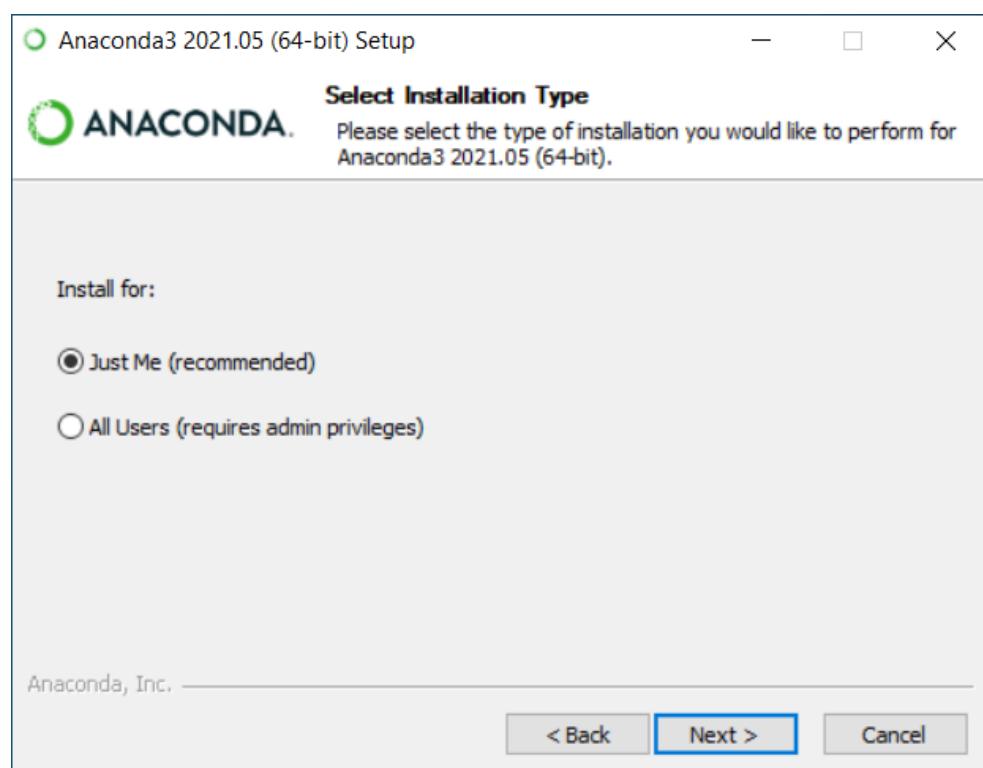
5. คลิก Next



6. คลิก I Agree

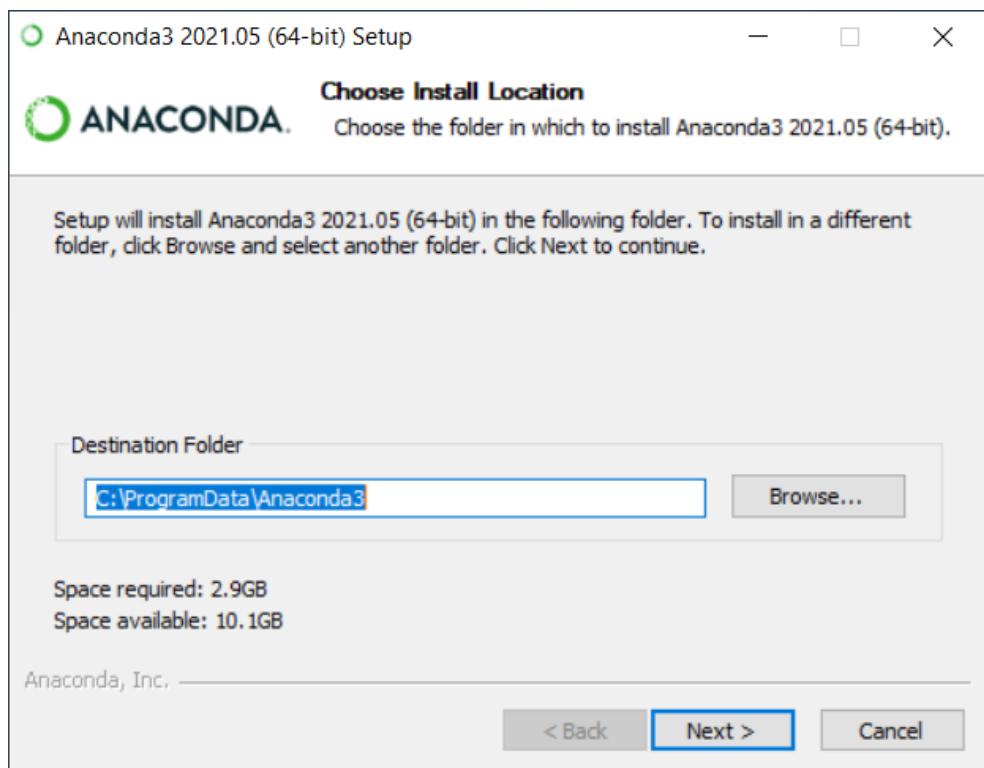


7. เลือก All Users

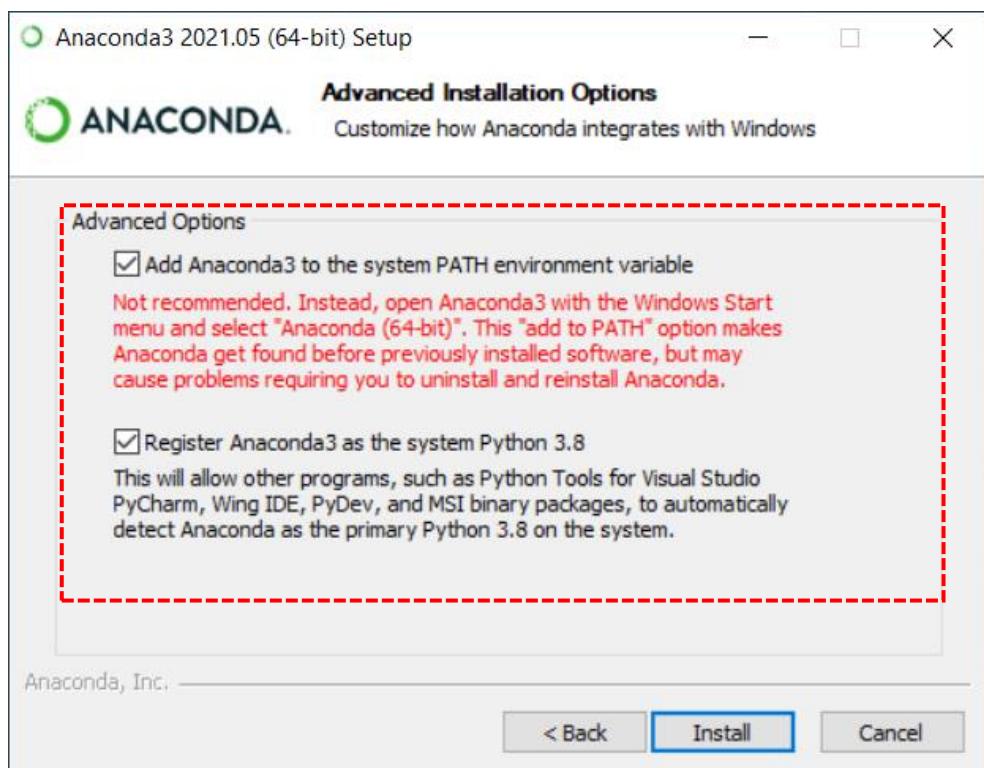




8. เลือกไดรฟ์ที่จะติดตั้งโปรแกรม แล้วคลิก **Next**

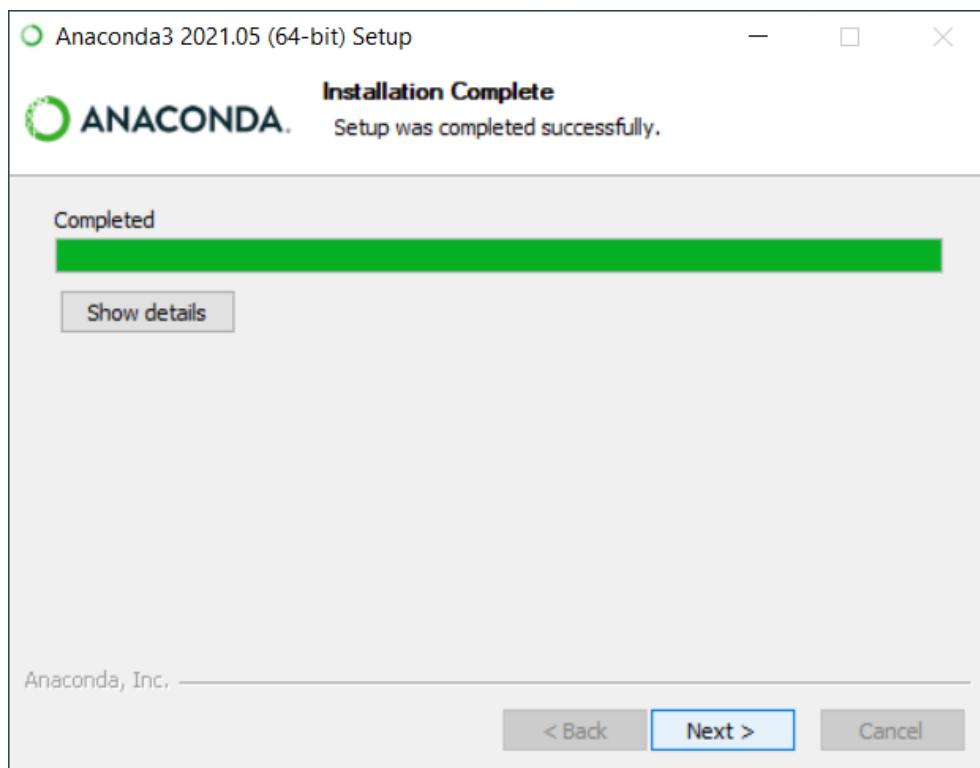


9. เลือก Advanced Options ทั้งหมด คลิก **Install**

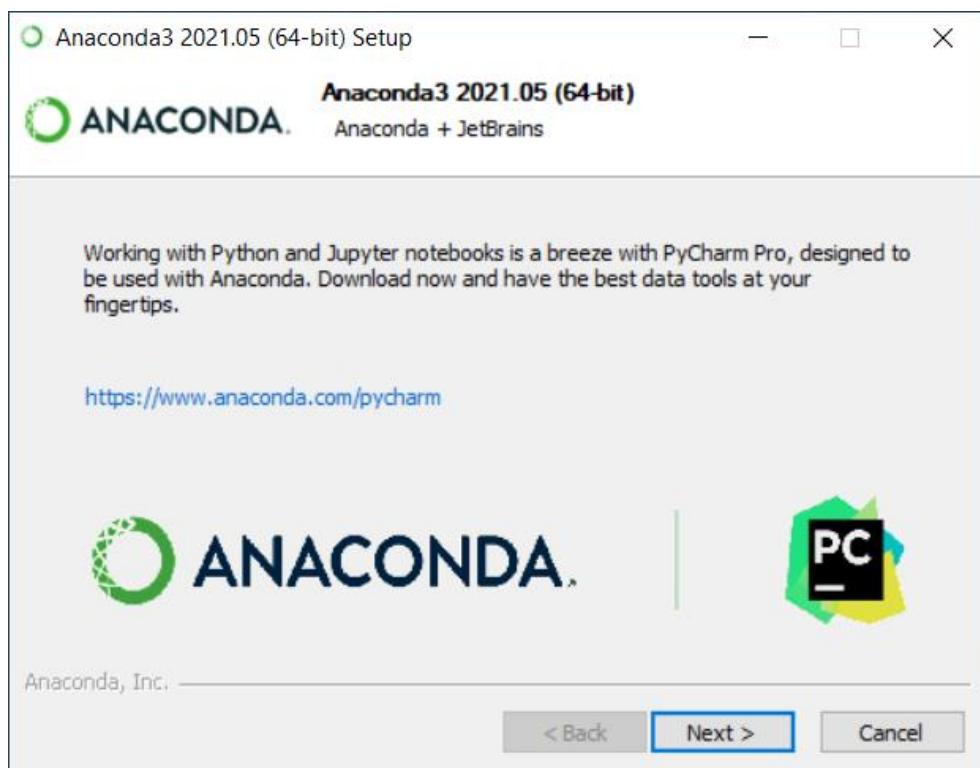




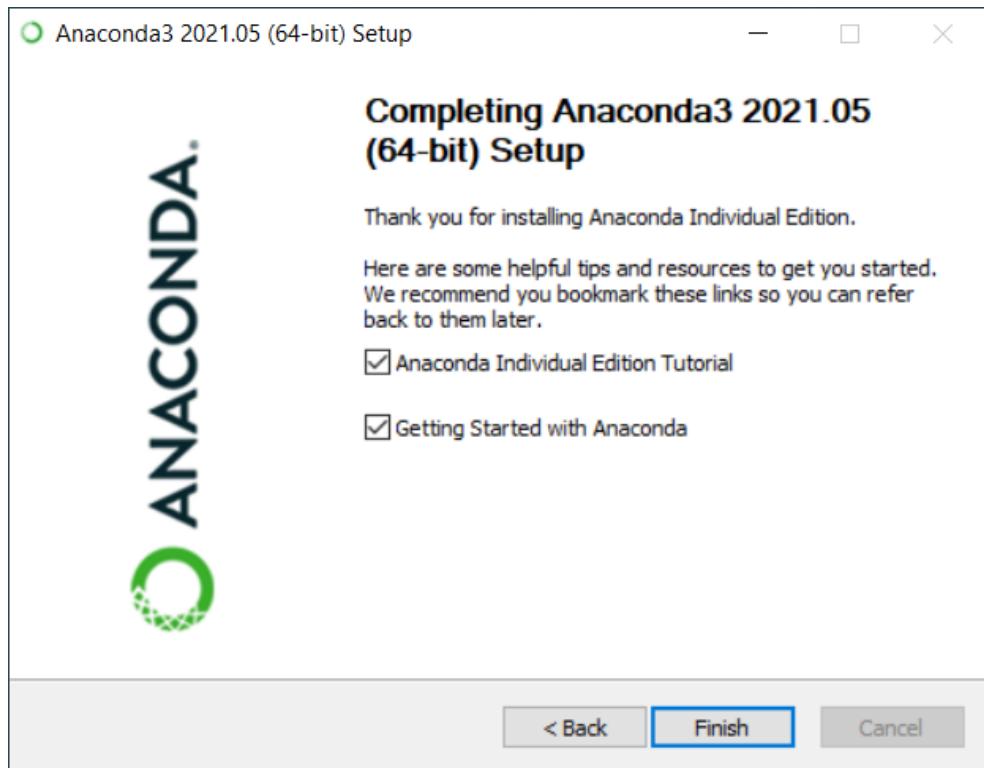
10. คลิก Next



11. คลิก Next



12. คลิก Finish



13. เมื่อการติดตั้งเสร็จสิ้น จากเมนู Start ให้เปิด Anaconda Prompt ทดสอบการติดตั้ง ในหน้าต่าง terminal หรือ Python พิมพ์คำสั่ง conda list รายการแพ็คเกจที่ติดตั้งจะปรากฏขึ้น หากติดตั้งอย่างถูกต้อง

ตัวอย่างที่

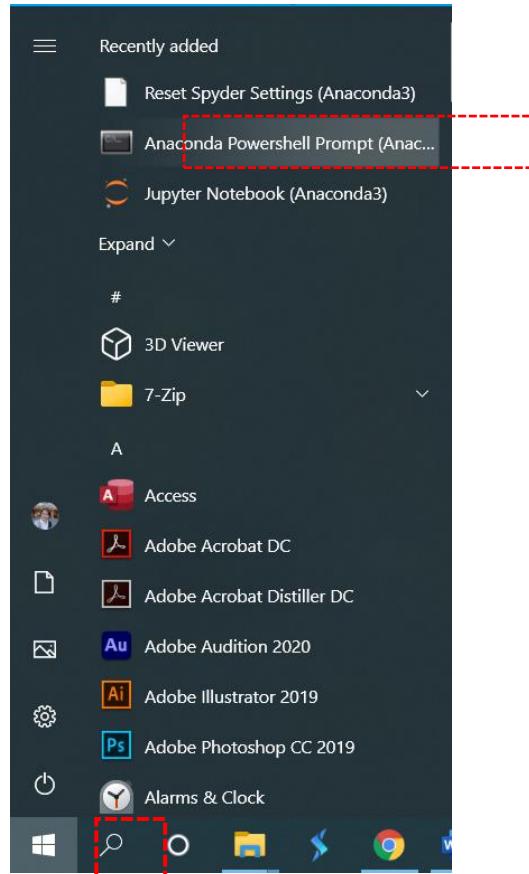
Name	Version	Build	Channel
ipyw_jlab_nb_ext_conf	0.1.0	py38_0	
alabaster	0.7.12	pyhd3eb1b0_0	
anaconda	2021.05	py38_0	
anaconda-client	1.7.2	py38_0	
anaconda-navigator	2.0.3	py38_0	
anaconda-project	0.9.1	pyhd3eb1b0_1	
anyio	2.2.0	py38haa95532_2	
appdirs	1.4.4	py_0	
argh	0.26.2	py38_0	
argon2-cffi	20.1.0	py38h2bbff1b_1	
asn1crypto	1.4.0	py_0	
astroid	2.5	py38haa95532_1	
astropy	4.2.1	py38h2bbff1b_1	
async_generator	1.10	pyhd3eb1b0_0	
atomicwrites	1.4.0	py_0	
attrs	20.3.0	pyhd3eb1b0_0	
autopep8	1.5.6	pyhd3eb1b0_0	
babel	2.9.0	pyhd3eb1b0_0	

ภาพที่ 1 แสดง Anaconda Powershell Prompt



3.2.2 การเริ่มต้น conda

Start Windows > เลือก Anaconda Powershell Prompt (Anaconda3)



ภาพที่ 2 แสดงการเข้าใช้งาน Anaconda Powershell Prompt

3.2.3 การจัดการ conda ตรวจสอบว่าติดตั้ง และ รัน conda บนระบบโดยพิมพ์: `conda -version` แสดงหมายเลขเวอร์ชันที่ติดตั้ง ไม่จำเป็นต้องไปที่เดร็กทอรี Anaconda

```
conda --version
```

ตัวอย่าง: conda 4.7.12

อัปเดต conda เป็นเวอร์ชันปัจจุบัน พิมพ์ต่อไปนี้:

```
conda update conda
```

Conda เปรียบเทียบเวอร์ชันต่างๆ แล้วแสดงสิ่งที่สามารถติดตั้งได้

หากมี conda เวอร์ชันใหม่กว่า ให้พิมพ์ `y` เพื่ออัปเดต:

```
Proceed ([y]/n)? y
```

 หากได้รับข้อความแสดงข้อผิดพลาด ตรวจสอบให้แน่ใจว่าได้ปิดและเปิดหน้าต่างทอร์มินัลอีกครั้งหลังจากติดตั้ง หรือดำเนินการทันที จากนั้นตรวจสอบว่าลงชื่อเข้าใช้บัญชีผู้ใช้เดียวกันกับที่ใช้ในการติดตั้ง Anaconda หรือ Miniconda

3.2.4 ตัวจัดการสภาพแวดล้อม (Managing environments)

Conda อนุญาตให้สร้างสภาพแวดล้อมที่แยกจากกันซึ่งประกอบด้วยไฟล์ แพ็คเกจ และการพิ่งพาที่จะไม่ตอบกับสภาพแวดล้อมอื่น เมื่อเริ่มใช้ conda จะมีสภาพแวดล้อมเริ่มต้นที่ชื่อ **base**. ไม่ต้องการให้โปรแกรมอยู่ในสภาพแวดล้อมพื้นฐาน สร้างสภาพแวดล้อมแยกกันเพื่อให้โปรแกรมแยกกัน

1) สร้างสภาพแวดล้อมใหม่ และติดตั้งแพ็คเกจในนั้น เราจะตั้งชื่อสภาพแวดล้อม **snowflakes** และติดตั้งแพ็คเกจ BioPython ที่พร้อม Anaconda หรือในหน้าต่างเทอร์มินัล ให้พิมพ์ดังต่อไปนี้:

```
conda create --name snowflakes biopython
```

Conda ตรวจสอบเพื่อดูว่าจำเป็นต้องใช้แพ็คเกจเพิ่มเติม ("dependencies") BioPython ได้บ้าง และถามว่าต้องการดำเนินการต่อหรือไม่:

```
Proceed ([y]/n)? y
```

พิมพ์ "y" และกด Enter เพื่อดำเนินการต่อ

2) เมื่อต้องการใช้ หรือ "เปิดใช้งาน" สภาพแวดล้อมใหม่ ให้พิมพ์ดังต่อไปนี้:

- Windows: **conda activate snowflakes**
- macOS and Linux: **conda activate snowflakes**

เมื่อยูใน snowflakes สภาพแวดล้อมแล้ว คำสั่ง conda ใดๆ ที่พิมพ์จะไปที่สภาพแวดล้อมนั้นกว่าจะปิดใช้งาน

3) หากต้องการดูรายการสภาพแวดล้อมทั้งหมด ให้พิมพ์:

```
conda info --envs
```

conda environments:

base	/home/username/Anaconda3
snowflakes	* /home/username/Anaconda3/envs/snowflakes



Note: สภาพแวดล้อมที่ใช้งาน คือ สภาพแวดล้อมที่มีเครื่องหมายดอกจัน (*)

4) เปลี่ยนสภาพแวดล้อมเดิมกลับให้เป็นค่าเริ่มต้น(default) (base): **conda activate** เมื่อสภาพแวดล้อมถูกปิดใช้งาน ชื่อของสภาพแวดล้อมจะไม่แสดงในพร้อมอีกต่อไป และเครื่องหมายดอกจัน (*) จะกลับสู่ base หากต้องการตรวจสอบ สามารถทำข้ามคำสั่งได้ **conda info --envs**



3.2.5 การจัดการไฟฟอน

เมื่อสร้างสภาพแวดล้อมใหม่ conda จะติดตั้งไฟฟอน เวอร์ชันเดียวกับที่ใช้มีอยู่ดาวน์โหลดและติดตั้ง Anaconda หากต้องการใช้ไฟฟอน เวอร์ชันอื่น เช่น Python 3.8 ให้สร้างสภาพแวดล้อมใหม่และระบุเวอร์ชันของไฟฟอน ที่ต้องการ

```
conda info --envs
```

```
python --version
```

3.2.6 การจัดการแพ็คเกจ

ในส่วนนี้ จะตรวจสอบว่าได้ติดตั้งแพ็คเกจใด ตรวจสอบว่ามีแพ็คเกจใดบ้าง และค้นหาแพ็คเกจเฉพาะ และติดตั้ง

- หากต้องการค้นหาแพ็คเกจที่ติดตั้งไว้แล้ว ให้เปิดใช้งานสภาพแวดล้อมที่ต้องการค้นหา ก่อน ดูข้างต้นสำหรับคำสั่งที่จะ เปิดใช้งานสภาพแวดล้อมไฟฟอน
- ตรวจสอบเพื่อดูว่ามีแพ็คเกจที่ยังไม่ได้ติดตั้งซึ่ง "beautifulsoup4" จากที่เก็บ Anaconda หรือไม่ (ต้องเชื่อมต่อกับอินเทอร์เน็ต):

```
conda search beautifulsoup4
```

conda แสดงรายการแพ็คเกจทั้งหมดที่มีชื่อนั้นบนที่เก็บ Anaconda ตั้งนั้นเราจะรู้ว่ามีให้ใช้งาน

- ติดตั้งแพ็คเกจนี้ในสภาพแวดล้อมปัจจุบัน:

```
conda install beautifulsoup4
```

ตรวจสอบเพื่อดูว่าโปรแกรมที่ติดตั้งใหม่อยู่ในสภาพแวดล้อมนี้หรือไม่:

```
conda list
```

คึกษาข้อมูลเพิ่มเติม

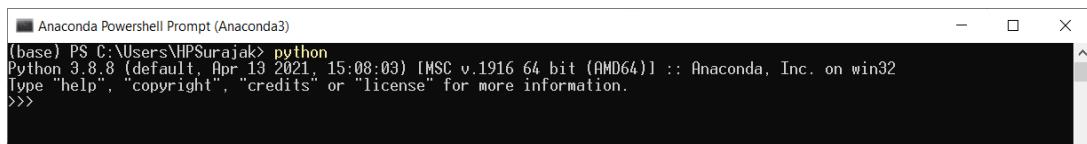
- Conda cheat sheet
- Full documentation--- <https://conda.io/docs/>
- Free community support--
 - <https://groups.google.com/a/anaconda.com/forum/#!forum/anaconda>
- Paid support options--- <https://www.anaconda.com/support/>

กิจกรรม

1. ให้ติดตั้งและตรวจสอบการติดตั้ง Anaconda เข้าสู่โหมด Prompt



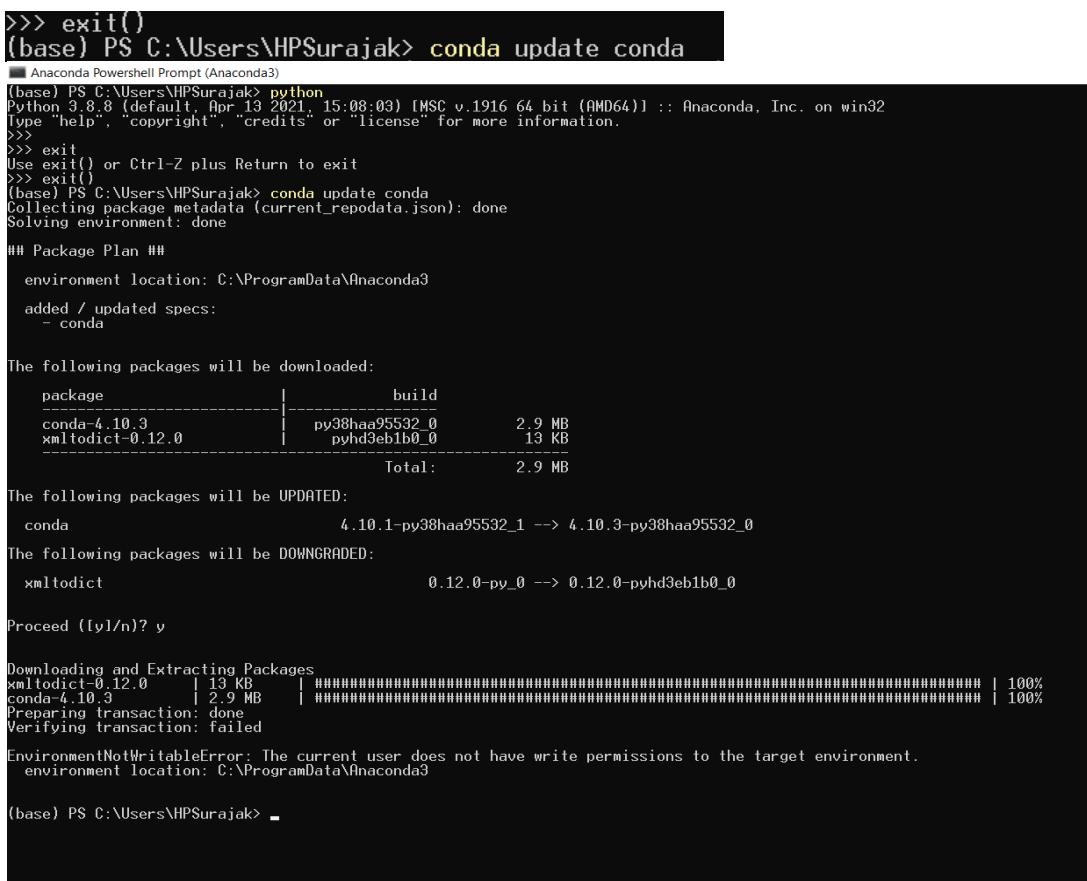
ทดลองพิมพ์คำสั่งไฟทน ถ้าขึ้นดังภาพแสดงว่าพร้อมใช้งาน



```
(base) PS C:\Users\HPSurajak> python
Python 3.8.8 (default, Apr 19 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

พิมพ์คำสั่ง `exit()`

`conda update conda`



```
>>> exit()
(base) PS C:\Users\HPSurajak> conda update conda
(base) PS C:\Users\HPSurajak> python
Python 3.8.8 (default, Apr 19 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> exit
Use exit() or Ctrl-Z plus Return to exit
>>> exit
(base) PS C:\Users\HPSurajak> conda update conda
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

environment location: C:\ProgramData\Anaconda3
added / updated specs:
- conda

The following packages will be downloaded:
  package          |           build
  conda-4.10.3    | py38haa95532_0
  xmltodict-0.12.0| pyhd3eb1b0_0
                                              Total:   2.9 MB

The following packages will be UPDATED:
  conda            4.10.1-py38haa95532_1 --> 4.10.3-py38haa95532_0
  The following packages will be DOWNGRADED:
    xmltodict        0.12.0-py_0 --> 0.12.0-pyhd3eb1b0_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
xmltodict-0.12.0 | 13 KB      | ################################### | 100%
conda-4.10.3     | 2.9 MB     | ################################### | 100%
Preparing transaction: done
Verifying transaction: failed
EnvironmentNotWritableError: The current user does not have write permissions to the target environment.
environment location: C:\ProgramData\Anaconda3

(base) PS C:\Users\HPSurajak> -
```

Conda update anaconda-navigator



```
(base) PS C:\Users\HPSurajak> conda update anaconda-navigator
```

Conda update anaconda-updater



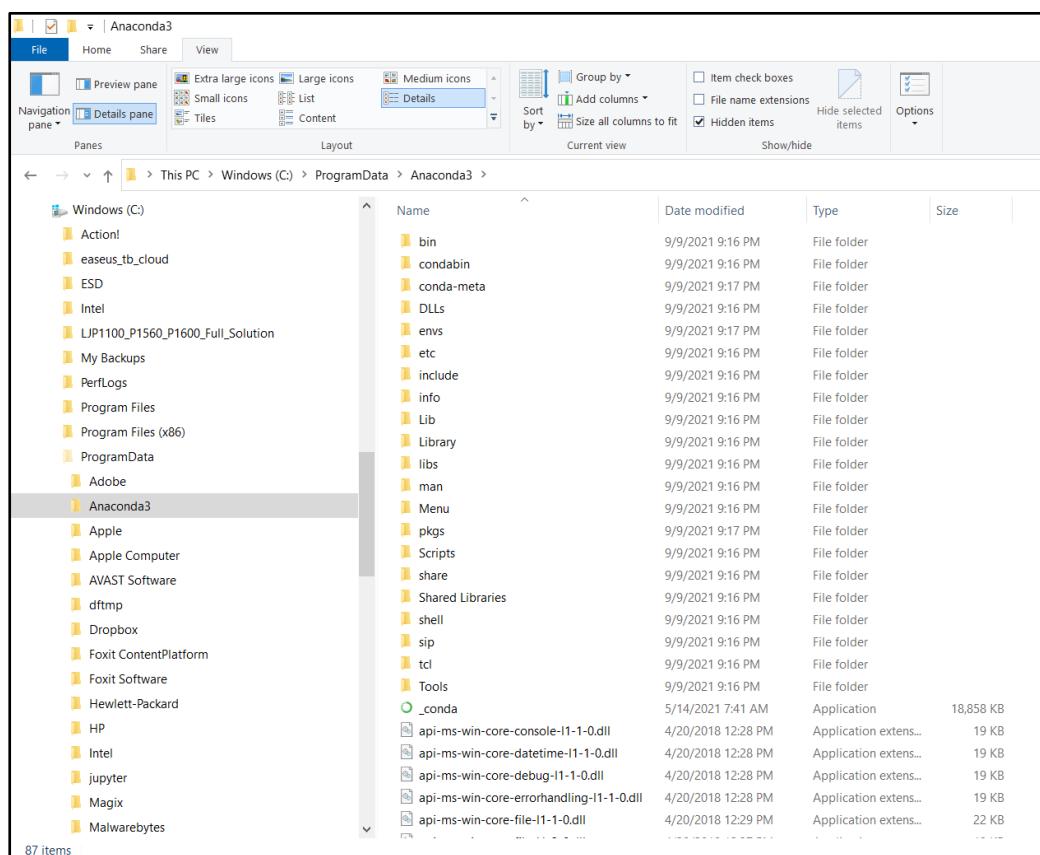
```
(base) PS C:\Users\HPSurajak> conda update navigator-updater
```

ตรวจสอบ Version ของ anaconda , python และ environment



```
(base) PS C:\Users\HPSurajak> conda -V
conda 4.10.1
(base) PS C:\Users\HPSurajak> python -V
Python 3.8.8
(base) PS C:\Users\HPSurajak> conda info --envs
# conda environments:
#
base           *  C:\ProgramData\Anaconda3
(base) PS C:\Users\HPSurajak>
```

ตรวจสอบไฟล์ anaconda3 หากไม่พบไฟล์ ให้แสดงไฟล์ที่ซ่อนอยู่ (Hidden Items)

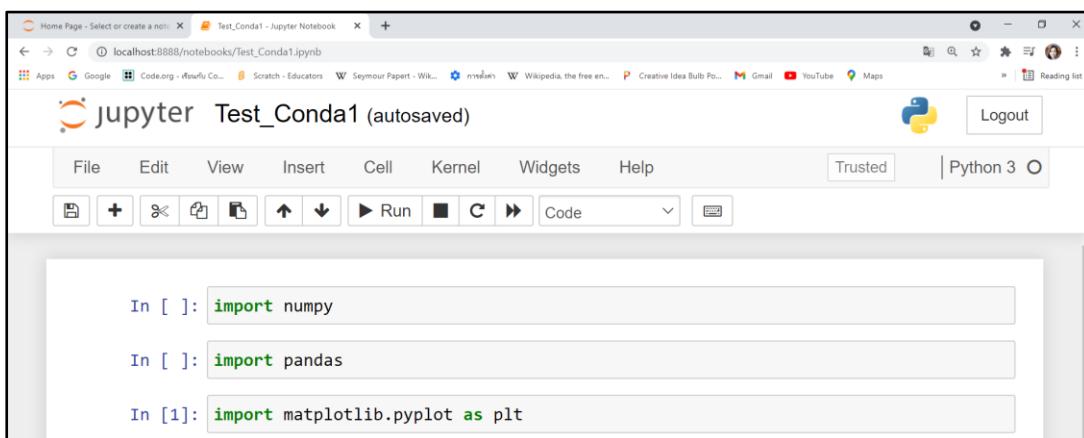
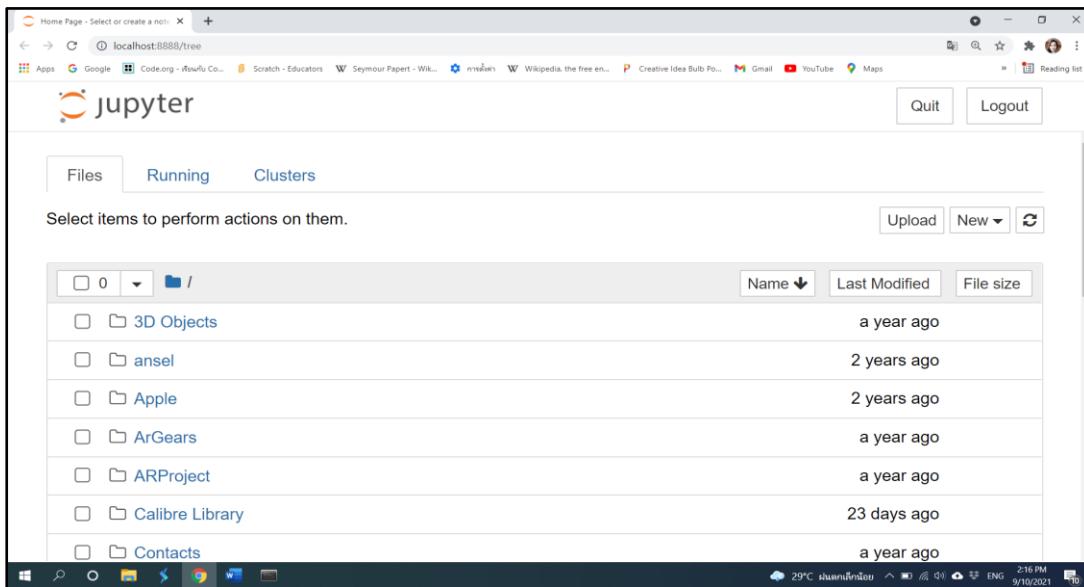


ภาพที่ 3 แสดงตำแหน่งไฟล์ Anaconda

3.3 เริ่มต้นทำงาน Jupyter Notebook

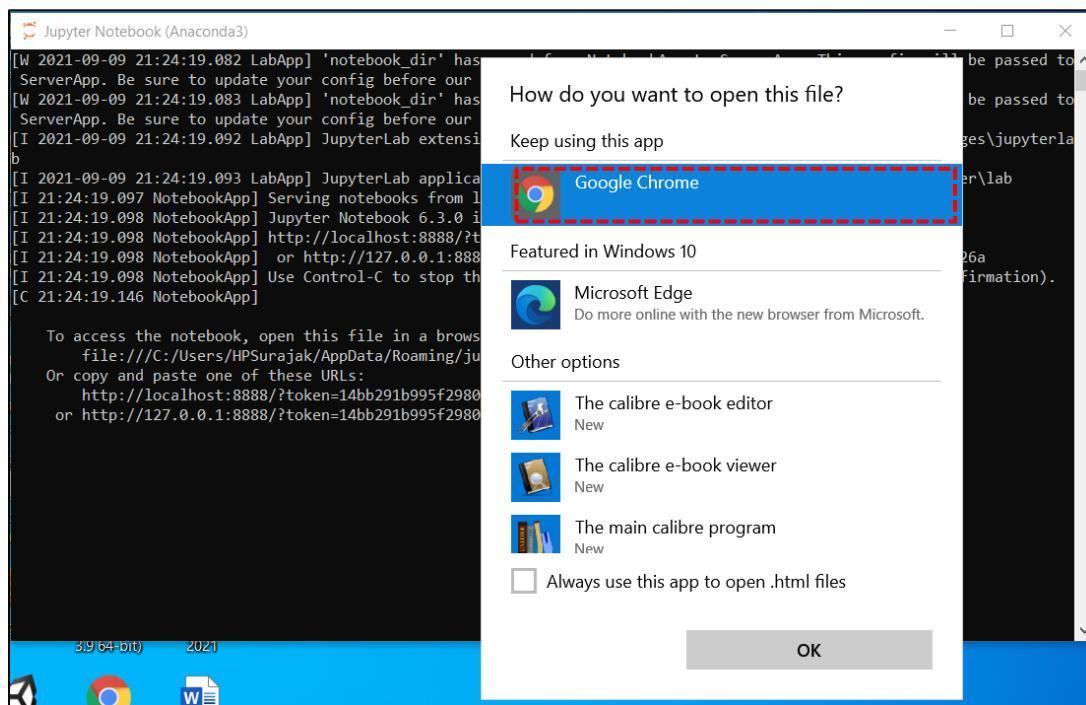
3.3.1 การเปิดใช้งาน

1) Start Windows เลือก Anaconda Powershell Prompt (Anaconda3)

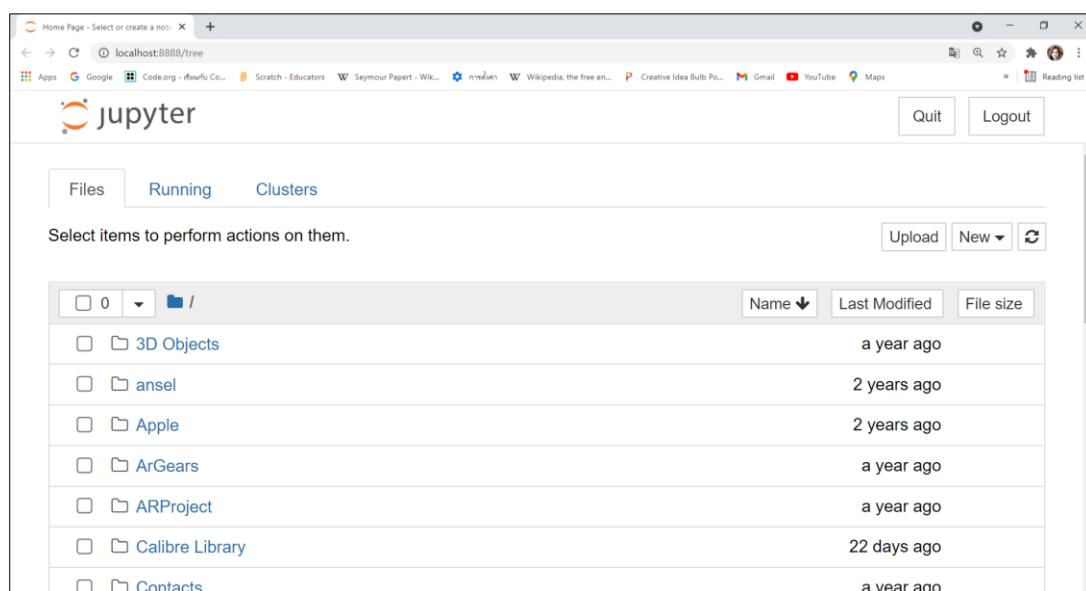




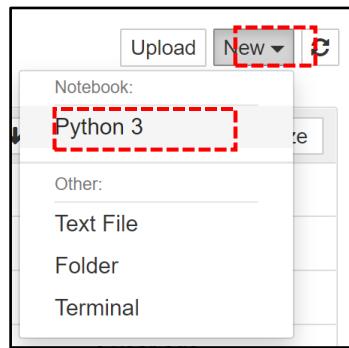
2) รัน Google Chrome > OK



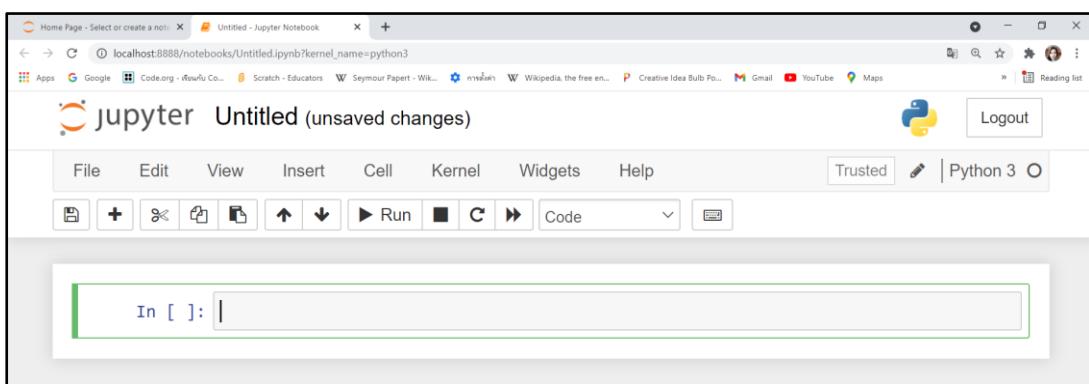
3) โปรแกรม jupyter Notebook จะเปิดบนเบราว์เซอร์ Google Chrome



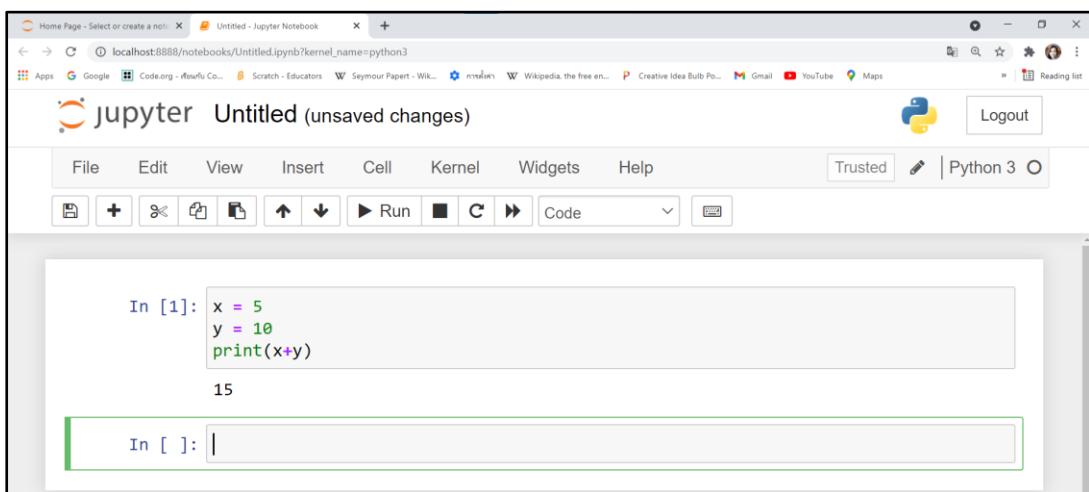
4) New > Python 3



5) เปิดหน้าจอร์มเขียนโปรแกรม



6) ทดลองเขียนโปรแกรม และแสดงผลลัพธ์

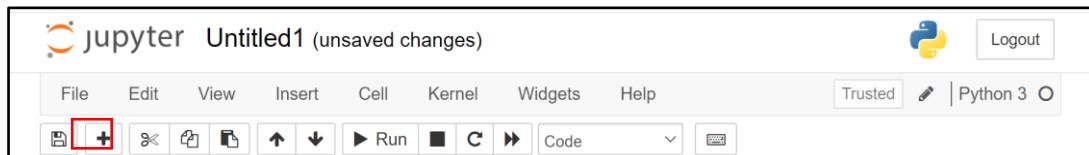




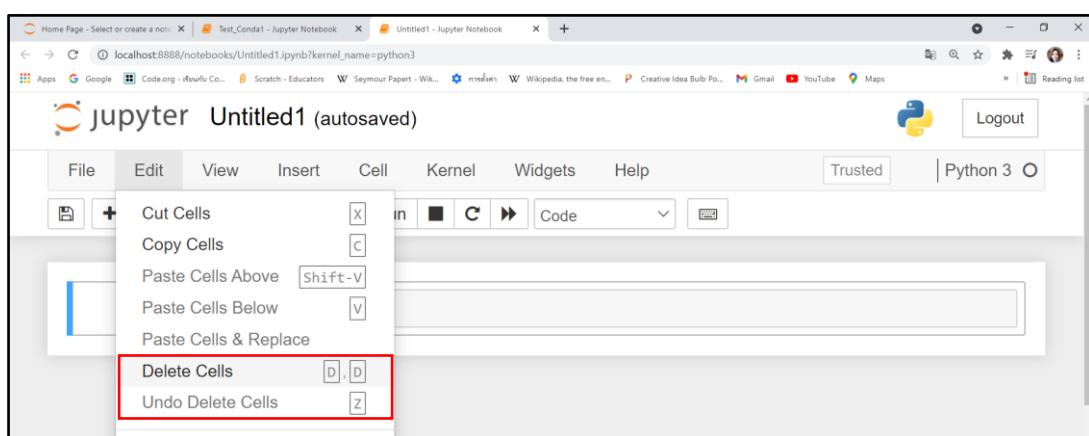
3.3.2 เริ่มต้นทำงาน Jupyter Notebook

1) คลิกเลือก cell ในการทำงานกับ cell ให้คลิกบริเวณที่ต้องการ cell จะถูกเลือกโดยมีสัญลักษณ์เป็นการตีกรอบ

2) การสร้าง cell ใหม่ เลือก cell กดปุ่ม + ในแถบเครื่องมือเพื่อสร้าง cell ด้านล่าง cell ที่กำลังเลือกอยู่

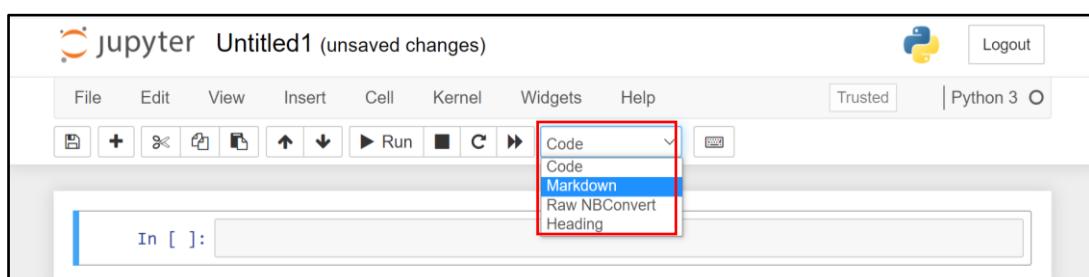


3) การลบ cell เลือกที่ cell ที่จะลบ แล้วเลือกเครื่องมือบนแถบเครื่องมือ Edit > Delete Cells



4) code cell - cell ที่ใช้เขียนไพทอน เมื่อสร้าง cell ขึ้นมา cell ที่จะได้คือ code cell ที่ใช้เขียนไพทอน

5) เปลี่ยนชนิด cell เมื่อเปลี่ยนประเภทเป็น markdown cell จะได้ cell ที่ใช้จดบันทึก

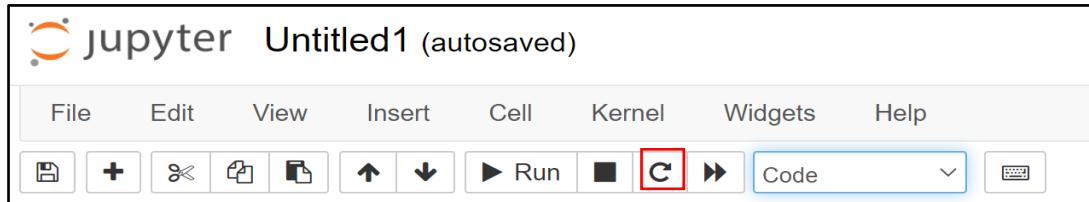


6) run cell - สั่งให้ประมวลผล เมื่อต้องการให้ cell ประมวลผล ให้ run cell นั้น



7) restart - สั่งปิดใหม่

เมื่อ Jupyter Notebook ไม่มีการตอบสนองสามารถ restart และลบผลการคำนวณก่อนหน้า เพื่อเริ่มใช้งานใหม่อีกรอบ



8) แก้ไขและ荷ดคำสั่ง หากเซลล์ทำงานอยู่สอง荷ดที่แตกต่างกัน:

- 荷ดแก้ไข
- 荷ดคำสั่ง

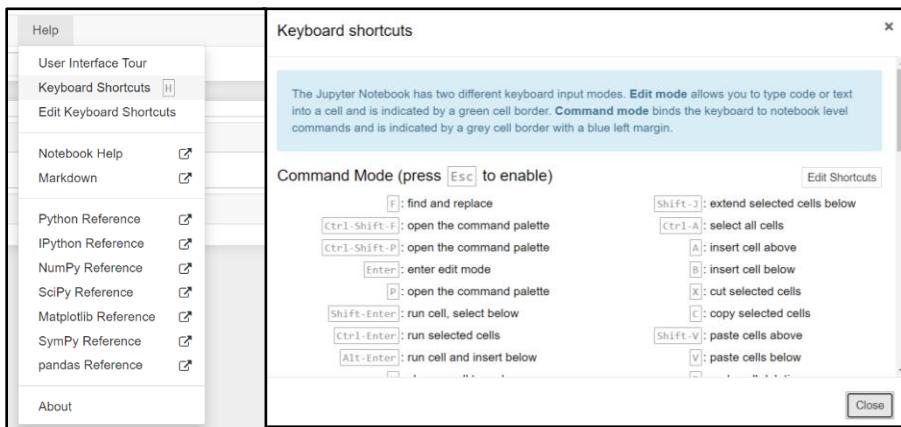
```
In [2]: print ('Hello World')
Hello World
```

荷ดแก้ไขจะเข้าสู่หากคลิกเข้าไปในพื้นที่โค้ดของเซลล์นั้น 荷ดนี้ระบุด้วยขอบสีเขียวทางด้านซ้ายของเซลล์:

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

หากต้องการออกจาก荷ดแก้ไข และกลับสู่荷ดคำสั่งอีกรอบเพียงแค่กด ESC

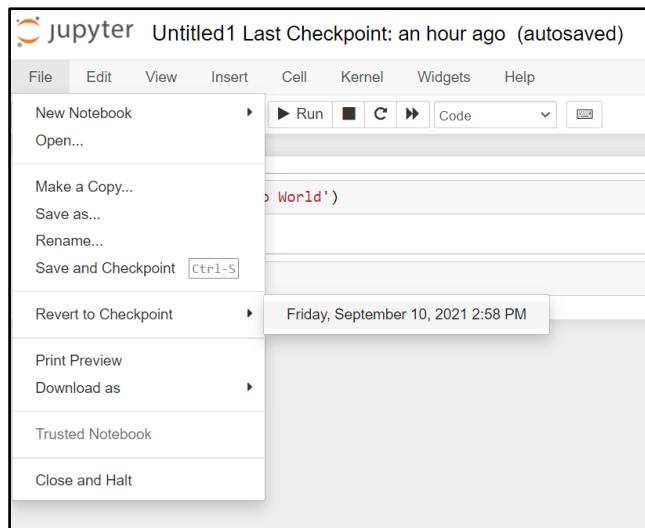
หากต้องการดูภาพรวมของฟังก์ชันที่มีอยู่ในคำสั่ง และใน荷ดแก้ไขสามารถเปิดภาพรวมของแป้นพิมพ์ลัดได้โดยใช้รายการเมนู **Help → Keyboard Shortcut**:



9) จุดตรวจ (checkpoint)

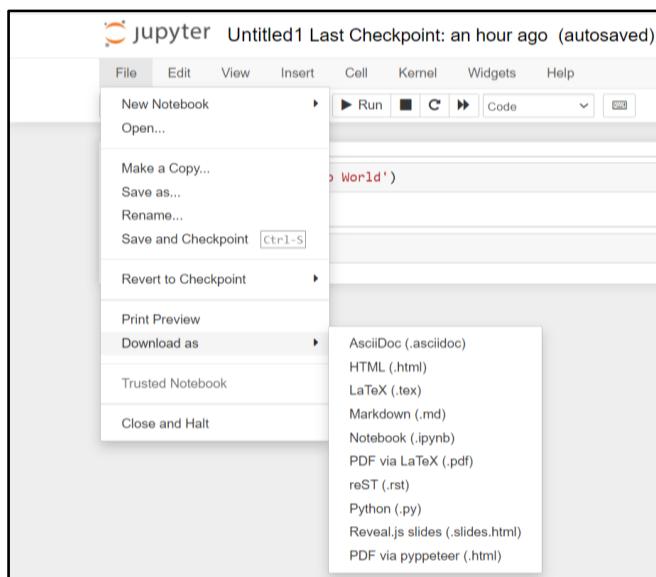
อีกหนึ่งฟังก์ชันเด็ดของ Jupyter Notebook คือ ความสามารถในการสร้างระดับ การสร้างจุดตรวจจะเป็นการจัดเก็บสถานะปัจจุบันของ Notebook เพื่อให้สามารถกลับไปที่จุดตรวจนี้ได้ในภายหลังและยกเลิกการเปลี่ยนแปลงที่เกิดขึ้นกับ Notebook ในระหว่างนี้

ในการสร้างจุดตรวจใหม่สำหรับ Notebook ให้เลือกรายการเมนู **File menu → Revert to Checkpoint**



10) การส่งออกโนํตบุ๊ก (exporting a notebook)

Jupyter Notebook มีตัวเลือกมากมายในการส่งออกสมุดบันทึก ตัวเลือกเหล่านี้สามารถพบได้ในเมนู **File → Download As :**



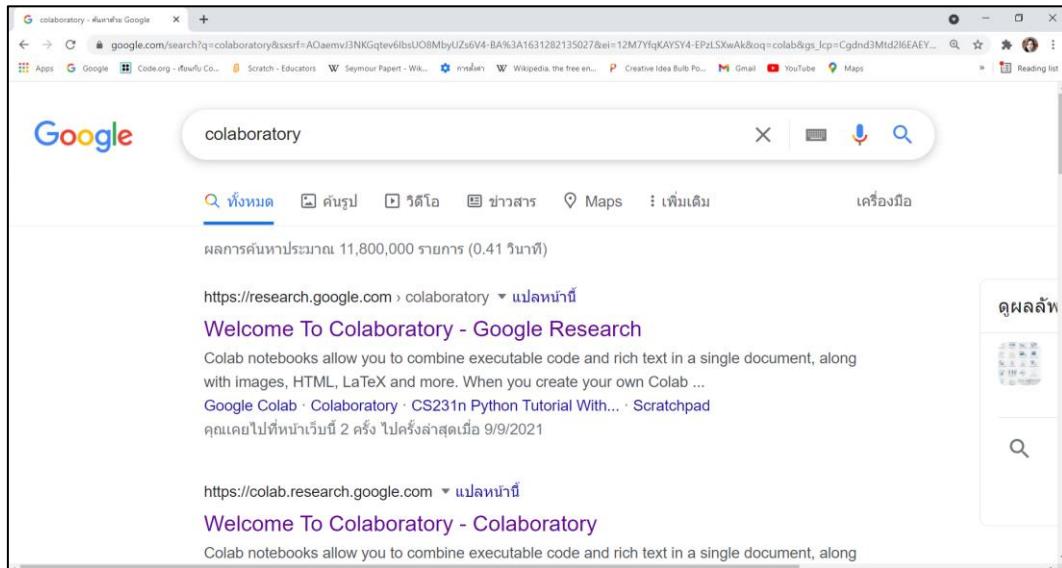
สรุปการเรียนรู้พื้นฐาน บน Jupyter Notebook ทำให้การลองผิดลองถูกง่ายขึ้น เนื่องจากสามารถทดลองรันโค้ดในแต่ละ cell เพื่อเรียนรู้การทำงานของคำสั่งได้ นอกจากที่จะใช้เรียนได้ง่าย Jupyter Notebook ยังนิยมใช้ในการทำงานวิเคราะห์ข้อมูล การจัดการข้อมูลจำนวนมาก อีกด้วย

4. การใช้งาน Google Colab

Colaboratory หรือเรียกสั้นๆ ว่า "Colab" ช่วยให้สามารถเขียนและเรียกใช้เพฟอน ในเบราว์เซอร์ได้ด้วย

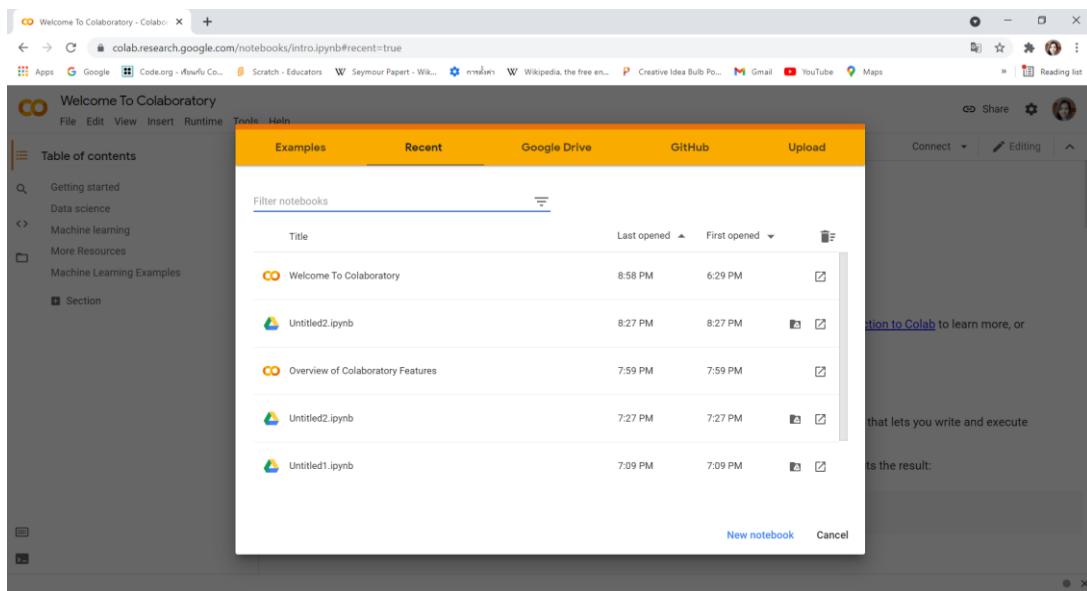
- ต้องมีการกำหนดค่าเป็นศูนย์
- เข้าถึง GPU ฟรี
- แบ่งปันง่าย

1. ไปที่ <https://colab.research.google.com/>



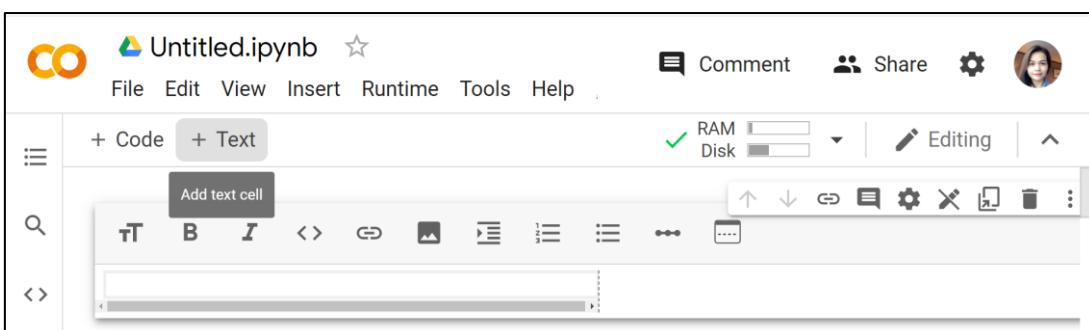
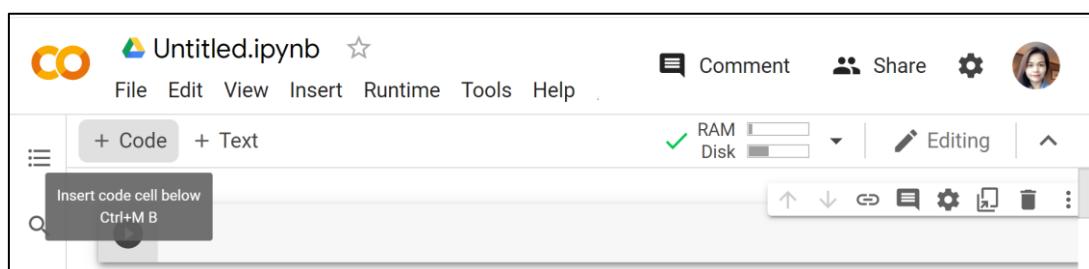
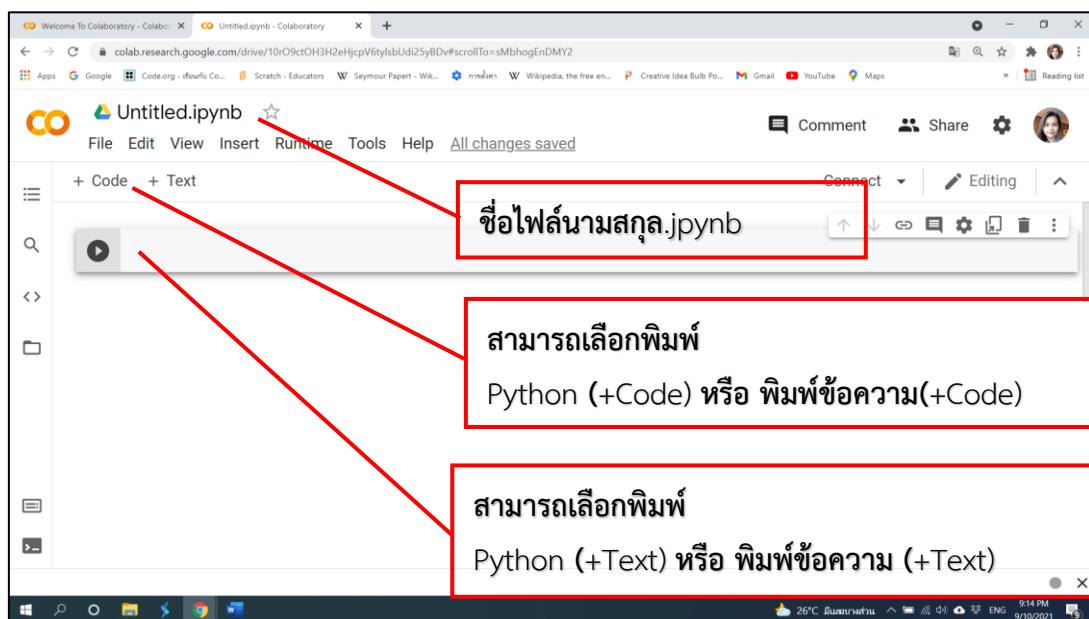
2. กด Sign in โดยใช้ Google Account (Gmail) หากไม่มีให้ทำการสมัครอีเมล

3. เข้าสู่หน้าหลักของ Google Colaboratory

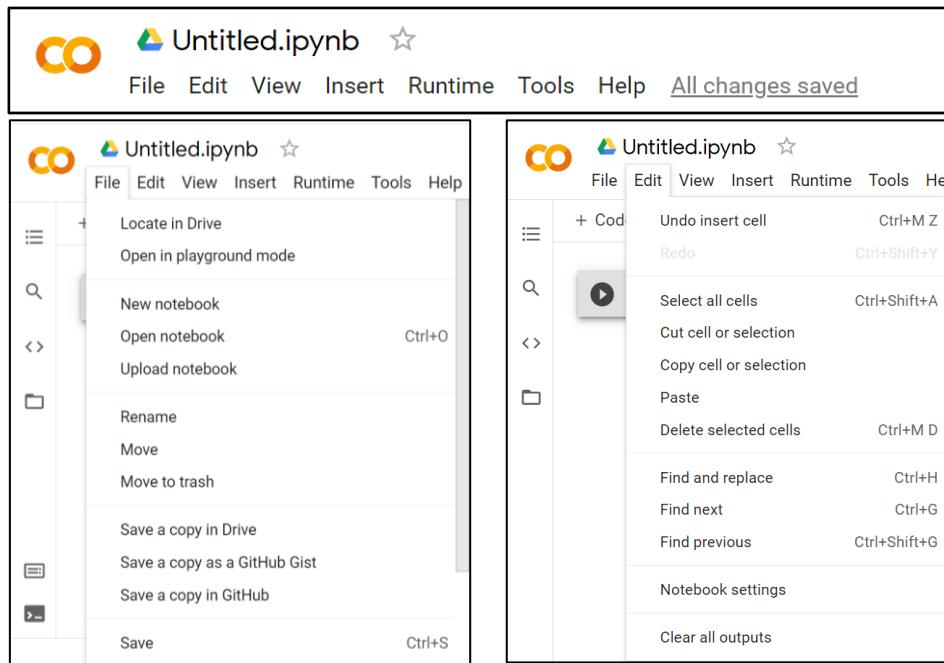




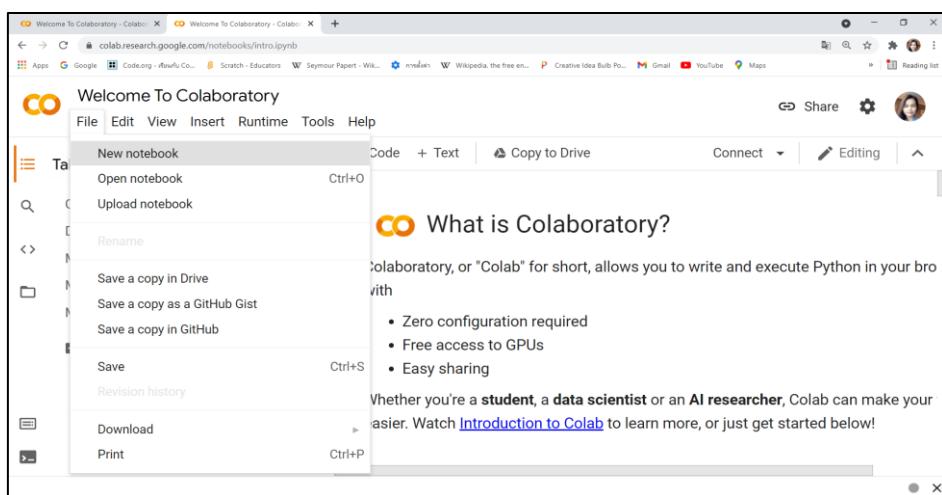
4. พร้อมเขียนคำสั่งใช้งานในภาษาไพทอน



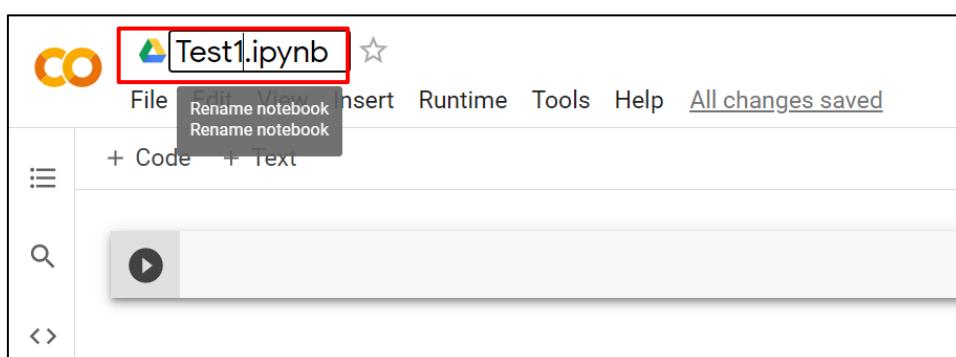
5. เมนูคำสั่ง



6. เลือก File -> New Notebook จากหน้าจัดแสดงเซลล์ในการป้อนคำสั่ง ดังในหน้าต่อไป



7. การตั้งชื่อไฟล์





8. พิมพ์ code (ให้เลือก + Code)

A screenshot of the Google Colab interface. The title bar says "Test1.ipynb". The menu bar includes File, Edit, View, Insert, Runtime, Tools, Help, and "All changes saved". On the left, there are buttons for "+ Code" and "+ Text". The main area shows a code cell with the Python command "print ('Hello Colab')". Below the code cell is a play button icon.

9. แสดงผลลัพธ์ (Run cell)

A screenshot of the Google Colab interface after running the code. The code cell now shows "Hello Colab" with a green checkmark and a play button icon. Below the cell, the output "Hello Colab" is displayed.

10. บันทึกไฟล์ (Save)

A screenshot of the Google Colab interface showing the "File" menu open. The menu options include File, Edit, View, Insert, Runtime, Tools, Help, Locate in Drive, Open in playground mode, New notebook, Open notebook, Upload notebook, Rename, Move, Move to trash, Save a copy in Drive (which is highlighted with a red box), Save a copy as a GitHub Gist, Save a copy in GitHub, and Save.

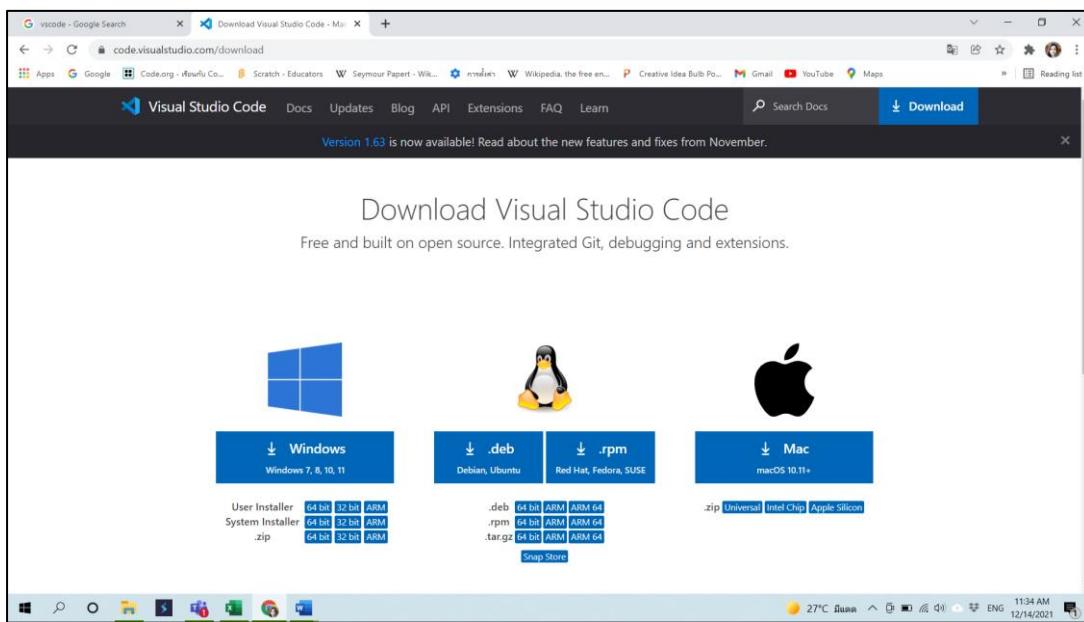
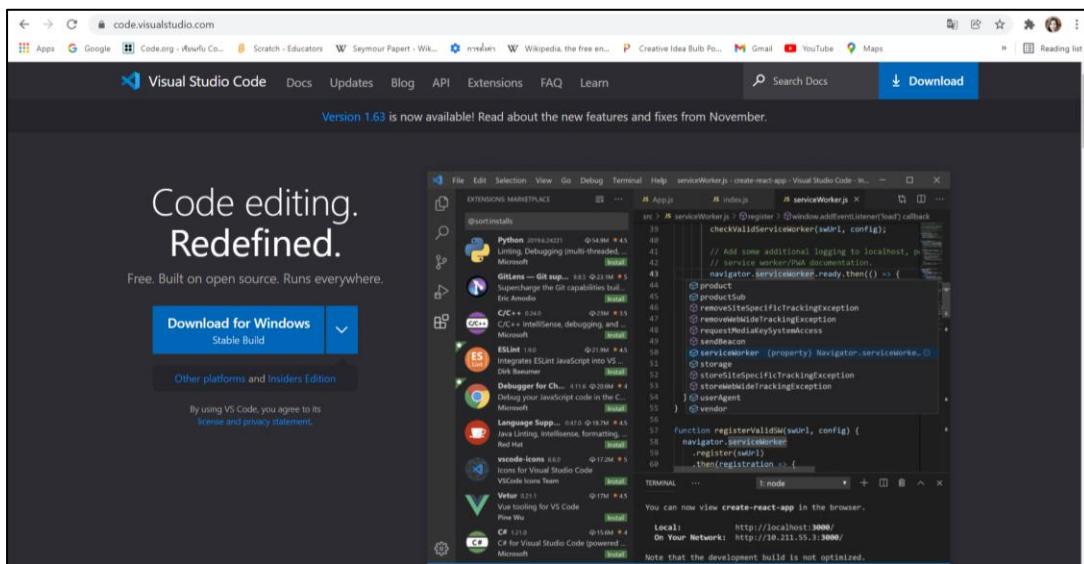
11. เปิดไฟล์งานที่เคยบันทึกไว้แล้ว

11.1 เข้าไปที่ google colab อีกครั้ง

11.2 คลิกเลือกไฟล์ที่เคยบันทึกไว้แล้ว

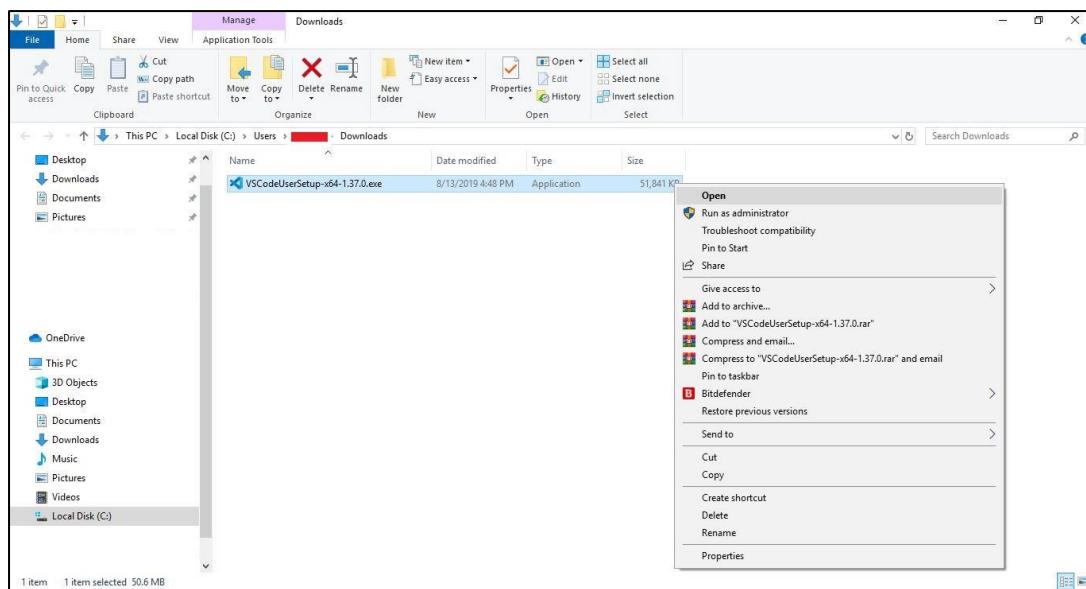
5. การติดตั้งโปรแกรม Visual Studio Code หรือ VS Code

1. เข้าไปที่เว็บไซต์ <https://code.visualstudio.com/> และ Download โปรแกรม VS Code โดยเลือกให้ตรงกับ OS ของเครื่องคอมพิวเตอร์

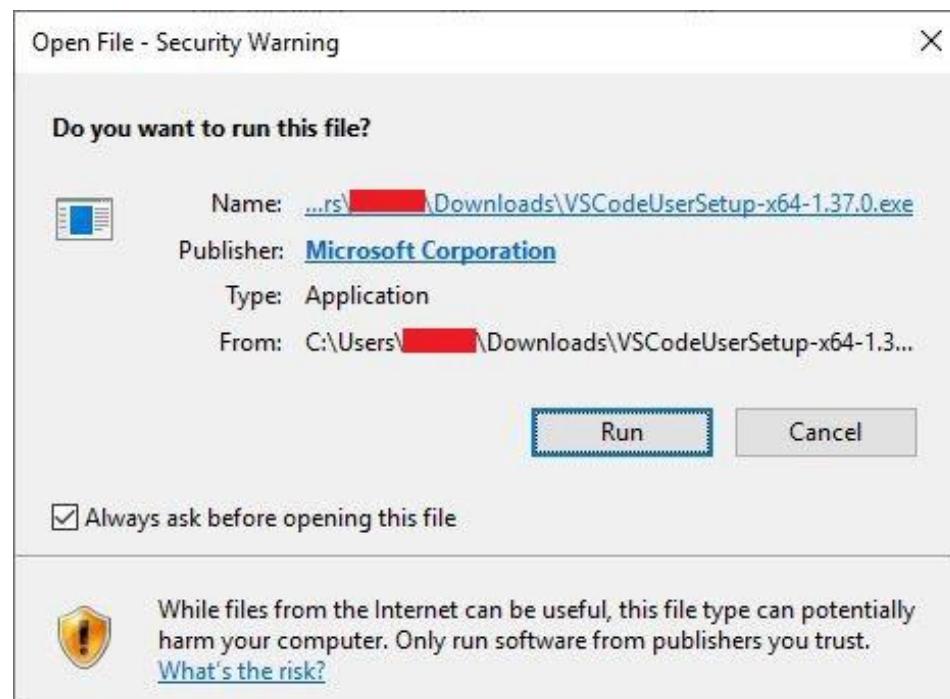




2. ดับเบิลคลิก หรือคิกขวาและกด “Open” โปรแกรมที่ดาวน์โหลดมา

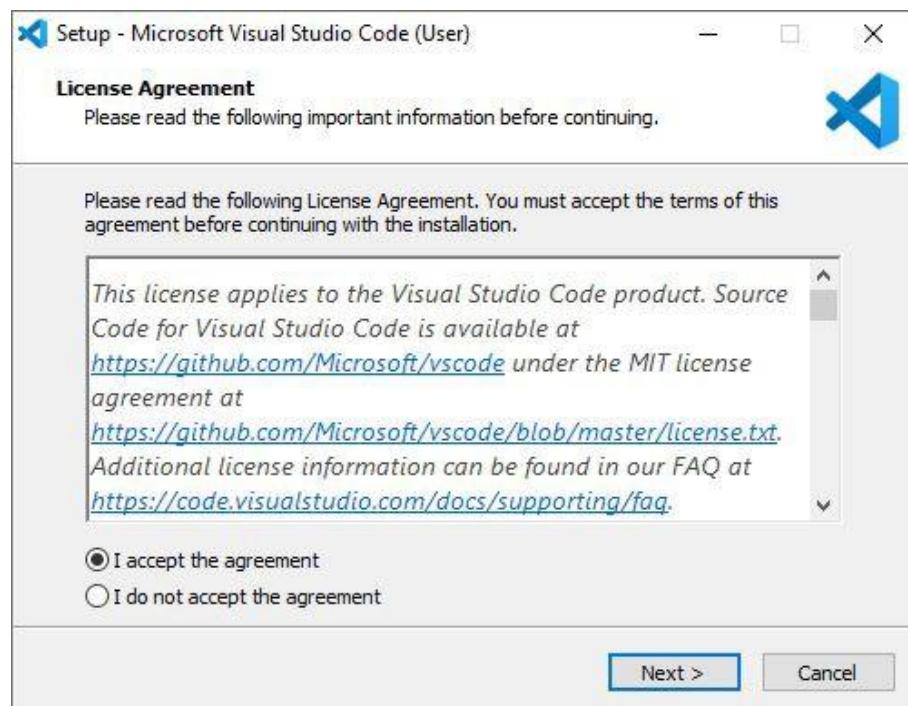


3. คลิกปุ่ม “Run”

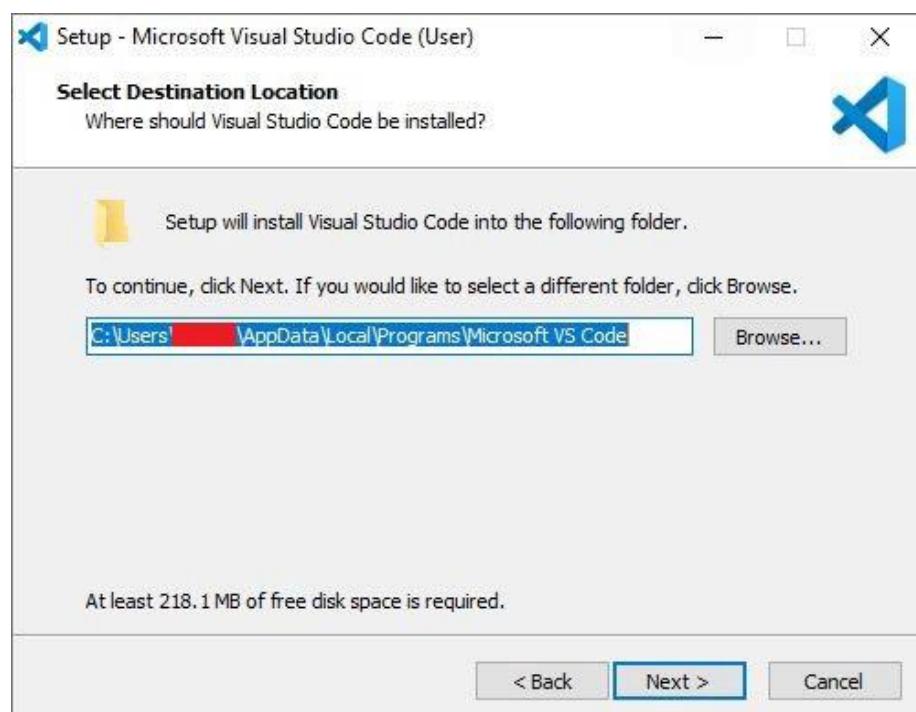




4. เลือก “I accept the agreement” และคลิกปุ่ม “Next”

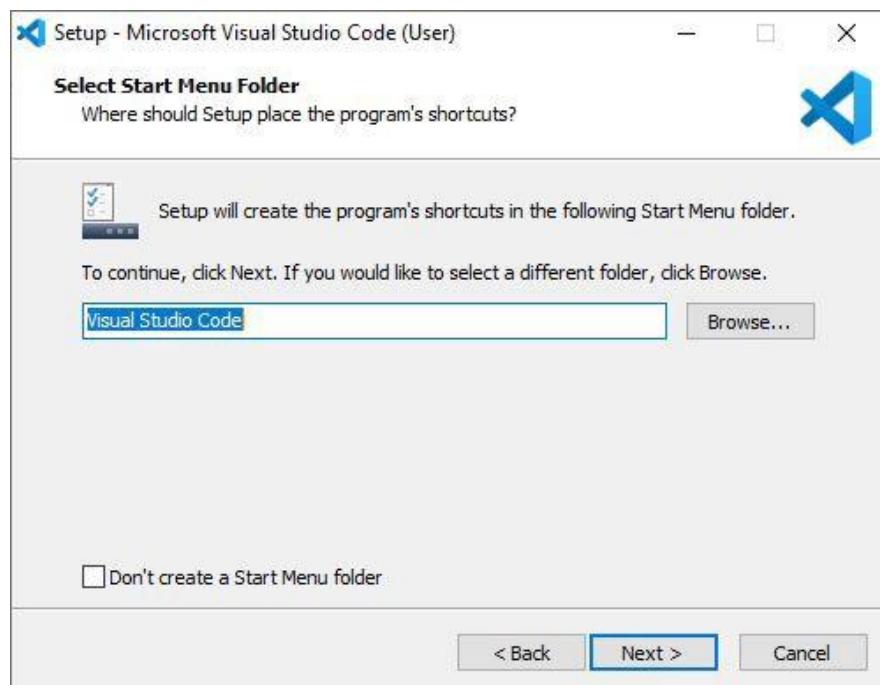


5. เลือกพื้นที่ในการจัดเก็บโปรแกรม (แนะนำให้ใช้ Default ที่ให้มา) และคลิกปุ่ม “Next”

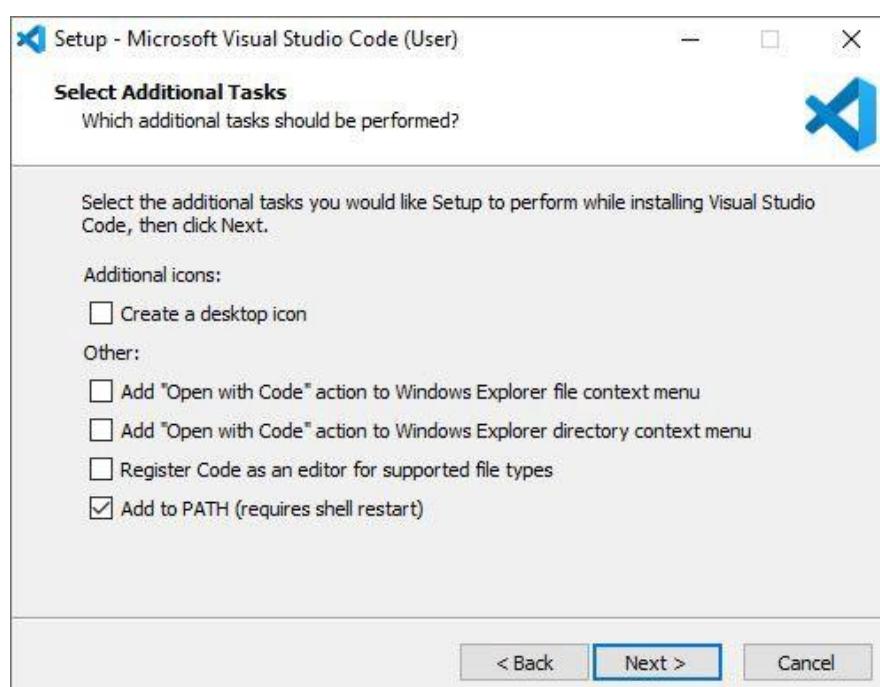




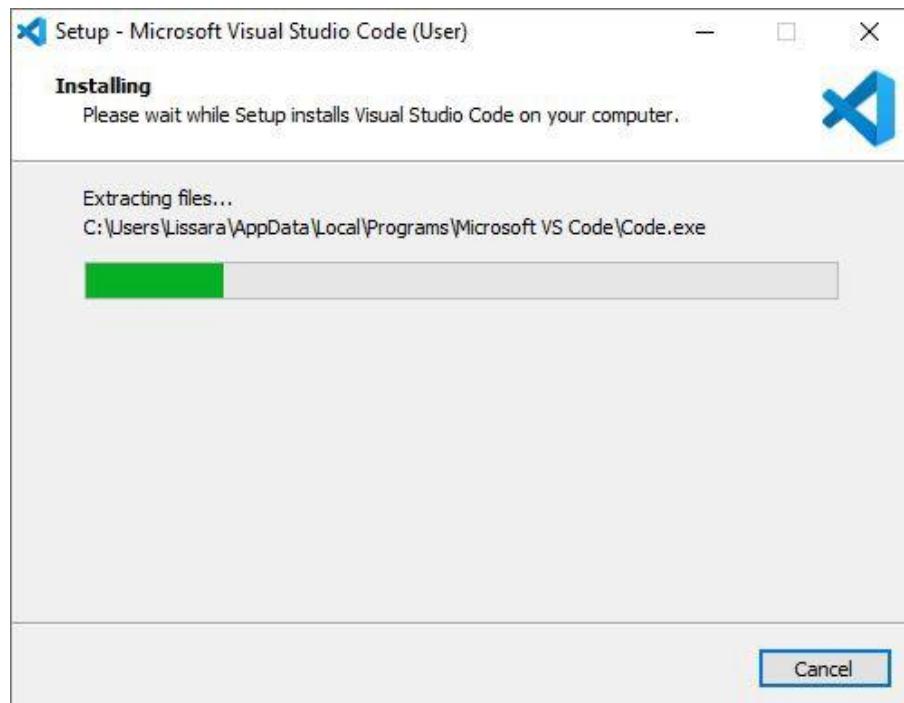
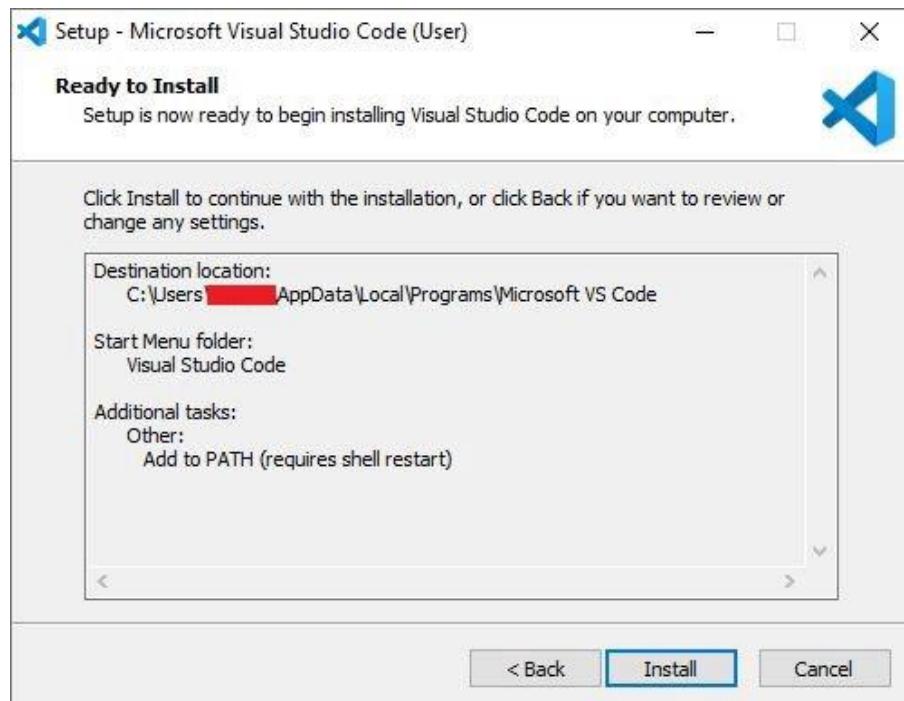
6. คลิกปุ่ม “Next”



7. เลือกส่วนเพิ่มงานให้เลือก Create a desktop icon และ Add to PATH (requires shell restart) จากนั้นให้คลิกปุ่ม “Next”



8. คลิกปุ่ม “Install” เพื่อติดตั้งโปรแกรม





9. คลิกปุ่ม “Finish” เสร็จสิ้นการติดตั้งโปรแกรม VS Code



6. สรุป



ในบทนี้ผู้เรียนจะได้เรียนรู้เกี่ยวกับภาษาไพทอน เป็นต้น แนะนำเครื่องมือสำหรับการเขียนโปรแกรม การดาวน์โหลด ติดตั้ง และเรียกใช้งานโปรแกรมไพทอน, PyCharm, Jupyter Notebook, VS Code ซึ่งเป็นเครื่องมือที่นิยมนำมาเขียนโปรแกรมภาษาไพทอน

7. แบบฝึกหัดท้ายบท



1. จงอธิบายคุณสมบัติเด่นของโปรแกรมภาษาไพทอน
2. ให้นักศึกษาดาวน์โหลดและติดตั้งโปรแกรม IDLE
3. ให้นักศึกษาดาวน์โหลดและติดตั้งโปรแกรม PyCharm และ PyCharm Community
4. ให้นักศึกษาดาวน์โหลดและติดตั้งโปรแกรม Jupyter Notebook
5. ให้นักศึกษาดาวน์โหลดและติดตั้งโปรแกรม VS Code

บทที่ 2

พื้นฐานการเขียนโปรแกรมภาษาไพทอน

ในบทนี้จะกล่าวถึงพื้นฐานการเขียนโปรแกรมภาษาไพทอน ที่มีองค์ประกอบและข้อกำหนดที่สำคัญ ที่เราควรรู้จักในเบื้องต้นที่จะกล่าวถึงในบทนี้ เช่น การเขียนและรันโปรแกรมไพทอน การใช้งานคำสั่งพื้นฐานที่สำคัญ รูปแบบการเขียนโปรแกรม การแสดงผล ตัวแปรและการกำหนดค่า เป็นต้น

1. คำสั่งที่เขียนในโปรแกรมไพทอน (Statement Python)

Statement คือ คำสั่งที่เขียนในโปรแกรมที่สามารถทำงานได้ โดยในภาษาไพทอน นั้นสามารถแบ่งได้ ด้วยการขึ้นบรรทัดใหม่ หรือใช้เครื่องหมายเซมิโคลอน (;) เมื่อไหร่ก็ได้

- Single statement คือ Statement ที่มีคำสั่งเดียว
- Compound statement คือ Statement ที่มีหลายคำสั่ง
การใช้เซมิโคลอน (;) ต่อท้ายคำสั่ง มีข้อดีคือการแบ่งคำสั่ง หรือ statement จะทำให้สามารถเขียนหลายคำสั่งในบรรทัดเดียวกันได้



ตัวอย่างที่ 2.1 คำสั่งที่เขียนในโปรแกรม

```
print("Hello World")
print("Welcome to Python Language.")
print("Python is a popular programming language")

print("Hello World"); print("Welcome to Python Language.");
print("Python is a popular programming language");
```

2. การแสดงผลข้อมูลด้วยฟังก์ชัน print()

เมื่อเราต้องการแสดงข้อมูลออกไปยังหน้าจอ มีรูปแบบพื้นฐานฟังก์ชัน print() ดังนี้

```
print(value, ..., sep = ' ', end = '\n');
```

ในรูปแบบการใช้งาน ฟังก์ชัน print() เราสามารถส่งอาร์กิวเมนต์ได้ตั้งแต่หนึ่งถึงหลายตัวเข้าไปในฟังก์ชัน นอกจากนี้ฟังก์ชันยังมี keyword อาร์กิวเมนต์ sep ซึ่งเป็นตัวแบ่งหากอาร์กิวเมนต์ที่ส่งเข้าไปนั้นมากกว่า 1 ตัว ซึ่งมีค่า default เป็น whitespace และ keyword อาร์กิวเมนต์ end เป็นการแสดงผลในตอนท้ายของฟังก์ชัน ซึ่งมีค่า default เป็น \n หมายถึงการขึ้นบรรทัดใหม่ มาดูตัวอย่างการใช้งานฟังก์ชัน



ตัวอย่างที่ 2.2 แสดงผลข้อมูลด้วยฟังก์ชัน print

```
print("Hello")
print(123)
print(1.22)
com = 'computer'
year = 2564
print(com)
print(year)
```

```
C:\Users\HPSurajak\PycharmProjects
Hello
123
1.22
computer
2564
Process finished with exit code 0
```

3. การรับข้อมูลทางคีย์บอร์ดด้วยฟังก์ชัน input()

ฟังก์ชัน input() ทำหน้าที่รับค่าทางคีย์บอร์ดจากผู้ใช้งาน สำหรับการรับค่าข้อความ ตัวเลข จำนวนหนึ่ง ซึ่งเป็นประเภทฟังก์ชัน Built-in การเรียกใช้งานดังตัวอย่างต่อไปนี้



ตัวอย่างที่ 2.3 การรับข้อมูลชื่อ สกุล และอายุ

```
f_name = input("Enter your name: ")
l_name = input("Enter your surname: ")
age = input("Input your age: ")
print(" ชื่อ: ", f_name)
print(" นามสกุล: ", l_name)
print(" อายุ: ", age)
print("Hello " + f_name + ' ' + l_name, " age: ", age)
```

```
C:\Users\HPSurajak\PycharmProjects\pythonProject1
Enter your name: Lawan
Enter your surname: Dulyachart
Input your age: 18
ชื่อ: Lawan
นามสกุล: Dulyachart
อายุ: 18
Hello Lawan Dulyachart age: 18
Process finished with exit code 0
```

4. การเขียนคำอธิบายโค้ด

คำอธิบายโค้ด (Comment) เป็นข้อความคิดเห็นสามารถใช้เพื่ออธิบายโค้ดของโปรแกรมให้ท่อน ทำให้โค้ดอ่านง่ายขึ้น อาจเขียนไว้เพื่อเตือนความจำ โดยการเขียนคำอธิบายโค้ดมี 2 รูปแบบ คือ การเขียนคำอธิบายแบบบรรทัดเดียว(Line Comment) และการเขียนคำอธิบายแบบหลายบรรทัด(Multi Line Comments) ดังนี้

- การเขียนคำอธิบายแบบบรรทัดเดียว(Line Comment) สำหรับเขียนคำอธิบายบรรทัดเดียว โดยใช้เครื่องหมาย Hash (#) สามารถวิเคราะห์ความคิดเห็นไว้ที่บรรทัดและไฟล่อน จะลงทะเบียนส่วนที่เหลือของบรรทัด เช่น



ตัวอย่างที่ 2.4 การใช้งาน พังก์ชัน print()

```
#This is a comment
print("Hello, World!")
print("Hello, World!") #This is a comment
```

output

```
"Hello, World!"
"Hello, World!"
```

Comment ไม่จำเป็นต้องเป็นข้อความเพื่ออธิบายโค้ด แต่ยังสามารถใช้เพื่อป้องกันไฟฟอนในการเรียกใช้งานโค้ด:

```
#print("Hello, World!")
print("Cheers, Mate!")
```

output

```
Cheers, Mate!
```

- การเขียนคำอธิบายแบบหลายบรรทัด(Multi Line Comments)

ไฟฟอน ไม่มีไวยากรณ์สำหรับ Comment หลายบรรทัด ในการเพิ่มความคิดเห็นหลายบรรทัดสามารถแทรก `#` สำหรับแต่ละบรรทัด หรือสามารถใช้สตริงหลายบรรทัด เนื่องจากไฟฟอน จะละเว้นตัวอักซ์รัลลิสต์ที่ไม่ได้กำหนดให้กับตัวแปรสามารถเพิ่มสตริงหลายบรรทัด (""" ... """) หรือ ("" ... "")



ตัวอย่างที่ 2.5 การเขียนคำอธิบายแบบหลายบรรทัด

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

```
demo_comment4.py:
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

```
C:\Users\My Name>python demo_comment4.py
Hello, World!
```



ตัวอย่างที่ 2.6 แสดงการเขียนคำอธิบายหลายบรรทัด

```
"""
```

```
This is a comment  
written in  
more than just one line  
"""  
  
print("Hello, World!")
```

```
"""
```

```
This is a comment  
written in  
more than just one line  
"""  
  
print("Hello, World!")
```

```
C:\Users\My Name>python demo_comment5.py  
Hello, World!
```

5. การกำหนดขอบเขตของคำสั่ง (Indentation)

ภาษาไพทอน ใช้การเว้นวรรค หรือ tab ในการกำหนดขอบเขตของโปรแกรม โดยการเว้นวรรค ต้องเว้นวรรค 4 ครั้ง ดังนั้นแนะนำให้ใช้ tab ใน การกำหนดขอบเขตของคำสั่ง ระบุบล็อกของ code โปรแกรม จากตัวอย่างจะเห็นว่า บรรทัดที่ 3 มี tab หน้าคำสั่ง ดังนั้นคำสั่ง print("Five is greater than two!") อยู่ในขอบเขตของ if block จะทำงานต่อเมื่อ $5 > 2$ เป็นจริงเท่านั้น



ตัวอย่างที่ 2.7 แสดงการกำหนดขอบเขตของคำสั่ง

```
if 5 > 2:  
    print("Five is greater than two!")
```

```
demo_indentation.py:  
if 5 > 2:  
    print("Five is greater than two!")
```

```
C:\Users\My Name>python demo_indentation.py  
Five is greater than two!
```

ไพทอนจะทำให้เกิดข้อผิดพลาดหากข้ามการเยื่อง ดังตัวอย่าง จำนวนช่องว่างต้องมีอย่างน้อยหนึ่งช่อง



ตัวอย่างที่ 2.8 การเขียน

```
if 5 > 2:  
    print("Five is greater than two!")  
  
if 5 > 2:  
    print("Five is greater than two!")
```

```
demo_indentation2_error.py:  
if 5 > 2:  
    print("Five is greater than two!")  
        print("Five is greater than two!")
```

```
C:\Users\My Name>python demo_indentation2_error.py  
File "demo_indentation2_error.py", line 3  
    print("Five is greater than two!")  
          ^  
IndentationError: unexpected indent
```

ต้องใช้ช่องว่างจำนวนเดียวกันในบล็อกของรหัสเดียวกันมิฉะนั้นไพทอน จะทำให้มีข้อผิดพลาดของไวยกรณ์



ตัวอย่างที่ 2.9 ข้อผิดพลาดของไวยกรณ์

```
if 5 > 2:  
    print("Five is greater than two!")  
        print("Five is greater than two!")
```

```
demo_indentation2_error.py:  
if 5 > 2:  
    print("Five is greater than two!")  
        print("Five is greater than two!")
```

```
C:\Users\My Name>python demo_indentation2_error.py  
File "demo_indentation2_error.py", line 3  
    print("Five is greater than two!")  
          ^  
IndentationError: unexpected indent
```

6. ตัวแปร (Python Variables)

ตัวแปร คือ คุณเท่านেอร์สำหรับการจัดเก็บค่าข้อมูล ไม่เหมือนกับภาษาอื่น ๆ ใน การเขียน โปรแกรมไพทอน ไม่มีคำสั่งให้ประกาศตัวแปร ตัวแปรจะถูกสร้างขึ้นเมื่อกำหนดค่าให้กับมันเป็นครั้งแรก



ตัวอย่างที่ 2.10 ตัวแปร

```
x = 5  
y = "John"  
print(x)  
print(y)
```

```
demo_variables1.py:  
x = 5  
y = "John"  
print(x)  
print(y)
```

```
C:\Users\My Name>python demo_variables1.py  
5  
John
```



ตัวแปรไม่จำเป็นต้องถูกประกาศด้วยชนิดเฉพาะได้ ๆ และยังสามารถเปลี่ยนชนิดหลังจากที่พิเศษได้รับการตั้งค่า



ตัวอย่างที่ 2.11 การเปลี่ยนชนิดตัวแปร

```
x = 4      # x is of type int
x = "Sally" # x is now of type str
print(x)
```

```
demo_variables2.py:
x = 4
x = "Sally"
print(x)
```

```
C:\Users\My Name>python demo_variables2.py
Sally
```

ตัวแปรสริงสามารถประกาศได้โดยใช้เครื่องหมายคำพูดเดี่ยว(' ') หรือคำพูดคู่ (" ") :

```
x = "John"
# is the same as
x = 'John'
```

```
demo_variables7.py:
x = "John"
print(x)
#double quotes are the same as single quotes:
x = 'John'
print(x)
```

```
C:\Users\My Name>python demo_variables7.py
John
John
```

6.1 ชื่อตัวแปร (Variable Names)

ตัวแปรสามารถมีชื่อแบบสั้น (เช่น x และ y) หรือชื่อที่สื่อความหมายได้มากขึ้น (อายุชื่อ carname, total_volume) กว่าสำหรับตัวแปรไฟฟอน:

- ชื่อตัวแปรจะต้องเริ่มต้นด้วยตัวอักษรหรือตัวอักษรขีดล่าง_(Underscore)
- ชื่อตัวแปรไม่สามารถเริ่มต้นด้วยตัวเลข
- ชื่อตัวแปรสามารถมีได้ทั้งตัวอักษรตัวเลขและขีดล่าง (A-z, 0-9 และ _)
- ชื่อตัวแปรตัวพิมพ์เล็กและตัวพิมพ์ใหญ่จะแตกต่างกัน เรียกว่าเป็น case sensitive
- ห้ามตั้งชื่อซ้ำกับคำส่วน (Reserved word) ในภาษาไฟฟอน ดังนี้

ตารางที่ 1 คำส่วน

and	def	exec	if	not	return
assert	del	finally	import	or	try
break	elif	for	in	pass	while
class	else	from	is	print	with
continue	except	global	lambda	raise	yield

6.2 กำหนดค่าให้กับหลายตัวแปร (Assign Value to Multiple Variables)

ไฟตอน อนุญาตให้กำหนดค่าให้กับตัวแปรหลายตัวในหนึ่งบรรทัด:

 ตัวอย่างที่ 2.12 กำหนดค่าให้กับตัวแปรหลายตัวในหนึ่งบรรทัด

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

<pre>demo_variables8.py: x, y, z = "Orange", "Banana", "Cherry" print(x) print(y) print(z)</pre>	<pre>C:\Users\My Name>python demo_variables8.py Orange Banana Cherry</pre>
--	---

และสามารถกำหนดค่าเดียวกันให้กับตัวแปรหลายตัวในหนึ่งบรรทัด:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

<pre>demo_variables6.py: x = y = z = "Orange" print(x) print(y) print(z)</pre>	<pre>C:\Users\My Name>python demo_variables6.py Orange Orange Orange</pre>
--	---

6.3 ตัวแปรเอาต์พุต

คำสั่ง Print มักถูกใช้กับตัวแปรแสดงผลลัพธ์ เพื่อร่วมทั้งข้อความ และตัวแปรโดยใช้ + ข้อความ:

 ตัวอย่างที่ 2.13 ตัวแปรเอาต์พุต

```
x = "awesome"
print("Python is " + x)
```

<pre>demo_variables3.py: x = "awesome" print("Python is " + x)</pre>	<pre>C:\Users\My Name>python demo_variables3.py Python is awesome</pre>
--	--

สามารถใช้ + อักษรเพื่อเพิ่มตัวแปรให้กับตัวแปรอื่น:



```
x = "Python is "
y = "awesome"
z = x + y
print(z)
```

```
demo_variables4.py:
x = "Python is "
y = "awesome"
z = x + y
print(z)
```

```
C:\Users\My Name>python demo_variables4.py
Python is awesome
```

สำหรับตัวเลข + เป็นตัวดำเนินการทางคณิตศาสตร์:

```
x = 5
y = 10
print(x + y)
```

```
demo_variables5.py:
x = 5
y = 10
print(x + y)
```

```
C:\Users\My Name>python demo_variables5.py
15
```

หากพยายามรวมสตริงและตัวเลขใน Python จะให้ข้อผิดพลาด:

```
x = 5
y = "John"
print(x + y)
```

```
demo_variables_test.py:
x = 5
y = "John"
print(x + y)
```

```
C:\Users\My Name>python demo_variables_test.py
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

6.4 ตัวแปรโกลบอล

ตัวแปรที่สร้างขึ้นนอกฟังก์ชัน (ดังในตัวอย่างด้านบนทั้งหมด) เรียกว่า ตัวแปรโกลบอล ทุกคนสามารถใช้ตัวแปรส่วนกลางได้ทั้งภายในและภายนอก



ตัวอย่างที่ 2.14 ตัวแปรโกลบอล

```
x = "awesome"

def myfunc():
    print("Python is " + x)

myfunc()
```

<pre>demo_variables_global.py: x = "awesome" def myfunc(): print("Python is " + x) myfunc()</pre>	<pre>C:\Users\My Name>python demo_variables_global.py Python is awesome</pre>
---	--

หากสร้างตัวแปรที่มีชื่อเดียวกันภายในฟังก์ชันตัวแปรนี้จะเป็นแบบโลคัล และสามารถใช้ได้ภายในฟังก์ชันเท่านั้น ตัวแปรโกลบอลที่มีชื่อเดียวกันจะยังคงเหมือนเดิม และมีค่าเดิม ดังตัวอย่าง สร้างตัวแปรภายในฟังก์ชันโดยใช้ชื่อเดียวกับตัวแปรกลาง

<pre>x = "awesome" def myfunc(): x = "fantastic" print("Python is " + x) myfunc() print("Python is " + x)</pre>	<pre>demo_variables_global2.py: x = "awesome" def myfunc(): x = "fantastic" print("Python is " + x) myfunc() print("Python is " + x)</pre>	<pre>C:\Users\My Name>python demo_variables_global2.py Python is fantastic Python is awesome</pre>
--	---	---

6.5 The global Keyword

โดยปกติเมื่อสร้างตัวแปรภายในฟังก์ชันตัวแปรนั้นจะเป็นแบบ local สามารถใช้ได้ภายในฟังก์ชันนั้นเท่านั้น เพื่อสร้างตัวแปรส่วนกลางภายในฟังก์ชันสามารถใช้คำว่า global



ตัวอย่างที่ 2.15 global Keyword

```
#หากใช้ global ตัวแปรจะอยู่ในขอบเขตส่วนกลาง:
def myfunc():
    global x
    x = "fantastic"

myfunc()

print("Python is " + x)
```



```
demo_variables_global4.py:
x = "awesome"

def myfunc():
    global x
    x = "fantastic"

myfunc()

print("Python is " + x)
```

```
C:\Users\My Name>python demo_variables_global4.py
Python is fantastic
```

ใช้ global Keyword หากต้องการเปลี่ยนตัวแปร globlal ภายในฟังก์ชัน

#เปลี่ยนค่าของตัวแปร globlal ภายในฟังก์ชันให้อ้างอิงกับตัวแปรโดยใช้ global คีย์เวิร์ด:

```
x = "awesome"

def myfunc():
    global x
    x = "fantastic"

myfunc()

print("Python is " + x)
```

```
demo_variables_global4.py:
x = "awesome"

def myfunc():
    global x
    x = "fantastic"

myfunc()

print("Python is " + x)
```

```
C:\Users\My Name>python demo_variables_global4.py
Python is fantastic
```

7. ชนิดข้อมูลแบบ Built-in (Built-in Data Types)

ในการเขียนโปรแกรมชนิดข้อมูลเป็นแนวคิดที่สำคัญ ตัวแปรสามารถเก็บข้อมูลประเภทต่าง ๆ และประเภทต่าง ๆ สามารถทำสิ่งต่าง ๆ ได้ ไฟทอน มีชนิดข้อมูลต่อไปนี้ตามค่าเริ่มต้นในหมวดหมู่เหล่านี้:

ตารางที่ 2 ชนิดข้อมูลแบบ Built-in

ประเภทข้อมูล:	
ประเภทตัวเลข:	str
ประเภทลำดับ:	int, float, complex
ประเภทจานวน:	list, tuple, range
ประเภทเซต:	dict
ประเภทบูลีน:	set, frozenset
ประเภทไบนารี:	bool
	bytes, bytearray, memoryview

7.1 การรับชนิดข้อมูล (Getting the Data Type)

สามารถรับชนิดข้อมูลของวัตถุได้ ๆ โดยใช้ฟังก์ชัน `type()`: ดังตัวอย่าง พิมพ์ประเภทข้อมูลของตัวแปร `x`:

 ตัวอย่างที่ 2.16 รับชนิดข้อมูล

```
x = 5
print(type(x))
```

demo_type.py:
x = 5
print(type(x))

C:\Users\My Name>python demo_type.py
<class 'int'>

7.2 การกำหนดค่าให้ชนิดข้อมูล (Setting the Data Type)

ในไฟล์ Python ชนิดข้อมูลจะถูกตั้งค่าเมื่อกำหนดค่าให้กับตัวแปร:

ตารางที่ 3 การกำหนดค่าให้ชนิดข้อมูล

ตัวอย่าง	ประเภทข้อมูล
<code>x = "Hello World"</code>	str
<code>x = 20</code>	int
<code>x = 20.5</code>	float
<code>x = 1j</code>	complex
<code>x = ["apple", "banana", "cherry"]</code>	list
<code>x = ("apple", "banana", "cherry")</code>	tuple
<code>x = range(6)</code>	range
<code>x = {"name" : "John", "age" : 36}</code>	dict
<code>x = {"apple", "banana", "cherry"}</code>	set
<code>x = frozenset({"apple", "banana", "cherry"})</code>	frozenset
<code>x = True</code>	bool
<code>x = b"Hello"</code>	bytes
<code>x = bytearray(5)</code>	bytearray
<code>x = memoryview(bytes(5))</code>	memoryview



7.3 การระบุชนิดข้อมูลพร้อมข้อมูล (Setting the Specific Data Type)

ถ้าต้องการระบุชนิดข้อมูลสามารถใช้ฟังก์ชัน Constructor ต่อไปนี้:

ตารางที่ 4 การระบุชนิดข้อมูล

ตัวอย่าง	ชนิดข้อมูล
x = str("Hello World")	str
x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = list("apple", "banana", "cherry")	list
x = tuple("apple", "banana", "cherry")	tuple
x = range(6)	range
x = dict(name="John", age=36)	dict
x = set("apple", "banana", "cherry")	set
x = frozenset("apple", "banana", "cherry")	frozenset
x = bool(5)	bool
x = bytes(5)	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

8. ข้อมูลชนิดตัวเลข (Python Numbers)

ไพทอน มีประเภทตัวเลข 3 ประเภท คือ int, float และ complex ตัวแปรประเภทตัวเลขถูกสร้างขึ้นเมื่อกำหนดค่าให้กับตัวแปร: ในการตรวจสอบประเภทของวัตถุใด ๆ ในไพทอน ให้ใช้ฟังก์ชัน `type()`



ตัวอย่างที่

```
x = 1    # int
y = 2.8 # float
z = 1j   # complex

print(type(x))
print(type(y))
print(type(z))
```

demo_numbers.py:	C:\Users\My Name>python demo_numbers.py
x = 1 y = 2.8 z = 1j	<class 'int'> <class 'float'> <class 'complex'>



8.1 จำนวนเต็ม (int)

int หรือ integer เป็นจำนวนเต็มบวก หรือ ลบ โดยไม่มีทศนิยมความยาวไม่จำกัด

ตัวอย่างที่

```
x = 1
y = 35656222554887711
z = -3255522
print(type(x))
print(type(y))
print(type(z))
```

demo_numbers_float.py:

```
x = 1.10
y = 1.0
z = -35.59
print(type(x))
print(type(y))
print(type(z))
```

C:\Users\My Name>python demo_numbers_float.py

```
<class 'float'>
<class 'float'>
<class 'float'>
```

8.2 เลขทศนิยม (Float)

Float หรือ "floating point number (เลขทศนิยม)" เป็นตัวเลขบวก หรือ ลบ ประกอบด้วย ทศนิยมหนึ่งตัว หรือมากกว่า

ตัวอย่างที่ 2.17 ข้อมูลชนิดตัวเลข

```
x = 1.10
y = 1.0
z = -35.59
print(type(x))
print(type(y))
print(type(z))
```

demo_numbers_float2.py:

```
x = 35e3
y = 12E4
z = -87.7e100
print(type(x))
print(type(y))
print(type(z))
```

C:\Users\My Name>python demo_numbers_float2.py

```
<class 'float'>
<class 'float'>
<class 'float'>
```

float สามารถเป็นตัวเลขทางวิทยาศาสตร์ด้วย "e" เพื่อระบุเลขยกกำลังของ 10

```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```

```
demo_numbers_float2.py:
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```

```
C:\Users\My Name>python demo_numbers_float2.py
<class 'float'>
<class 'float'>
<class 'float'>
```

8.3 จำนวนเชิงซ้อน (Complex)

Complex numbers (จำนวนเชิงซ้อน) เขียนด้วย "j" เป็นส่วนจินตภาพ:



ตัวอย่างที่ 2.18 จำนวนเชิงซ้อน

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

```
demo_numbers_complex.py:
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

```
C:\Users\My Name>python demo_numbers_complex.py
<class 'complex'>
<class 'complex'>
<class 'complex'>
```

8.4 การแปลงประเภทข้อมูล (Type Conversion)

สามารถแปลงจากประเภทหนึ่งไปยังอีกด้วย int(), float() และcomplex() :



ตัวอย่างที่ 2.19 การแปลงประเภทข้อมูล

```
x = 1 # int
y = 2.8 # float
z = 1j # complex

#convert from int to float:
```



```
a = float(x)
#convert from float to int:
b = int(y)
#convert from int to complex:
c = complex(x)
print(a)
print(b)
print(c)
print(type(a))
print(type(b))
print(type(c))
```

```
demo_numbers_convert.py:
#convert from int to float:
x = float(1)

#convert from float to int:
y = int(2.8)

#convert from int to complex:
z = complex(x)

print(x)
print(y)
print(z)
```

```
C:\Users\My Name>python demo_numbers_convert.py
1.0
2
(1+0j)
<class 'float'> <class 'int'> <class 'complex'>
```



หมายเหตุ: ไม่สามารถแปลงจำนวนเชิงซ้อนเป็นประเภทตัวเลขอื่นได้

8.5. ตัวเลขสุ่ม (Random Number)

ไฟฟอน ไม่มีฟังก์ชัน `random()` ในการสร้างตัวเลขสุ่ม แต่ไฟฟอน มีโมดูลในตัวที่เรียกว่า `random` สามารถใช้เพื่อสร้างตัวเลขสุ่ม:



ตัวอย่างที่ 2.20 นำเข้าโมดูลสุ่มและแสดงตัวเลขสุ่มระหว่าง 1 ถึง 9:

```
import random
print(random.randrange(1,10))
```

```
demo_numbers_random.py:
import random
print(random.randrange(1,10))
```

```
C:\Users\My Name>python demo_numbers_random.py
5
```

8.6 การแปลงชนิดข้อมูล (Python Casting)

▪ ระบุประเภทตัวแปร

อาจมีบางครั้งที่ต้องการระบุประเภทให้กับตัวแปร นี้สามารถทำได้ด้วยการ Casting เป็นภาษาเชิงวัตถุและด้วยเหตุนี้จึงใช้คลาสเพื่อกำหนดชนิดข้อมูลรวมถึงประเภทเดิม ดังนั้น Casting ในไฟฟอน จึงทำได้โดยใช้ฟังก์ชัน:

1. `int ()` – สร้างตัวเลขจำนวนเต็ม
2. `float ()` – สร้างจำนวนทศนิยม
3. `str ()` – สร้างสตริง จากประเภทข้อมูลที่หลากหลายรวมถึงสตริงตัวอักษรจำนวนเต็มและตัวอักษรโดย



ตัวอย่างที่ 2.21 โค้ดคำสั่งการแปลงชนิดข้อมูลจำนวนเต็ม

```
x = int(1)
```

```
y = int(2.8)
```

```
z = int("3")
```

```
print(x)
```

```
print(y)
```

```
print(z)
```

output

```
1
```

```
2
```

```
3
```



ตัวอย่างที่ 2.22 โค้ดคำสั่งการแปลงชนิดข้อมูลจำนวนทศนิยม

```
x = float(1)
y = float(2.8)
z = float("3")
w = float("4.2")

print(x)
print(y)
print(z)
print(w)
```

output

1.0
2.8
3.0
4.2



ตัวอย่างที่ 2.23 โค้ดคำสั่งการแปลงชนิดข้อมูลจำนวนสตริง

```
x = str("s1")
y = str(2)
z = str(3.0)

print(x)
print(y)
print(z)
```

output

s1
2
3.0

9. ชนิดข้อมูลสตริง (Python Strings)

- 9.1 Python Strings
- 9.2 Slicing Strings
- 9.3 Modify Strings
- 9.4 Concatenate Strings
- 9.5 Format Strings
- 9.6 Escape Characters
- 9.7 String Methods

9.1 ชนิดข้อมูลสตริง (Python Strings)

สตริง หรือตัวอักษรในไฟล์ Python ต้องครอบด้วยเครื่องหมายคำพูดเดี่ยว (single quotation marks) หรือเครื่องหมายคำพูดคู่ (double quotation marks) ‘hello’ ใช้ได้เช่นเดียวกับ “hello” สามารถแสดงตัวอักษรด้วย ฟังก์ชัน print():


ตัวอย่างที่ 2.24 ชนิดข้อมูลสตริง

```
print("Hello")
print('Hello')
```

<pre>demo_string_literal.py: #You can use double or single quotes: print("Hello") print('Hello')</pre>	C:\Users\My Name>python demo_string_literal.py Hello Hello
--	--

- การกำหนดสตริงให้กับตัวแปร (Assign String to a Variable)

การกำหนดสตริงให้กับตัวแปรทำด้วยชื่อตัวแปรตามด้วยเครื่องหมายเท่ากับและสตริง:


ตัวอย่างที่ 2.25 กำหนดสตริงให้กับตัวแปร

```
a = "Hello"
print(a)
```

<pre>demo_string_var.py: a = "Hello" print(a)</pre>	C:\Users\My Name>python demo_string_var.py Hello
---	--



■ การกำหนดสตริงหลายบรรทัด (Multiline Strings)

สามารถกำหนดสตริงหลายบรรทัดให้กับตัวแปรโดยใช้เครื่องหมายคำพูด 3 คำ “”” (three quotes) หรือ ‘’’ (three single quotes):



ตัวอย่างที่ 2.26 กำหนดสตริงหลายบรรทัด

```
a = “”” คอมพิวเตอร์
```

```
คณะศึกษาศาสตร์และนวัตกรรมการศึกษาและนวัตกรรมการศึกษา
```

```
มหาวิทยาลัยกาฬสินธุ์
```

```
“””
```

```
print(a)
```

```
a = ‘’’Computer
```

```
Faculty of Education and Educational Innovation
```

```
Kalasin University
```

```
‘’’
```

```
print(a)
```

```
C:\Users\HPSurajak\PycharmProjects\pythonProject1\
```

```
คอมพิวเตอร์
```

```
คณะศึกษาศาสตร์และนวัตกรรมการศึกษาและนวัตกรรมการศึกษา
```

```
มหาวิทยาลัยกาฬสินธุ์
```

```
Computer
```

```
Faculty of Education and Educational Innovation
```

```
Kalasin University
```

```
Process finished with exit code 0
```



หมายเหตุ: ในผลลัพธ์การแบ่งบรรทัดจะถูกแทรกในตำแหน่งเดียวกับในโค้ด

■ อาเรย์ในสตริง (Strings are Arrays)

เช่นเดียวกับภาษาการเขียนโปรแกรมยอดนิยมอื่น ๆ ตัวอักษรในไฟฟอนเป็นอาร์เรย์ของไบต์ที่แสดงถึงอักขระ Unicode อย่างไรก็ตามไฟฟอนไม่มีประเภทข้อมูลตัวอักษรอักขระเดียวเป็นเพียงสตริงที่มีความยาว 1 [] สามารถใช้เพื่อเข้าถึงองค์ประกอบของสตริง



ตัวอย่างที่ 2.27 อาเรย์ในสตริง

```
a = “Hello, World!”
```

```
print(a [1])
```

output

e

- วนลูปผ่านสตริง (Looping Through a String)

เนื่องจากสตริงเป็นอาร์เรย์ เราจึงสามารถวนซ้ำอักษรในสตริงด้วยการ for วนซ้ำ



ตัวอย่างที่ 2.28 วนลูปผ่านสตริง

```
for x in "banana":
```

```
    print(x)
```

Output

```
b  
a  
n  
a  
n  
a
```

- การนับความยาวของสตริง (String Length)

ในการนับความยาวของสตริงให้ใช้ `len()` ฟังก์ชัน



ตัวอย่างที่ 2.29 นับความยาวของสตริง

```
a = "My name is Lawan!"
```

```
print(len(a))
```

Output

```
17
```

- ตรวจสอบสตริง (Check String)

เพื่อตรวจสอบว่า วลีบางอย่าง หรือตัวอักษรที่อยู่ในสตริงเราสามารถใช้คำหลัก `in` หรือ `not in`



ตัวอย่างที่ 2.30 ตรวจสอบว่าวลี “ain” มีอยู่ในข้อความต่อไปนี้:

```
txt = "The rain in Spain stays mainly in the plain"
```

```
x = "ain" in txt
```

```
print(x)
```

```
demo_string_in.py:  
txt = "The rain in Spain stays mainly in the plain"  
x = "ain" in txt  
print(x)
```

```
C:\Users\My Name>python demo_string_in.py  
True
```



ตัวอย่างที่ 2.31 ใช้ in ในคำสั่ง if :

```
txt = "The best things in life are free!"  
if "free" in txt:  
    print("Yes, 'free' is present.")
```

Output

Yes, 'free' is present.



ตัวอย่างที่ 2.32 ใช้ not in ในคำสั่ง if :

```
txt = "The best things in life are free!"  
if "expensive" not in txt:  
    print("Yes, 'expensive' is NOT present.")
```

Output

Yes, 'expensive' is NOT present.

9.2 Slicing Strings

สามารถส่งกลับช่วงของอักขระโดยใช้ไวยากรณ์หั่นหรือแบ่ง ระบุด้ชนีเริ่มต้นและด้ชนีสิ้นสุดคันด้วยเครื่องหมายโคลอน : เพื่อส่งคืนส่วนของสตริง



ตัวอย่างที่ 2.33 Slicing

```
b = "Hello, World!"  
print(b[2:5])
```

Output

llo



ตัวอย่างที่ 2.34 Slice From the Start

```
b = "Hello, World!"  
print(b[:5])
```

Output

Hello



ตัวอย่างที่ 2.35 Slice To the End

```
b = "Hello, World!"
print(b[2:])
```

Output

llo, World!



ตัวอย่างที่ 2.36 Negative Indexing

```
b = "Hello, World!"
print(b[-5:-2])
```

Output

orl

9.3 การแก้ไขสตริง (Modify Strings)

ไพทอน มีชุดเมธอดในตัวที่สามารถใช้กับสตริงได้

- ตัวพิมพ์ใหญ่



ตัวอย่างที่ 2.37 เมธอด upper() ส่งกลับสตริงในกรณีที่ อักษรตัวพิมพ์ใหญ่

```
a = "Hello, World!"
print(a.upper())
```

```
demo_string_upper.py:
a = "Hello, World!"
print(a.upper())
```

```
C:\Users\My Name>python demo_string_upper.py
HELLO, WORLD!
```

- ตัวพิมพ์เล็ก



ตัวอย่างที่ 2.38 lower() เมธอดส่งกลับสตริงในกรณีที่ อักษรตัวพิมพ์เล็ก

```
a = "Hello, World!"
print(a.lower())
```

```
demo_string_lower.py:
a = "Hello, World!"
print(a.lower())
```

```
C:\Users\My Name>python demo_string_lower.py
hello, world!
```



- ลบช่องว่าง



ตัวอย่างที่ 2.39 เมธอด strip() ขัดช่องว่างได้ ๆ จากจุดเริ่มต้นหรือจุดสิ้นสุด:

```
a = "Hello, World!"
print(a.strip()) # returns "Hello, World!"
```

```
demo_string_strip.py:
a = "Hello, World!"
print(a.strip())
```

```
C:\Users\My Name>python demo_string_strip.py
Hello, World!
```

- แทนที่สตริง



ตัวอย่างที่ 2.40 replace() วิธีการแทนที่สตริงกับสตริงอื่น:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

```
demo_string_replace.py:
a = "Hello, World!"
print(a.replace("H", "J"))
```

```
C:\Users\My Name>python demo_string_replace.py
Jello, World!
```

- แยกสตริง



ตัวอย่างที่ 2.41 split() วิธีการแยกสตริงเข้าสตริงหากพบว่าการนี้ของคัน:

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

```
demo_string_split.py:
a = "Hello, World!"
b = a.split(",")
print(b)
```

```
C:\Users\My Name>python demo_string_split.py
['Hello', ' World!']
```

9.4 การต่อสตริง (String Concatenation)

ในการเชื่อมหรือรวมสองสตริงสามารถใช้ตัวดำเนินการ +



ตัวอย่างที่ 2.42 รวมตัวแปร a กับตัวแปร b เก็บเข้ากับตัวแปร c:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

หากต้องการเพิ่มช่องว่างระหว่างทันให้เพิ่ม” “:

```
a = "Hello"
b = "World"
```

```
c = a + " " + b
```

```
print(c)
```

```
demo_string_concat.py:
a = "Hello"
b = "World"
c = a + b
print(c)
```

```
C:\Users\My Name>python demo_string_concat.py
HelloWorld
```

9.5 รูปแบบสตริง (Format – Strings)

รูปแบบ

```
Print("Str1 : %s , Str2 : %d , Str3 : %f " %(Argu1, Argu2, Argu3))
```

```
Print("Str1 : {p0} , Str2 : {p1}, Str3 : {p2} " .format(p0,p1,p2))
```

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

ตารางที่ 5 สัญลักษณ์การแสดงรูปแบบชนิดข้อมูล

สัญลักษณ์	ความหมาย
%s	- String แสดงผลอักษรผ่านฟังก์ชันstr ()
%d	- Integers แสดงผลเลขฐานสิบ (1, 0, -1)
%f	- Floating point numbers แสดงเลขทศนิยม %.<number of digits>f กำหนดตำแหน่งทศนิยม เช่น %f, %.1f, %.2f,
%x/%X	แสดงผลเลขฐานสิบหกอักษรตัวเล็ก/ตัวใหญ่ (lowercase/uppercase)

เราไม่สามารถรวมสตริงและตัวเลข ด้วยตัวดำเนินการ + ดังนี้:



ตัวอย่างที่ 2.43 โค้ดคำสั่งการเชื่อมสตริงกับตัวเลข

```
age = 36
```

```
txt = "My name is Lawan, I am " + age
```

```
print(txt)
```

```
Traceback (most recent call last):
```

```
File "C:\Users\HPSurajak\PycharmProjects\FirstApp\main.py", line 2, in <module>
```

```
    txt = "My name is John, I am " + age
```

```
TypeError: can only concatenate str (not "int") to str
```

```
Process finished with exit code 1
```



แต่เราสามารถรวมสตริง และตัวเลขโดยใช้เมธอด `format()` !

เมธอด `format()` ใช้อาร์กิวเมนต์ผ่านรูปแบบ และ {} คือ:



ตัวอย่างที่ 2.44 ใช้เมธอด `format()` แทรกตัวเลขลงในสตริง:

```
age = 36  
txt = "My name is Lawan, and I am {}"  
print(txt.format(age))
```

```
My name is Lawan, and I am 36
```

```
Process finished with exit code 0
```

`format()` เมื่อกดจำนวนของอาเกิมั่นท์ ที่ว่างลง {} ที่เกี่ยวข้อง:

```
quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want {} pieces of item {} for {} dollars."  
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item 567 for 49.95 dollars.
```

```
Process finished with exit code 0
```

สามารถใช้หมายเลขดัชนี {0} เพื่อให้แน่ใจว่ามีการวางอาร์กิวเมนต์ในตัวยึดตำแหน่งที่ถูกต้อง:

```
quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."  
print(myorder.format(quantity, itemno, price))
```

```
I want to pay 49.95 dollars for 3 pieces of item 567.
```

```
Process finished with exit code 0
```

9.6 การใช้อักขระพิเศษ (Escape Character)

- ในการแทรกอักขระที่ผิดกฎหมายในสตริงให้ใช้อักขระเดี่ยง
- อักขระ escape เป็น แบ็คสแลช \ ตามด้วยอักขระที่ต้องการแทรก

- ตัวอย่างของอักษรที่ผิดกฎหมาย คือ เครื่องหมายคำพูดคู่ภายในสตริงที่ล้อมรอบด้วยเครื่องหมายคำพูดคู่ :



ตัวอย่างที่ 2.45 จะเกิดข้อผิดพลาดหากใช้เครื่องหมายคำพูด " " ภายในสตริง " " ซึ่งกัน

```
txt = "We are the so-called "Vikings" from the north."
```

<pre>demo_string_escape_error.py: txt = "We are the so-called "Vikings" from the north. #You will get an error if you use double quotes inside a string that are surrounded by double quotes:</pre>	<pre>C:\Users\My Name>python demo_string_escape_error.py File "demo_string_escape_error.py", line 1 txt = "We are the so-called "Vikings" from the north." ^ SyntaxError: invalid syntax</pre>
--	---

ในการแก้ไขปัญหานี้ให้ใช้ Escape Character \":



ตัวอย่างที่ 2.46 Escape Character อนุญาตให้ใช้เครื่องหมายคำพูดคู่ " " :

```
txt = "We are the so-called \"Vikings\" from the north."
```

<pre>demo_string_escape.py: txt = "We are the so-called \"Vikings\" from the north." print(txt)</pre>	<pre>C:\Users\My Name>python demo_string_escape.py We are the so-called "Vikings" from the north.</pre>
---	--

ตารางที่ 6 อักษร escape อีน ๆ ที่ใช้ในไฟฟอน

Code	Result
\'	Single Quote
\\"	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value



9.7 String Methods

ไพทอน มี built-in method ที่สามารถใช้กับสตริงได้

ตารางที่ 7 String Methods

Method	Description
<code>capitalize()</code>	Converts the first character to upper case
<code>casefold()</code>	Converts string into lower case
<code>center()</code>	Returns a centered string
<code>count()</code>	Returns the number of times a specified value occurs in a string
<code>encode()</code>	Returns an encoded version of the string
<code>endswith()</code>	Returns true if the string ends with the specified value
<code>expandtabs()</code>	Sets the tab size of the string
<code>find()</code>	Searches the string for a specified value and returns the position of where it was found
<code>format()</code>	Formats specified values in a string
<code>format_map()</code>	Formats specified values in a string
<code>index()</code>	Searches the string for a specified value and returns the position of where it was found
<code>isalnum()</code>	Returns True if all characters in the string are alphanumeric
<code>isalpha()</code>	Returns True if all characters in the string are in the alphabet
<code>isdecimal()</code>	Returns True if all characters in the string are decimals
<code>isdigit()</code>	Returns True if all characters in the string are digits
<code>isidentifier()</code>	Returns True if the string is an identifier
<code>islower()</code>	Returns True if all characters in the string are lower case
<code>isnumeric()</code>	Returns True if all characters in the string are numeric
<code>encode()</code>	Returns an encoded version of the string
<code>endswith()</code>	Returns true if the string ends with the specified value
<code>expandtabs()</code>	Sets the tab size of the string
<code>find()</code>	Searches the string for a specified value and returns the position of where it was found
<code>format()</code>	Formats specified values in a string
<code>format_map()</code>	Formats specified values in a string
<code>index()</code>	Searches the string for a specified value and returns the position of where it was found
<code>isalnum()</code>	Returns True if all characters in the string are alphanumeric
<code>isalpha()</code>	Returns True if all characters in the string are in the alphabet
<code>isdecimal()</code>	Returns True if all characters in the string are decimals
<code>isdigit()</code>	Returns True if all characters in the string are digits
<code>isidentifier()</code>	Returns True if the string is an identifier
<code>islower()</code>	Returns True if all characters in the string are lower case
<code>isnumeric()</code>	Returns True if all characters in the string are numeric

Method	Description
<u>isprintable()</u>	Returns True if all characters in the string are printable
<u>isspace()</u>	Returns True if all characters in the string are whitespaces
<u>istitle()</u>	Returns True if the string follows the rules of a title
<u>isupper()</u>	Returns True if all characters in the string are upper case
<u>join()</u>	Joins the elements of an iterable to the end of the string
<u>ljust()</u>	Returns a left justified version of the string
<u>lower()</u>	Converts a string into lower case
<u>lstrip()</u>	Returns a left trim version of the string
<u>maketrans()</u>	Returns a translation table to be used in translations
<u>partition()</u>	Returns a tuple where the string is parted into three parts
<u>replace()</u>	Returns a string where a specified value is replaced with a specified value
<u>rfind()</u>	Searches the string for a specified value and returns the last position of where it was found
<u>rindex()</u>	Searches the string for a specified value and returns the last position of where it was found
<u>rjust()</u>	Returns a right justified version of the string
<u>rpartition()</u>	Returns a tuple where the string is parted into three parts
<u>rsplit()</u>	Splits the string at the specified separator, and returns a list
<u>rstrip()</u>	Returns a right trim version of the string
<u>split()</u>	Splits the string at the specified separator, and returns a list
<u>splitlines()</u>	Splits the string at line breaks and returns a list
<u>startswith()</u>	Returns true if the string starts with the specified value
<u>strip()</u>	Returns a trimmed version of the string
<u>swapcase()</u>	Swaps cases, lower case becomes upper case and vice versa
<u>title()</u>	Converts the first character of each word to upper case
<u>translate()</u>	Returns a translated string
<u>upper()</u>	Converts a string into upper case
<u>zfill()</u>	Fills the string with a specified number of 0 values at the beginning
<u>rpartition()</u>	Returns a tuple where the string is parted into three parts
<u>rsplit()</u>	Splits the string at the specified separator, and returns a list
<u>rstrip()</u>	Returns a right trim version of the string
<u>split()</u>	Splits the string at the specified separator, and returns a list
<u>splitlines()</u>	Splits the string at line breaks and returns a list
<u>startswith()</u>	Returns true if the string starts with the specified value
<u>strip()</u>	Returns a trimmed version of the string
<u>swapcase()</u>	Swaps cases, lower case becomes upper case and vice versa
<u>title()</u>	Converts the first character of each word to upper case
<u>translate()</u>	Returns a translated string
<u>upper()</u>	Converts a string into upper case
<u>zfill()</u>	Fills the string with a specified number of 0 values at the beginning



10. บูลีน (Python Booleans)

10.1 ค่าบูลีน (Boolean Values)

ในการเขียนโปรแกรมมักจะต้องทราบว่าการแสดงออกเป็น True หรือ False สามารถประเมิน และได้รับหนึ่งในสองคำตอบ True หรือ False เมื่อเปรียบเทียบสองค่านิพจน์จะถูกประเมินค่าและไฟฟอน จะส่งคืนคำตอบบูลีน:



ตัวอย่างที่ 2.47 ค่าบูลีน

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

output

True
False
False

เมื่อเรียกใช้เงื่อนไขในคำสั่ง if ไฟฟอน จะคืนค่า True หรือ False:



ตัวอย่างที่ 2.48 พิมพ์ข้อความตามเงื่อนไข True หรือ False:

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

output

b is not greater than a

10.2 ประเมินค่าและตัวแปร (Evaluate Values and Variables)

ฟังก์ชัน bool() ช่วยให้สามารถประเมินค่าได ๆ และคืนค่า True หรือ False



ตัวอย่างที่ 2.49 หากค่าสตริงและตัวเลข:

```
print(bool("Hello"))
print(bool(15))
```

output

True

True



ตัวอย่างที่ 2.51 ประเมินตัวแปรสองตัว:

```
x = "Hello"
```

```
y = 15
```

```
print(bool(x))
```

```
print(bool(y))
```

output

True

True

10.3 ค่าส่วนใหญ่เป็นจริง(Most Values are True)

- เกือบทุกค่าจะถูกประเมิน True ว่ามีเนื้อหาประเภทใด
- สตริงใด ๆ True ยกเว้นสตริงว่าง
- จำนวนใด ๆ ที่เป็นยกเว้น True 0
- รายการ tuple set และ dict มีค่าเป็น True ยกเว้นรายการที่ว่างเปล่า



ตัวอย่างที่ 2.52 ส่งคืนค่า True:

```
bool("abc")
```

```
bool(123)
```

```
bool(["apple", "cherry", "banana"])
```

output

True

True

True

10.4 ค่าบางอย่างเป็นเท็จ (Some Values are False)

ในความเป็นจริงแล้ว มีค่าไม่มากที่ถูกประเมินให้เป็น False ยกเว้นค่าที่ว่างเปล่า เช่น (), [], {}, "" จำนวน 0, None และแน่นอนค่าที่ประเมินต้องเป็น False



ตัวอย่างที่ 2.42 ส่งคืนค่า False

```
bool(False)  
bool(None)  
bool(0)  
bool("")  
bool(())  
bool([])  
bool({})
```

output

```
False  
False  
False  
False  
False  
False  
False
```

ค่าอื่นๆ หรือ Object ในกรณี จะถูกประเมินให้มีค่า False และถ้า Object จาก Class ที่เป็นฟังก์ชัน `_len_` ให้ส่งกลับค่า 0 หรือ False:

ตัวอย่างที่ 2.53 ฟังก์ชัน `_len_`

```
class myclass():  
    def __len__(self):  
        return 0  
  
myobj = myclass()  
print(bool(myobj))
```

output

```
False
```

10.5 พิมพ์ชื่อสามารถคืนค่าบูลีน

ไพทอน มีพิมพ์ชื่อในตัววายด้วยตัวที่ส่งคืนค่าบูลีน เช่น `isinstance()` พิมพ์ชื่อสามารถใช้เพื่อกำหนดว่าออบเจ็กมีชนิดข้อมูลชนิดใด :



ตัวอย่างที่ 2.54 ตรวจสอบว่าวัตถุเป็นจำนวนเต็มหรือไม่:

```
x = 200
print(isinstance(x, int))
```

output

True



ในบทนี้จะกล่าวถึงพื้นฐานการเขียนโปรแกรมภาษาไพทอน ที่มีองค์ประกอบและข้อกำหนดที่สำคัญ ที่เราควรรู้จักในเบื้องต้นที่จะกล่าวถึงในบทนี้ เช่น การเขียนและรันโปรแกรมไฟทอน การใช้งานคำสั่งพื้นฐานที่สำคัญ รูปแบบการเขียนโปรแกรม คำอธิบายโปรแกรม การแสดงผลตัวแปร ชนิดข้อมูล และบูลีน



12. แบบฝึกหัดท้ายบท

- ให้ใช้พิมพ์ชื่อ `print()` เพื่อนำตัวอักษร * มาสร้างเป็นรูปสามเหลี่ยมกลับหัว
- ให้ใช้พิมพ์ชื่อ `print()` เพื่อนำตัวเลข 1-5 มาสร้างเป็นรูปสามเหลี่ยม
- จะเขียนโปรแกรมเพื่อรับข้อมูลของนักศึกษา ได้แก่ ชื่อ นามสกุล สาขาวิชา คณะ มหาวิทยาลัย จากคีย์บอร์ด และแสดงผลลัพธ์

บทที่ 3

ตัวดำเนินการ

ตัวดำเนินการ (Operators) คือ กลุ่มของเครื่องหมาย หรือ สัญลักษณ์ที่ใช้ทำงานเมื่อกับฟังก์ชัน แต่แตกต่างกันตรงไวยากรณ์ หรือความหมายในการใช้งานในภาษาไพทอน นั้นสนับสนุนตัวดำเนินการประเภทต่างๆ สำหรับการเขียนโปรแกรม เช่น ตัวดำเนินการ + เป็นตัวดำเนินการทางคณิตศาสตร์ที่ใช้สำหรับการบวกตัวเลขเข้าด้วยกัน หรือตัวดำเนินการ > เป็นตัวดำเนินการเพื่อให้เปรียบเทียบค่าสองค่า

ตัวดำเนินการในไพทอน แบ่งตามกลุ่มการทำงานได้ดังนี้ :

- Assignment operators
- Arithmetic operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

และจะบញ្ជីថា ฟังก์ชัน哪些 กับตัวเลข(Python Math)

1. ตัวดำเนินการกำหนดค่า (Python Assignment Operators)

ตัวดำเนินการที่ใช้ในการกำหนดค่าให้กับตัวแปร:

ตารางที่ 8 แสดงตัวดำเนินการที่ใช้ในการกำหนดค่าให้กับตัวแปร

ตัวดำเนินการ	คำอธิบาย	ตัวอย่าง	Same As
=	กำหนดค่า	x = 5	x = 5
+=	บวกก่อนแล้วกำหนดค่า	x += 3	x = x + 3
-=	ลบก่อนแล้วกำหนดค่า	x -= 3	x = x - 3
*=	คูณก่อนแล้วกำหนดค่า	x *= 3	x = x * 3
/=	หารก่อนแล้วกำหนดค่า	x /= 3	x = x / 3
%=	หารเอาเศษแล้วกำหนดค่า	x %= 3	x = x % 3
//=	หารเอาส่วนแล้วกำหนดค่า	x //= 3	x = x // 3
**=	ยกกำลังแล้วกำหนดค่า	x **= 3	x = x ** 3



ตัวอย่างที่ 3.1 โค้ดคำสั่งตัวดำเนินการกำหนดค่า

```
a = 3
b = 5.29
c = b
name = 'Computer EDU KSU'
my_list = [2, 5, 8, 10, 24]
x, y = 10, 20
```

ในตัวอย่าง เป็นการใช้งานตัวดำเนินการกำหนดค่าสำหรับกำหนดค่าให้กับตัวแปรประเภทต่างๆ โดยทั่วไปแล้ว ตัวดำเนินการกำหนดค่านั้นเกือบจะใช้ในทุกๆ ที่ในโปรแกรมและเป็นตัวดำเนินการที่ใช้บ่อยที่สุดของในบรรดาตัวดำเนินการทั้งหมด

2. ตัวดำเนินการคณิตศาสตร์ (Python Arithmetic Operators)

ตารางที่ 9 ตัวดำเนินการทางคณิตศาสตร์

ตัวดำเนินการ	คำอธิบาย	ตัวอย่าง	ผลลัพธ์($x=5, y=3$)
+	บวก(Addition)	$z=x + y$	$z = 8$
-	ลบ(Subtraction)	$z=x - y$	$z = 2$
*	คูณ(Multiplication)	$z=x * y$	$z = 15$
/	หารเอาผลลัพธ์(Division)	$z=x / y$	$z = 1.6666666666666667$
%	หารเอาเศษ(Modulus)	$z=x \% y$	$z = 2$
//	หารปัดเศษผลลัพธ์ลงเป็นจำนวนเต็มที่ใกล้ที่สุด(Floor division)	$z=x // y$	$z = 1$
**	เลขยกกำลัง (Exponentiation)	$z=x ** y$	$z = 125$



ตัวอย่างที่ 3.2 การนำตัวดำเนินการทางคณิตศาสตร์มาใช้คำนวณหาผลลัพธ์

```
a = 5
b = 3
print("a + b = ", a + b)
print("a - b = ", a - b)
print("a * b = ", a * b)
print("a / b = ", a / b)
print("a // b = ", a // b) # floor number to integer
print("a % b = ", a % b) # get division remainder
```



```
print("a ** b = ", a ** b) # power
```

Output

```
a + b =  8  
a - b =  2  
a * b =  15  
a / b =  1.666666666666667  
a // b =  1  
a % b =  2  
a ** b =  125
```

3. ตัวดำเนินการเปรียบเทียบ (Python Comparison Operators)

ตัวดำเนินการเปรียบเทียบ (Comparison operators) คือตัวดำเนินการที่ใช้สำหรับเปรียบเทียบค่าหรือค่าในตัวแปร ซึ่งผลลัพธ์ของการเปรียบเทียบนั้นจะเป็น True หากเงื่อนไขเป็นจริง และเป็น False หากเงื่อนไขไม่เป็นจริง ตัวดำเนินการเปรียบเทียบมักจะใช้กับคำสั่งตรวจสอบเงื่อนไข if และคำสั่งวนซ้ำ for while เพื่อควบคุมการทำงานของโปรแกรม

ตารางที่ 10 ตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการ	คำอธิบาย	ตัวอย่าง
==	เท่ากับ	x == y
!=	ไม่เท่ากับ	x != y
>	มากกว่า	x > y
<	น้อย	x < y
>=	มากกว่าหรือเท่ากับ	x >= y
<=	น้อยกว่าหรือเท่ากับ	x <= y



ตัวอย่างที่ 3.3 โค้ดคำสั่งตัวดำเนินการเปรียบเทียบ

```
# Constant comparison  
print('4 == 4:', 4 == 4)  
print('1 < 2:', 1 < 2)  
print('3 > 10:', 3 > 10)  
print('2 <= 1.5', 2 <= 1.5)  
print()  
  
# Variable comparison  
a = 10  
b = 8
```



```
print('a != b:', a != b)
print('a - b == 2:', a - b == 2)
print()
```

output

```
4 == 4 : True
1 < 2: True
3 > 10: False
2 <= 1.5 False
```

```
a != b: True
a - b == 2: True
```

4. ตัวดำเนินการตรรกศาสตร์ (Python Logical Operators)

ตัวดำเนินการตรรกศาสตร์ (Logical operators) คือตัวดำเนินการที่ใช้สำหรับประเมินค่าทางตรรกศาสตร์ ซึ่งเป็นค่าที่มีเพียงจริง (True) และเท็จ (False) เท่านั้น โดยทั่วไปแล้วเรามักใช้ตัวดำเนินการตรรกศาสตร์ในการเชื่อม Boolean expression ตั้งแต่หนึ่ง expression ขึ้นไปและผลลัพธ์สุดท้ายที่ได้นั้นจะเป็น Boolean ตัวดำเนินการเชิงตรรกะมักถูกใช้เพื่อรวมคำสั่งแบบมีเงื่อนไข:

ตารางที่ 11 ตัวดำเนินการตรรกศาสตร์

ตัวดำเนินการ	คำอธิบาย	ตัวอย่าง
and	และ: Returns True if both statements are true	<code>x < 5 and x < 10</code>
or	หรือ: Returns True if one of the statements is true	<code>x < 5 or x < 4</code>
not	ไม่: Reverse the result, returns False if the result is true	<code>not(x < 5 and x < 10)</code>



ตัวอย่างที่ 3.4 โค้ดคำสั่งตัวดำเนินการตรรกศาสตร์

```
a = 3 # 00000011
b = 5 # 00000101

print('a & b =', a & b)
print('a | b =', a | b)
print('a ^ b =', a ^ b)
print('~a =', ~a)
```



```
print('a << 1 =', a << 1)
print('a << 2 =', a << 2)
print('100 >> 1 =', 100 >> 1)
```

output

```
a & b = 1
a | b = 7
a ^ b = 6
~a = -4
a << 1 = 6
a << 2 = 12
100 >> 1 = 50
```

ในตัวอย่าง เป็นการใช้ตัวดำเนินการระดับบิตประเพณฑ์ต่างๆ ในภาษาไพทอน เราเมื่อตัวแปร a และตัวแปร b และกำหนดค่า 3 และ 5 ให้กับตัวแปรตามลำดับ เราได้คอมเมนต์ค่าในฐานสองไว้ด้วย ในการทำงานนั้นโปรแกรมจะทำงานทีละคู่ของบิต ดูวิธีการคำนวณต่อไปนี้ประกอบ

```
# 3 & 5
00000011
00000101
00000001 # result = 1

# 3 | 5
00000011
00000101
00000111 # result = 7

# 3 ^ 5
00000011
00000101
00000110 # result = 6

# ~3
00000011
11111100 # result = -4
```

จากการแสดงการทำงานข้างบนนี้ ในตัวดำเนินการ & หากทั้งสองบิตมีค่าเป็น 1 จะได้ผลลัพธ์เป็น 1 ไม่เช่นนั้น 0 ในตัวดำเนินการ | หากอย่างน้อยหนึ่งบิตที่มีค่าเป็น 1 จะได้ผลลัพธ์เป็น 1 ไม่เช่นนั้น 0 ในตัวดำเนินการ ^ หากทั้งสองบิตนั้นแตกต่างกันจะได้ผลลัพธ์เป็น 1 ไม่เช่นนั้น 0 และในตัวดำเนินการ ~ นั้นเป็นการกลับค่าของบิต หลังจากนั้นเราแบ่งผลลัพธ์ที่ได้กลับไปยังฐานสิบ

```
# 3 << 1
00000011
00000110 # result = 6

# 3 << 2
00000011
00001100 # result = 12

# 100>> 1
1100100
0110010 # result = 50
```

อีกสามคำสั่งต่อมาเป็นการใช้งานตัวดำเนินการเลื่อนบิต ในการทำงานนี้จะเป็นการเลื่อนบิตไปทางซ้ายหรือขวาตามทิศทางของลูกศรของตัวดำเนินการ บิตที่เข้ามาใหม่ทางด้านซ้ายหรือขวา นั้นเป็นบิต 0 เสมอ โดยทั่วไปแล้ว เมื่อเราเลื่อนบิตของตัวเลขใดๆ ไปทางด้านซ้ายหนึ่งครั้งจะทำให้ค่าเพิ่มขึ้นสองเท่า และถ้าหากเลื่อนไปทางด้านขวาหนึ่งครั้งจะทำให้ค่าลดลงครึ่งหนึ่ง

5. ตัวดำเนินการเอกลักษณ์ (Python Identity Operators)

Identity operators จะใช้ในการเปรียบเทียบข้อมูลระหว่างสองตัวแปร(objects) คือ `is` และ `is not` ว่าเหมือนหรือไม่เหมือนกัน :

ตารางที่ 12 ตัวดำเนินการเอกลักษณ์

Operator	Description	Example
<code>is</code>	Returns true if both variables are the same object	<code>x is y</code>
<code>is not</code>	Returns true if both variables are not the same object	<code>x is not y</code>



ตัวอย่างที่ 3.5 การใช้ `is`

```
x = ["apple", "banana"]
y = ["apple", "banana"]
```



```
z = x
print(x is z)

# returns True because z is the same object as x
print(x is y)

# returns False because x is not the same object as y, even if they have the same
content
print(x == y)

# to demonstrate the difference between "is" and "==": this comparison returns
True because x is equal to y
```

output

```
True
False
True
```



ตัวอย่างที่ 3.6 การใช้ is not

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is not z)
# คืนค่า False เพราะว่า z เท่ากับ x

print(x is not y)
# คืนค่าTrue เพราะ x ไม่เท่ากับ y, แม้ว่าจะมีเนื้อหาเหมือนกัน

print(x != y)
# เพื่อแสดงความแตกต่างระหว่าง "is not" and "!=": เปรียบเทียบส่งคืนค่า False เพราะ x
# เท่ากับ y
```

output

```
False
True
False
```

6. ตัวดำเนินการตรวจสอบสมาชิก (Python Membership Operators)

ในภาษาไพทอน มีตัวดำเนินการในการตรวจสอบการเป็นสมาชิกในออบเจ็คประเภท List Tuple และ Dictionary ตัวดำเนินการ in ใช้ในการตรวจสอบถ้าหากค่านั้นมีอยู่ในออบเจ็ค ถ้าหากพบจะได้ผลลัพธ์เป็น True และหากไม่พบจะได้ผลลัพธ์เป็น False และตัวดำเนินการ not in นั้นจะทำงานตรงกันข้าม หากไม่พบจะได้ผลลัพธ์เป็น True แทน

ตารางที่ 13 ตัวดำเนินการในการตรวจสอบการเป็นสมาชิกในออบเจ็ค

Operator	Description	Example
in	Object memberships	x in y
not in	Negated object memberships	x not in y



ตัวอย่างที่ 3.7 การใช้ in

```
# ส่งคืนค่า True เพราะลำดับค่า "banana" อยู่ใน list
x = ["apple", "banana"]

print("banana" in x)
```

output

True



ตัวอย่างที่ 3.8 การใช้ not in

```
# ส่งคืนค่า True เพราะลำดับค่า "pineapple" ไม่อยู่ใน list
x = ["apple", "banana"]
print("pineapple" not in x)
```

output

True



ตัวอย่างที่ 3.9 คำสั่งตรวจสอบเงื่อนไข

```
if 'Prim' in names:
    print('\'Prim\' exist in the list')
else:
    print('\'Prim\' not exist in the list')
```



```
if 'Jib' in names:  
    print('\'Jib\' exist in the list')  
else:  
    print('\'Jib\' not exist in the list')  
  
numbers = {'1': 'one', '3': 'three', '2': 'two', '5': 'five'}  
if 'one' in numbers.values():  
    print('\'one\' exist in the dictionary values')  
else:  
    print('\'one\' not exist in the dictionary values')  
  
if '7' in numbers.keys():  
    print('\'7\' exist in the dictionary keys')  
else:  
    print('\'7\' not exist in the dictionary keys')
```

output

```
'Prim' exist in the list  
'Jib' not exist in the list  
'one' exist in the dictionary values  
'7' not exist in the dictionary keys
```

7. ตัวดำเนินการระดับบิต (Python Bitwise Operators)

ตัวดำเนินการระดับบิต (Bitwise operators) เป็นตัวดำเนินการที่ทำงานในระดับบิตของข้อมูล หรือจัดการข้อมูลในระบบเลขฐานสอง โดยทั่วไปแล้วตัวดำเนินการระดับบิตมักจะใช้กับการเขียนโปรแกรมระดับต่ำ เช่น การเขียนโปรแกรมเพื่อควบคุมฮาร์ดแวร์ อย่างไรก็ตาม ในภาษาไพทอนนั้นสนับสนุนตัวดำเนินการเพื่อให้สามารถจัดการกับบิตของข้อมูลโดยตรงได้

ตารางที่ 14 ตัวดำเนินการระดับบิตในภาษาไพทอน

ตัวดำเนินการ	ชื่อ	คำอธิบาย
&	AND	กำหนดค่าบิตเป็น 1 ถ้าทั้งสองบิตมีค่าเป็น 1
	OR	กำหนดค่าบิตเป็น 1 ถ้าบิตตัวใดตัวหนึ่งมีค่าเป็น 1
^	XOR	กำหนดค่าบิตเป็น 1 ถ้าตัวใดตัวหนึ่งหรือทั้งสองตัวมีค่าเป็น 1
~	NOT	กลับบิต
<<	Zero fill left shift	เลื่อนไปทางซ้ายโดยเก็บศูนย์จากด้านขวาแล้วปล่อยให้บิตซ้ายสุดหลุดออกมา
>>	Signed right shift	เลื่อนไปทางขวาโดยเก็บสำเนาของบิตซ้ายสุดเข้าจากซ้าย แล้วปล่อยให้บิตขวาสุดหลุดออกมา



ตัวอย่างที่ 3.10 คำสั่งตัวดำเนินการระดับบิต

```
a = 3 # 00000011
b = 5 # 00000101

print('a & b =', a & b)
print('a | b =', a | b)
print('a ^ b =', a ^ b)
print('~a =', ~a)
print('a << 1 =', a << 1)
print('a << 2 =', a << 2)
print('100 >> 1 =', 100 >> 1)
```

Output

```
a & b = 1
a | b = 7
a ^ b = 6
~a = -4
a << 1 = 6
a << 2 = 12
100 >> 1 = 50
```

8. ลำดับการประมวลผลของตัวดำเนินการ

ตารางที่ 15 ลำดับการประมวลผลของตัวดำเนินการ

ลำดับ	ตัวดำเนินการ	คำอธิบาย
1	()	ใช้เพื่อแบ่งนิพจน์ และลำดับการทำงาน
2	**	ยกกำลัง
3	~, +, -	คอมพลีเม้น, unary plus หรือ unary minus (การดำเนินการทางคณิตศาสตร์โดยมีการกระทำกับ operand เพียงตัวเดียว เพื่อทำการเพิ่มค่า / ลดค่า)
4	*, /, %, //	คูณ, หาร, หารเอาเศษ, หารเอาส่วน
5	+, -	บวก, ลบ
6	=, +=, -=, *=, /=, %=, //=, **=	ตัวดำเนินการกำหนดค่า



หมายเหตุ นิพจน์เริ่มจากซ้ายไปขวาเสมอ



9. พัฟ์ชันเกี่ยวกับตัวเลข (Python Math)

ไพทอน มีชุดพัฟ์ชันเกี่ยวกับตัวเลขให้เลือกใช้มากมาย แยกเป็น 2 กลุ่ม คือ Built-in Math Functions สามารถเรียกใช้งานได้ทันที และ The Math Module เรียกใช้งานผ่านโมดูล math ดังนี้

▪ Built-in Math Functions

`min()` และ `max()` พัฟ์ชันที่สามารถใช้ในการหาค่าต่ำสุดหรือสูงสุดใน iterable:



ตัวอย่างที่ 3.11 คำสั่งพัฟ์ชันที่สามารถใช้ในการหาค่าต่ำสุดหรือสูงสุด

```
x = min(5, 10, 25)  
y = max(5, 10, 25)  
  
print(x)  
print(y)
```

output

```
5  
25
```

`abs()`พัฟ์ชันส่งกลับແน่นอนค่า (บวก) ของจำนวนที่ระบุ:



ตัวอย่างที่ 3.12 `abs()`พัฟ์ชัน

```
x = abs(-7.25)  
  
print(x)  
  
output  
7.25
```

พัฟ์ชันส่งกลับค่าของ x ยกกำลัง Y (x^Y)`pow(x, y)`



ตัวอย่างที่ 3.13 คืนค่า 4 เป็นยกกำลัง 3 (เช่นเดียวกับ $4 * 4 * 4$):

```
x = pow(4, 3)  
  
print(x)  
  
output  
64
```

■ The Math Module

ไฟฟอน ยังมีโมดูลในตัวที่เรียกว่า math ซึ่งขยายรายการฟังก์ชันทางคณิตศาสตร์ ในการใช้งานต้องนำเข้าmathโมดูล:

```
import math
```

เมื่อ import math โมดูล สามารถเริ่มใช้วิธีการและค่าคงที่ของโมดูลได้ เมธอด math.sqrt() ตัวอย่างคืนค่า square root ของจำนวน:



ตัวอย่างที่ 3.14 import math

```
import math
```

```
x = math.sqrt(64)
```

```
print(x)
```

output

8.0

เมธอด math.ceil() ปัดเศษตัวเลขขึ้นไปเต็มที่ใกล้เคียงที่สุด และ เมธอด math.floor() เลขจำนวนเต็มที่ใกล้เคียงที่สุด:



ตัวอย่างที่ 3.15 เมธอด math.ceil()

```
import math
```

```
x = math.ceil(1.4)
```

```
y = math.floor(1.4)
```

```
print(x) # returns 2
```

```
print(y) # returns 1
```

output

2

1



ตัวอย่างที่ 3.16 math.pi คงสั่งกลับค่าของ PI (3.14 ...)

```
import math

x = math.pi

print(x)
```

output

3.141592653589793

10. กิจกรรม



1. จงฝึกเขียนโปรแกรมและหาผลลัพธ์ สลับค่าก่อนหลัง

```
a = 7
b = 11
print('ก่อนสลับค่า a = {a}, b = {b}')

c = a
a = b
b = c
print('หลังสลับค่า a = {a}, b = {b}')
```

```
ก่อนสลับค่า a = 7, b = 11
หลังสลับค่า a = 11, b = 7
Process finished with exit code 0
```

จงฝึกเขียนโปรแกรมและหาผลลัพธ์ จำนวนและราคางานค้าที่ซื้อ

```
quantity = int(input('จำนวนสินค้าที่ซื้อ: '))
#จำนวนสินค้าต้องเป็นจำนวนเต็ม
price = float(input('ราคาสินค้า: '))
#ราคาสินค้าอาจมีทศนิยมได้

total = price * quantity
print('รวมเป็นเงิน: ', total)

pay = float(input('จำนวนเงินที่จ่าย: '))
change = pay - total
print('เงินทอน: ', change)
```

```
จำนวนสินค้าที่ซื้อ: 12
ราคาสินค้า: 22
รวมเป็นเงิน: 264.0
จำนวนเงินที่จ่าย: |
```

2. จะฝึกเขียนโปรแกรมและหาผลลัพธ์ สลับค่า

```
a = 10
b = 20
print(f'ก่อนสลับค่า a = {a}, b = {b}')

b = a + b      #b = 10 + 20 = 30
a = b - a      #a = 30 - 10 = 20
b = b - a      #b = 30 - 20 = 10
print(f'หลังสลับค่า a = {a}, b = {b}\n')

a = 108
b = -1009
print(f'ก่อนสลับค่า a = {a}, b = {b}')

b = a + b
a = b - a
b = b - a
print(f'หลังสลับค่า a = {a}, b = {b}')
```

```
ก่อนสลับค่า a = 10, b = 20
หลังสลับค่า a = 20, b = 10

ก่อนสลับค่า a = 108, b = -1009
หลังสลับค่า a = -1009, b = 108

Process finished with exit code 0
```

3. จะฝึกเขียนโปรแกรมและหาผลลัพธ์ จำนวนเงินที่จะถอนและรับบัตร

```
withdraw = int(input('จำนวนเงินที่จะถอน: '))

b1000 = withdraw // 1000
remain = withdraw % 1000

b500 = remain // 500
remain = remain % 500

b100 = remain // 100
remain = remain % 100

print(f'รับบัตรที่ได้:\nB1000 = {b1000} \nB500 = {b500} \nB100 = {b100}')
```

```
จำนวนเงินที่จะถอน: 500
รับบัตรที่ได้:
B1000 = 0
B500 = 1
B100 = 0

Process finished with exit code 0
```



4. จงฝึกเขียนโปรแกรมและหาผลลัพธ์ ค่าเฉลี่ย

```
sum = 0

sum += int(input('จำนวนที่ #1: '))
sum += int(input('จำนวนที่ #2: '))
sum += int(input('จำนวนที่ #3: '))
sum += int(input('จำนวนที่ #4: '))

average = sum / 4

print('ผลรวมเท่ากับ: ', sum)
print('ค่าเฉลี่ยเท่ากับ: ', average)
```

```
จำนวนที่ #1: 10
จำนวนที่ #2: 20
จำนวนที่ #3: 30
จำนวนที่ #4: 40

ผลรวมเท่ากับ: 100
ค่าเฉลี่ยเท่ากับ: 25.0

Process finished with exit code 0
```

5. จงฝึกเขียนโปรแกรมและหาผลลัพธ์ การรับค่า

```
n = int(input('กรุณาใส่เลข 4 หลัก: '))

x = n
x //= 1000
print(x)

x = n
x %= 1000
x //= 100
print(x)

x = n
x %= 100
x //= 10
print(x)

x = n
x %= 10
print(x)
```

```
กรุณาใส่เลข 4 หลัก: 1234
1
2
3
4

Process finished with exit code 0
```

11. สรุปท้ายบท



บทนี้นักศึกษาได้เรียนรู้เกี่ยวกับโครงสร้างของภาษาไพทอน ตัวแปรภาษาในไพทอน ได้รู้จักชนิดข้อมูล การแปลงชนิดข้อมูล นิพจน์ รวมไปถึงตัวดำเนินการประเภทต่างๆ ซึ่งจะช่วยให้เขียนโปรแกรมภาษาไพทอน ได้อย่างถูกต้อง

12. แบบฝึกหัดท้ายบท



1. แปลงค่าปีพุทธศักราชให้เป็นปีคริสต์ศักราช
2. หาปริมาตรของน้ำในตู้ปลาทรงสี่เหลี่ยมมุมฉาก เมื่อทราบความกว้าง ความยาว และ ความสูง
3. หาค่าเฉลี่ยและร้อยละของคะแนนสอบ 5 วิชา จากคะแนนเต็ม 100 คะแนน
4. ลุงถิตเริ่มวิ่งจากจุดต้นทางทิศตะวันออก x กิโลเมตร และวิ่งไปทางทิศเหลืออีก y กิโลเมตร ลุงถิตอยู่ห่างจากจุดเริ่มต้นเท่าใด
5. ป้าหญิงต้องการไปเที่ยวแอฟริกาใต้ พยากรณ์อากาศแจ้งว่า อุณหภูมิช่วงที่ไป 55 องศา ฟาเรนไฮต์ จะเทียบเท่ากี่องศาเซลเซียส
6. จงเขียนโปรแกรมเพื่อคำนวณหาพื้นที่รูปพื้นที่วงกลม ที่มีความยาวแต่ละด้านมุมเท่ากัน ทั้งหมด มีสูตรคำนวณดังนี้

$$\text{พื้นที่วงกลม} = \pi r^2$$

 * π มีค่าเท่ากับ $22/7$ หรือ 3.14
7. จงเขียนโปรแกรมเพื่อรับค่าจากคีย์บอร์ด เพื่อคำนวณหามูลค่าการลงทุนในอนาคต (future value)

$$fv = amount * (1 * rate)^{\text{period}}$$

- Amount = จำนวนเงินที่ลงทุนเริ่มแรก
- Rate = อัตราดอกเบี้ย
- Period = ช่วงระยะเวลาการลงทุน

```
จำนวนเงินที่ลงทุนเริ่มแรก >>100000
จำนวนปีที่ลงทุน >>10
อัตราดอกเบี้ยต่อปี (%) >>5

มูลค่าในอนาคตของเงินที่ลงทุนคือ: 162,889.46

Process finished with exit code 0
```

8. จงเขียนโปรแกรมเพื่อสร้าง Password แบบสุ่ม ที่ ประกอบด้วย 0-9,a-z,A-Z รวมกัน 6 ตัว

```
Your password is: XK694E

Process finished with exit code 0
```



9. จงเขียนโปรแกรมเพื่อสร้างเลขสุ่มให้เป็นวินาที ระหว่าง 100,000 -1,000,000 แล้ว
แปลงหน่วยเป็น ชั่วโมง:นาที:วินาที

```
เวลา 83,612 วินาที เท่ากับ 23 ชั่วโมง 13 นาที 32 วินาที
```

```
Process finished with exit code 0
```

10. จงเขียนโปรแกรมเพื่อรับตัวเลขจากคีย์บอร์ด ซึ่งสมมติว่าเป็นเงินที่ต้องจ่ายออก
แล้วตรวจสอบว่าเงินดังกล่าวต้องประกอบด้วยธนบัตรและเหรียญชนิดใด โดยให้เริ่มจากค่าที่มาก
ที่สุดที่เป็นไปได้เสมอ คือ 1000,500,100,50,20,10,5,2,1,50สตางค์,25สตางค์

```
จำนวนเงินที่ต้องจ่าย >>1999
```

```
B1000 = 1
B500 = 1
B100 = 4
B50 = 1
B20 = 2
B10 = 9
B5 = 1
B2 = 2
B1 = 0
B.50 = 0
B.25 = 0
```

```
Process finished with exit code 0
```

บทที่ 4

ข้อมูลแบบลำดับรายการ

หากมีตัวแปรที่จัดเก็บข้อมูลจำนวนมาก การอ้างถึงตัวแปรทีละตัวก็จะเกิดความยุ่งยาก ซึ่งเราอาจเปลี่ยนมาใช้วิธีการสร้างที่สามารถเก็บชุดข้อมูลแทนได้หลายแบบ เช่น Lists, Tuples, Sets, หรือ Dictionaries เป็นต้น ซึ่งข้อมูลดังกล่าวมีลักษณะเป็นลำดับรายการที่ต่อเนื่องกัน ดังนั้น เราสามารถใช้ลูป for หรือ while รวมถึงฟังก์ชันที่เกี่ยวข้อง

1. รายการข้อมูลแบบ Lists

List (ลิสต์) คือ โครงสร้างข้อมูลชนิดหนึ่งในภาษาไพทอนที่ใช้เก็บข้อมูลแบบลำดับ (Sequence) โดยมี Index เป็นตัวระบุตำแหน่งในการเข้าถึงข้อมูล เราสามารถใช้ List เพื่อเก็บข้อมูลจำนวนมาก และหลากหลายประเภทในเวลาเดียวกัน List เป็นประเภทข้อมูลที่ใช้อย่างหลากหลายในการเขียนโปรแกรม นอกจากนี้ ในภาษาไพทอนยังมี built-in function ที่สามารถทำงานกับ List และใน List ออกแบบเองก็ไม่ยากต่างๆ เป็นจำนวนมากที่ช่วยอำนวยความสะดวกในการเขียนโปรแกรม

1.1 การประกาศและใช้งาน List

List นั้นเป็นตัวแปรประเภทหนึ่ง การใช้งานของมันจะเหมือนกับอาเรย์ในภาษาอื่นๆ ในการประกาศ List นั้นข้อมูลของมันจะอยู่ภายใต้เครื่องหมาย [...] และค้นสมาชิกแต่ละตัวด้วยเครื่องหมาย commas , มาดูตัวอย่างการประกาศ List ในภาษาไพทอน



ตัวอย่างที่

```
numbers = [-1, 2, 5, 8, 10, 13]
names = ['Sit', 'Jame', 'Ton', 'Leo', 'Ang', 'Pusit', 'Kawin', 'Prim']
mixed_type = [-2, 5, 84.2, "C++", "Python"]
```

1.2 เมรอดและฟังก์ชันที่ใช้กับ List

ตารางที่ 16 เมรอดและฟังก์ชันที่ใช้กับ List

เมรอด	ความหมาย	รูปแบบการใช้งาน
append()	เพิ่มข้อมูลต่อท้ายลิสต์	lst.append(x) x คือ ข้อมูลที่ต้องการเพิ่ม
clear()	ลบข้อมูลทั้งหมดออกจากลิสต์	lst.clear()
copy()	คัดลอกชนิดข้อมูลลิสต์	lst_new = lst.copy() lst_new = ชื่อตัวแปรเก็บข้อมูลลิสต์ lst.copy = ชื่อตัวแปรชนิดข้อมูลลิสต์ชื่อตัวแปรเก็บข้อมูลลิสต์



เมธอด	ความหมาย	รูปแบบการใช้งาน
<code>count()</code>	นับจำนวนข้อมูลที่ซ้ำกันในลิสต์	<code>lst.count(x)</code> x คือ ข้อมูลที่ต้องการนับ
<code>extend()</code>	เพิ่มลิสต์เข้าไปในลิสต์	<code>lst.extend(x)</code>
<code>index()</code>	ค้นหาตำแหน่งของข้อมูลที่เก็บในลิสต์	<code>lst.index(x, start, stop)</code> x คือ ข้อมูลที่ต้องการค้นหา start คือ จุดเริ่มต้นที่ต้องการค้นหา stop คือ จุดสุดท้ายที่ต้องการค้นหา
<code>insert()</code>	แทรกข้อมูลเข้าไปในลิสต์	<code>lst.insert(index, x)</code> index คือ ตำแหน่งที่ต้องการแทรก x คือ ข้อมูลที่ต้องการแทรก
<code>pop()</code>	ลบข้อมูลโดยการระบุตำแหน่ง	<code>lst.pop(index)</code> index คือ ตำแหน่งที่ต้องการลบ
<code>remove()</code>	ลบข้อมูลด้วยการระบุชื่อ	<code>lst.remove(x)</code> x คือ ข้อมูลที่ต้องการลบ
<code>reverse()</code>	สลับตำแหน่งจากด้านหลังมาด้านหน้า	<code>lst.reverse()</code>
<code>sort()</code>	เรียงลำดับข้อมูลในลิสต์	<code>lst.sort(reverse=False)</code> reverse=False คือ ค่าเริ่มต้นที่จะเรียงจากมากไปหาน้อย กำหนดหรือไม่กำหนดก็ได้ ถ้ากำหนดให้เป็น True จะเรียงจากมากไปหาน้อย
<code>len()</code>	ฟังก์ชันที่ใช้แสดงข้อมูลในลิสต์	<code>len(lst)</code>

หมายเหตุ `lst` คือ ตัวแปรชนิด List

1.3 การเข้าถึงตำแหน่งข้อมูล Lists (Access List Items)

List นั้นใช้ Index สำหรับการเข้าถึงข้อมูล โดย Index ของ List จะเป็นจำนวนเต็มที่เริ่มจาก 0 และเพิ่มขึ้นทีละ 1 ไปเรื่อยๆ ดังนั้น เราจึงสามารถเข้าถึงข้อมูลภายใน List เพื่ออ่านหรืออัพเดตค่าได้โดยตรงผ่าน Index ของมัน นี่เป็นโค้ดการเข้าถึงข้อมูลภายใน List ในภาษาไพทอน

ตำแหน่งจากด้านท้าย	-8	-7	-6	-5	-4	-3	-2	-1
<code>Names =</code>	Sit	Jan	Ton	Leo	Ang	Pu	Kaew	Prim
ตำแหน่งจากด้านหน้า	0	1	2	3	4	5	6	7



ตัวอย่างที่ 4.1 ให้พิมพ์รายการที่ 2 ของ Lists:

```
thislist = ["Sit", "Jan", "Ton", "Leo", "Ang", "Pu", "Kaew", "Prim"]
print(thislist[1])
```

Output

Jan

1.4 การเปลี่ยนค่าของ List (Change List Items)

- การเปลี่ยนค่า Lists



ตัวอย่างที่ 4.2 การเปลี่ยน Lists ที่ 2:

```
thislist = ["Sit", "Jan", "Ton", "Leo", "Ang", "Pu", "Kaew", "Prim"]
thislist[1] = "Dang"
print(thislist[1])
```

Output

Dang

- การเปลี่ยนช่วงค่าของ Lists



ตัวอย่างที่ 4.3 การเปลี่ยนช่วงค่าของ Lists:

```
thislist = ["Sit", "Jan", "Ton", "Leo", "Ang", "Pu", "Kaew", "Prim"]
thislist[1:3] = ["black", "melon"]
print(thislist)
```

Output

['Sit', 'black', 'melon', 'Leo', 'Ang', 'Pu', 'Kaew', 'Prim']



1.5 การเพิ่ม Lists (Add List Items)

▪ การเพิ่ม Lists ด้วยเมธอด append



ตัวอย่างที่ 4.4 เมธอด append() แพร่รายการที่ดัชนีที่ระบุ:

```
thislist = ["Sit", "Jan", "Ton", "Leo", "Ang", "Pu", "Kaew", "Prim"]  
thislist.append("Aey")  
print(thislist)
```

output

```
['Sit', 'Jan', 'Ton', 'Leo', 'Ang', 'Pu', 'Kaew', 'Prim', 'Aey']
```

▪ การแทรก Lists ด้วยเมธอด insert

ในการแทรก Lists ใหม่โดยไม่ต้องแทนที่ค่าใด ๆ ที่มือญี่เราสามารถใช้เมธอด insert()



ตัวอย่างที่ 4.5 เมธอด insert() แพร่รายการที่ดัชนีที่ระบุ:

```
thislist = ["Sit", "Jan", "Ton", "Leo", "Ang", "Pu", "Kaew", "Prim"]  
thislist.insert(2, "Kim")  
print(thislist)
```

output

```
['Sit', 'Jan', 'Kim', 'Ton', 'Leo', 'Ang', 'Pu', 'Kaew', 'Prim']
```

▪ การขยาย List ด้วยเมธอด extend

ในการผนวกองค์ประกอบจากรายการอื่นไปยังรายการปัจจุบันให้ใช้เมธอด extend()



ตัวอย่างที่ 4.6 ใช้เพิ่มรายการ thislist2 ลง thislist :

```
thislist = ["Sit", "Jan", "Ton", "Leo", "Ang", "Pu", "Kaew", "Prim"]  
thislist2 = ["Lin", "Pin", "Fin"]  
thislist.extend(thislist2)  
print(thislist)
```

output

```
['Sit', 'Jan', 'Ton', 'Leo', 'Ang', 'Pu', 'Kaew', 'Prim', 'Lin', 'Pin', 'Fin']
```

1.6 การลบ Lists (Remove List Items)

- ลบรายการที่ระบุด้วยเมธอด `remove()`



ตัวอย่างที่ 4.7 การลบ "banana" :

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

output

```
["apple", "cherry"]
```

- ลบตัวนึงที่ระบุด้วยเมธอด `pop()`



ตัวอย่างที่ 4.8 การลบ List ที่ 2 :

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

output

```
["apple", "cherry"]
```



หมายเหตุ หากไม่ระบุตัวนึง เมธอดจะลบรายการสุดท้าย

- การลบด้วยคำลักษณ์ `Del`



ตัวอย่างที่ 4.9 การลบ List ตัวที่ 0 :

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

output

```
["banana", "cherry"]
```



ตัวอย่างที่ 4.10 การลบ List ทั้งหมด :

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

```
thislist = ["apple", "banana", "cherry"]
del thislist
print(thislist) #this will cause an error because you have successfully deleted
# "thislist".
```

```
Traceback (most recent call last):
  File "demo_list_del2.py", line 3, in <module>
    print(thislist) #this will cause an error because you have successfully deleted "thislist".
NameError: name 'thislist' is not defined
```



▪ การล้าง Lists ด้วย clear()

Lists ยังคงอยู่แต่ไม่มีเนื้อหา



ตัวอย่างที่ 4.11 การล้าง Lists :

```
thislist = ["apple", "banana", "cherry"]  
thislist.clear  
print(thislist)
```

Output

```
[ ]
```

2. รายการข้อมูลแบบทูเพิล (Tuples)

2.1 ทำความรู้จักทูเพิล

- ทูเพิล คือ ชุดของวัตถุที่เรียกลำดับและไม่เปลี่ยนรูป Tuples เป็นลำดับ เช่น เดียวกับ List ความแตกต่างระหว่างทูเพิลและลิสต์ คือ ทูเพิลไม่สามารถเปลี่ยนแปลงได้ต่างจากลิสต์ และทูเพิลใช้งานเดียว(...) ในขณะที่ลิสต์ใช้งานเดียวเหลือมี[...]
- ทูเพิลถูกจัดทำด้วยรายการแรกมีดัชนี [0] รายการที่สองมีดัชนี [1]
- ทูเพิลไม่สามารถเปลี่ยนแปลงได้ซึ่งหมายความว่าเราไม่สามารถเปลี่ยนแปลงเพิ่ม หรือลบรายการหลังจากสร้างทูเพิลแล้ว
- การสร้างทูเพิลทำได้ง่ายเพียงแค่ใส่ค่าต่างๆที่คั่นด้วยจุดפסיק สามารถเลือกที่จะใส่ค่าที่คั่นด้วยเครื่องหมายจุลภาคระหว่างวงเล็บได้ด้วย ตัวอย่างการสร้างทูเพิล เช่น

```
thistuple = ("apple", "banana", "cherry")  
print(thistuple)
```

- อนุญาตรายการที่ซ้ำกัน เนื่องจากทูเพิลถูกจัดทำด้วยจุดפסיקมีไอเท็มที่มีค่าเดียวกันได้:



ตัวอย่างที่ 4.12 ทูเพิลอนุญาตให้มีค่าซ้ำกันได้

```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")  
print(thistuple)
```

Output

```
('apple', 'banana', 'cherry', 'apple', 'cherry')
```



ความยาวทูเพิล หากต้องการแสดงจำนวนข้อมูลที่ทูเพิลให้ใช้ฟังก์ชัน `len()`:



ตัวอย่างที่ 4.13 พิมพ์จำนวนรายการในทูเพิล:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Output

3

สร้างทูเพิลด้วยรายการเดียว ในการสร้างทูเพิลที่มีเพียงรายการเดียวต้องเพิ่มลูกน้ำ(,) หลังรายการมิฉะนั้นไฟฟอนจะไม่รับรู้ว่าเป็นทูเพิล

```
thistuple = ("apple",)
print(type(thistuple))
```

#NOT a tuple

```
thistuple = ("apple")
print(type(thistuple))
```

output

```
<class 'tuple'>
<class 'str'>
```

Tuple Items – ประเภทข้อมูล

รายการ Tuple สามารถเป็นประเภทข้อมูลใดก็ได้:



ตัวอย่างที่ 4.14 ชนิดข้อมูลสตริง จำนวนเต็ม และบูลีน:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
tuple4 = ("abc", 34, True, 40, "male")
```

```
print(tuple1)
```

```
print(tuple2)
```

```
print(tuple3)
```

```
print(tuple4)
```

output

```
('apple', 'banana', 'cherry')
```

```
(1, 5, 7, 9, 3)
```

```
(True, False, False)
```

```
('abc', 34, True, 40, 'male')
```



ตัวอย่างที่ 4.15 ทูเพิลที่มีสตริงจำนวนเต็มและค่าบูลีน:

```
tuple4 = ("abc", 34, True, 40, "male")
print(tuple4)
```

output

```
('abc', 34, True, 40, 'male')
```

2.2 การเข้าถึงค่าในทูเพิล

หากต้องการเข้าถึงค่าในทูเพิลให้ใช้งานเล็บเหลี่ยมสำหรับการแบ่งส่วนพร้อมกับดัชนีหรือดัชนีเพื่อรับค่าที่มีอยู่ในดัชนีนั้น ตัวอย่างเช่น -

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );
print "tup1[0]: ", tup1[0];
print "tup2[1:5]: ", tup2[1:5];
```

output

```
tup1[0]: physics
```

```
tup2[1:5]: [2, 3, 4, 5]
```

2.3 การอัปเดตทูเพิล (Update Tuples)

■ เปลี่ยนค่าทูเพิล(Change Tuple Values)

เมื่อสร้างทูเพิลแล้วจะไม่สามารถเปลี่ยนค่าได้ ทูเพิลไม่สามารถเปลี่ยนแปลงได้หรือไม่เปลี่ยนรูปตามที่เรียก

แต่มีวิธีแก้ปัญหาคือ สามารถแปลงเป็น ลิสต์ เปลี่ยนลิสต์ และแปลงลิสต์กลับเป็นทูเพิลได้



ตัวอย่างที่ 4.16 แปลงทูเพิลเป็นลิสต์เพื่อให้สามารถเปลี่ยนแปลงได้:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

output

```
("apple", "kiwi", "cherry")
```



■ วนลูปผ่านทูเพิล (Loop Tuples)

สามารถวนลูปเพื่อเข้าถึงข้อมูลทูเพิลโดยการใช้ลูป for



ตัวอย่างที่ 4.17 วนชี้รายการและพิมพ์ค่า:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
    print(x)
```

output

apple
banana
cherry

■ วนชี้หมายเลขดัชนี

สามารถวนชี้รายการการทูเปลี่ยนได้โดยอ้างถึงหมายเลขดัชนี ใช้ฟังก์ชันrange()และlen()เพื่อสร้างการทำซ้ำที่เหมาะสม



ตัวอย่างที่ 4.18 พิมพ์รายการทั้งหมดโดยอ้างถึงหมายเลขดัชนี:

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
    print(thistuple[i])
```

output

apple
banana
cherry

■ วนลูป (While)

สามารถวนรอบรายการโดยใช้การ while วนซ้ำ ใช้len()ฟังก์ชันเพื่อกำหนดความยาวของทู เปิลจากนั้นเริ่มต้นที่ 0 และวนซ้ำไปตามรายการทูเปลี่ยนโดยอ้างถึงดัชนี อย่าลืมเพิ่มดัชนีทีละ 1 หลังจากการทำซ้ำแต่ละครั้ง



ตัวอย่างที่ 4.19 วนลูป

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
    print(thistuple[i])
    i = i + 1
```

**Output**

```
apple
banana
cherry
```

2.4 การรวมทูเพิล (Join Tuples)

การรวมทูเพิล โดยใช้ตัวดำเนินการ +

**ตัวอย่างที่ 4.20 การรวมทูเพิล**

```
tuple1 = ("a", "b", "c")
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)
```

output

```
('a', 'b', 'c', 1, 2, 3)
```

การคูณทูเพิล การคูณเนื้อหาของทูเพิลตามจำนวนครั้งที่กำหนดสามารถใช้ตัว * ดำเนินการ:

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

output

```
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

2.5 เมรอดและฟังก์ชันที่ใช้งานกับชนิดข้อมูลทูเพิล

ชนิดข้อมูลทูเพิลเก็บชนิดข้อมูลได้หลากหลายประเภท เช่น จำนวนเต็ม ทศนิยม стрิง ลิสต์ เป็นต้น ภาษาไพทอนได้เตรียมเมรอดและฟังก์ชันไว้ให้เรียกใช้งานดังต่อไปนี้

ตารางที่ 17 เมรอดและฟังก์ชันที่ใช้งานกับชนิดข้อมูลทูเพิล

เมรอด	ความหมาย	รูปแบบการใช้งาน
index()	ใช้สำหรับแสดงตำแหน่งของข้อมูล	tub.index()
count()	ใช้สำหรับนับจำนวนข้อมูล	tub.count(x)
len()	ฟังก์ชันใช้สำหรับแสดงจำนวนข้อมูลที่อยู่ในทูเพิล	len(tub)



หมายเหตุ tub คือ ตัวแปรชนิดทูเพิล , x คือ ค่าข้อมูลใดๆ



ตัวอย่างที่ 4.21 ให้แปลงเลขอารบิกเป็นเลขไทยและแสดงคำอ่านแบบเรียงคำ โดยเก็บรายการเลขไทยแบบทุกเพิล จากนั้นนำเลขอารบิกแต่ละตัวมาใช้เป็นลำดับในการอ่านข้อมูลจากทุกเพิล เพื่อให้ได้ค่า ที่ตรงกัน

```
tp_thai_digits = ('๐', '๑', '๒', '๓', '๔', '๕', '๖', '๗', '๘', '๙')
tp_digit_words = ('ศูนย์', 'หนึ่ง', 'สอง', 'สาม', 'สี่', 'ห้า', 'หก', 'เจ็ด', 'แปด', 'เก้า')

number = input('กำหนดตัวเลขอารบิกเป็นจำนวนเต็มบวก >>')
number = tuple(number)      #ไม่ต้องมีบรรทัดนี้เลยก็ได้
thai_digit = ""
thai_word = ""

for n in number:           #วนลูปตามจำนวนตัวเลข
    n = int(n)
    thai_digit += tp_thai_digits[n]  #ใช้ตัวเลขนั้นเป็นลำดับในการอ่านค่าจากทุกเพิล
    thai_word += tp_digit_words[n]

print('เลขไทย:', thai_digit)
print('คำอ่าน:', thai_word)
```

กำหนดตัวเลขอารบิกเป็นจำนวนเต็มบวก >>2520
 เลขไทย: ๒๕๒๐
 คำอ่าน: สองห้าสองศูนย์
 Process finished with exit code 0

3. รายการข้อมูลแบบเซต (Sets)

```
myset = {"apple", "banana", "cherry"}
```

- เชต ใช้เพื่อเก็บหลายรายการในตัวแปรเดียว
- เชต เป็นหนึ่งใน 4 ประเภทข้อมูลในตัวในไฟฟอนที่ใช้ในการจัดเก็บคอลเลคชันข้อมูลอีก 3 ประเภท คือ List , Tuple และDictionary ซึ่งทั้งหมดมีคุณภาพและการใช้งานที่แตกต่าง
- เชต ไม่มีลำดับในการจัดเก็บที่แน่นอน ดังนั้นจึงไม่สามารถเข้าถึงสมาชิกตัวใดตัวหนึ่งแบบเจาะจงด้วยเลขลำดับเหมือนลิสต์หรือทุกเพิล
- เชต จะเขียนด้วยวงเล็บปีกกา(curlly brackets) {}



ตัวอย่างที่ 4.22 การสร้างเซต

```
thisset = {"apple", "banana", "cherry"}  
print(thisset)
```

output

```
{'cherry', 'apple', 'banana'}
```



หมายเหตุ เซตไม่เรียงลำดับหมายความว่ารายการจะเรียงลำดับแบบสุ่ม



ตัวอย่างที่ 4.23 ค่าที่ซ้ำกันจะถูกละเว้น:

```
thisset = {"apple", "banana", "cherry", "apple"}  
print(thisset)
```

output

```
{ "banana", "cherry", "apple"}
```



ตัวอย่างที่ 4.24 นับจำนวนรายการในเซต:

```
thisset = {"apple", "banana", "cherry"}  
print(len(thisset))
```

output

```
3
```

```
#ชนิดข้อมูลสตริง จำนวนเต็ม และบูลีน  
set1 = {"apple", "banana", "cherry"}  
set2 = {1, 5, 7, 9, 3}  
set3 = {True, False, False}
```

#ชุดที่มีสตริงจำนวนเต็มและค่าบูลีน:

```
set1 = {"abc", 34, True, 40, "male"}
```

3.1 การเข้าถึงเซต (Access Set Items)

ไม่สามารถเข้าถึงรายการในเซตได้โดยอ้างถึงตัวชี้หรือคีย์ แต่สามารถวนซ้ำรายการชุดโดยใช้การ for วนซ้ำหรือถามว่ามีค่าที่ระบุอยู่ในเซตหรือไม่โดยใช้คำสำคัญ in



ตัวอย่างที่ 4.25 การเข้าถึงเซต

```
thisset = {"apple", "banana", "cherry"}
```

```
for x in thisset:
```

```
    print(x)
```

output

cherry

banana

apple

```
thisset = {"apple", "banana", "cherry"}
```

```
print("banana" in thisset)
```

output

True

3.2 การเพิ่มรายการเซต

เมื่อสร้าง set แล้วจะไม่สามารถเปลี่ยนรายการได้ แต่สามารถเพิ่มรายการใหม่ได้ หากต้องการเพิ่มนึ่งรายการในชุดให้ใช้ เมธอด add()

- Add Set Items



ตัวอย่างที่ 4.26 เพิ่มรายการ(item) ลงในเซตโดยใช้เมธอด add():

```
thisset = {"apple", "banana", "cherry"}
```

```
thisset.add("orange")
```

```
print(thisset)
```

output

{'orange', 'banana', 'cherry', 'apple'}

- Add Sets

หากต้องการเพิ่มเซตจากชุดอื่นลงในเซตปัจจุบันให้ใช้เมธอด update():



ตัวอย่างที่ 4.27 Add elements from **tropical** and **thisset** into **newset**:

```
thisset = {"apple", "banana", "cherry"}
```

```
tropical = {"pineapple", "mango", "papaya"}
```

```
thisset.update(tropical)
```

```
print(thisset)
```

**Output**

{'apple', 'mango', 'cherry', 'pineapple', 'banana', 'papaya'}

▪ เพิ่มรายการที่ทำซ้ำได้ (Add Any Iterable)

อ็อบเจกต์ใน เมธอด update() ที่ไม่มีเซตสามารถเป็นอ็อบเจกต์ที่ทำซ้ำได้ (ทูเพล, ลิสต์ พจนานุกรม และอื่น ๆ)

**ตัวอย่างที่ 4.28 เพิ่มรายการทำซ้ำ update()**

```
thisset = {"apple", "banana", "cherry"}  
mylist = ["kiwi", "orange"]  
thisset.update(mylist)  
  
print(thisset)
```

output

{'banana', 'cherry', 'apple', 'orange', 'kiwi'}

3.3 Remove Set Items

หากต้องการลบรายการในชุดให้ใช้เมธอด remove() หรือdiscard()

**ตัวอย่างที่ 4.29 ลบ "banana" โดยใช้ remove() method:**

```
#เมธอด remove()  
thisset = {"apple", "banana", "cherry"}  
thisset.remove("banana")  
  
print(thisset)
```

output

{'apple', 'cherry'}



หมายเหตุ: หากไม่มีรายการที่จะลบremove()จะทำให้เกิดข้อผิดพลาดขึ้น

**ตัวอย่างที่ 4.30 ลบ "banana" โดยใช้ discard() method:**

```
#เมธอด discard()  
thisset = {"apple", "banana", "cherry"}  
thisset.discard("banana")  
  
print(thisset)
```

output

{'cherry', 'apple'}



หมายเหตุ: หากไม่มีรายการที่จะลบdiscard()จะ ไม่ทำให้เกิดข้อผิดพลาด

สามารถใช้`pop()` method ลบรายการได้ แต่วิธีนี้จะลบรายการสุดท้าย โปรดจำไว้ว่า`set` ไม่เรียงลำดับดังนั้นจะไม่ทราบว่ารายการใดถูกลบออกไป
ค่าที่ส่งคืนของ`pop()` method คือ รายการที่ถูกลบออก



ตัวอย่างที่ 4.31 ลบรายการสุดท้ายโดยใช้`pop()` method:

```
#ลบรายการสุดท้ายโดยใช้เมธอด pop()
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()
print(x)

print(thisset)
```

output

```
apple
{'cherry', 'banana'}
```



ตัวอย่างที่ 4.32 `clear()` method จะทำให้เซ็ตว่าง:

```
thisset = {"apple", "banana", "cherry"}
thisset.clear()

print(thisset)
```

output

```
set()
```



ตัวอย่างที่ 4.33 คีย์เวิร์ด `del` จะลบเซ็ตได้สมบูรณ์:

```
thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset)
```

output

Traceback (most recent call last):

```
  File "C:\Users\HPSurajak\AppData\Local\Temp\Rar$Dla20196.49467\del_set.py", line 5, in <module>
    print(thisset)
NameError: name 'thisset' is not defined
```



3.4 Loop Sets

สามารถวนซ้ำรายการการที่กำหนดได้โดยใช้ลูป for :



ตัวอย่างที่ 4.34 วนรอบเซต และพิมพ์ค่า:

```
thisset = {"apple", "banana", "cherry"}  
for x in thisset:  
    print(x)
```

output

```
banana  
cherry  
apple
```

3.5 Join Sets

มีหลายวิธีในการรวมสองเซตขึ้นไปในไฟฟอนซึ่งสามารถใช้เมธอด union() ที่ส่งคืนเซตใหม่ที่มีรายการทั้งหมดจากทั้งสองชุดหรือ update() วิธีที่แทรกรายการทั้งหมดจากชุดหนึ่งไปยังอีกชุดหนึ่ง:



ตัวอย่างที่ 4.35 เมธอด union() ส่งกลับเซตใหม่กับรายการทั้งหมดจากทั้ง 2 เซต:

```
set1 = {"a", "b", "c"}  
set2 = {1, 2, 3}  
set3 = set1.union(set2)  
print(set3)
```

output

```
{'a', 2, 'c', 3, 1, 'b'}
```



ตัวอย่างที่ 4.36 เมธอด update() แทรกรายการใน set2 ลง set1:

```
set1 = {"a", "b", "c"}  
set2 = {1, 2, 3}  
set1.update(set2)  
print(set1)
```

output

```
{3, 2, 'b', 'a', 'c', 1}
```



หมายเหตุ: ทั้งสอง union() และ update() จะไม่รวมรายการที่ซ้ำกัน

3.6 Set Methods

ตารางที่ 18 เมธอดและคำอธิบาย

เมธอด	คำอธิบาย
<code>add()</code>	Adds an element to the set
<code>clear()</code>	Removes all the elements from the set
<code>copy()</code>	Returns a copy of the set
<code>difference()</code>	Returns a set containing the difference between two or more sets
<code>difference_update()</code>	Removes the items in this set that are also included in another, specified set
<code>discard()</code>	Remove the specified item
<code>intersection()</code>	Returns a set, that is the intersection of two other sets
<code>intersection_update()</code>	Removes the items in this set that are not present in other, specified set(s)
<code>isdisjoint()</code>	Returns whether two sets have a intersection or not
<code>issubset()</code>	Returns whether another set contains this set or not
<code>issuperset()</code>	Returns whether this set contains another set or not
<code>pop()</code>	Removes an element from the set
<code>remove()</code>	Removes the specified element
<code>symmetric_difference()</code>	Returns a set with the symmetric differences of two sets
<code>union()</code>	Return a set containing the union of sets
<code>update()</code>	Update the set with the union of this set and others

4. รายการข้อมูลแบบดิกชันนารี (Dictionaries)

4.1 เกี่ยวกับดิกชันนารี

- ดิกชันนารี คือ การจัดเก็บข้อมูลแบบรายการที่สามารถแบ่งตัวมืองค์ประกอบ 2 อย่าง คือ key และ value
- ดิกชันนารี ใช้เพื่อกีบค่าข้อมูลใน key : value
- ดิกชันนารี คือ คอลเลกชันที่ไม่มีการเรียงลำดับเบลี่ยนแปลงได้และไม่อนุญาตให้มีการทำซ้ำ
- ดิกชันนารี เขียนด้วย วงเล็บปีกๆ { } และมีคีย์และค่าดังนี้

key	value
Red	แดง
Green	เขียว
blue	น้ำเงิน
Pink	ชมพู



- Dictionary Items



ตัวอย่างที่ 4.37 สร้างและพิมพ์ dictionary:

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}  
print(thisdict)
```

Output

```
{'brand': 'Toyota', 'model': 'Fortuner', 'year': 2020}
```



ตัวอย่างที่ 4.38 พิมพ์ "brand" ค่าใน dictionary:

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020,  
}  
print(thisdict["brand"])
```

Output

```
Toyota
```

- ไม่เรียงลำดับ (Unordered)

เมื่อเราบอกว่าพจนานุกรมไม่เรียงลำดับหมายความว่ารายการนั้นไม่มีลำดับที่กำหนดไว้ไม่สามารถอ้างถึงรายการโดยใช้ชั้นนี้ได้

- เปลี่ยนแปลงได้ (Changeable)

พจนานุกรมสามารถเปลี่ยนแปลงได้ซึ่งหมายความว่าเราสามารถเปลี่ยนแปลงเพิ่มหรือลบรายการต่างๆได้หลังจากสร้างพจนานุกรมแล้ว

- ไม่อนุญาตให้ทำซ้ำ (Duplicates Not Allowed)

พจนานุกรมไม่สามารถมีสองรายการที่มีคีย์เดียวกัน



ตัวอย่างที่ 4.39 ค่าที่ซ้ำกันจะเขียนทับค่าที่มีอยู่:

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020,  
    "year": 2021  
}
```

```
print(thisdict)
```

Output

```
{'brand': 'Toyota', 'model': 'Fortuner', 'year': 2021}
```

4.2 การเข้าถึงสมาชิกของตัวชี้อันนารี

สามารถเข้าถึงรายการของพจนานุกรมโดยอ้างถึงชื่อคีย์ภายในวงเล็บ []



ตัวอย่างที่ 4.40 รับค่าของคีย์ "model":

```
thisdict = {
    "brand": "Toyota",
    "model": "Fortuner",
    "year": 2020
}
x = thisdict["model"]
print(x)
```

Output

```
Fortuner
```



ตัวอย่างที่ 4.41 รับค่าของคีย์ "model": นอกเหนือไปนี้ยังใช้ เมธอด get() จะให้ผลลัพธ์เดียวกัน:

```
thisdict = {
    "brand": "Toyota",
    "model": "Fortuner",
    "year": 2020
}
x = thisdict.get("model")
print(x)
```

Output

```
Fortuner
```

เมธอด keys() จะกลับรายการของคีย์ทั้งหมดในพจนานุกรม



ตัวอย่างที่ 4.42 เมธอด keys()

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020}  
  
x = thisdict.keys()  
  
print(x)
```

Output

```
dict_keys(['brand', 'model', 'year'])
```

4.3 การเปลี่ยนรายการติกชั้นนำไปสู่การเปลี่ยนค่าของรายการเฉพาะได้โดยอ้างถึงชื่อคีย์:



ตัวอย่างที่ 4.43 เปลี่ยนค่า

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}  
  
thisdict["year"] = 2021  
  
print(thisdict)
```

Output

```
{'brand': 'Toyota', 'model': 'Fortuner', 'year': 2021}
```

เมธอด update() สามารถรับปรุ่ง dict ที่มีรายการจากอาร์กิวเมนต์ที่กำหนด อาร์กิวเมนต์ต้องเป็น dict หรือ object ที่ทำสำเนาได้ซึ่งมี key: value



ตัวอย่างที่ 4.44 อัปเดต "ปี" ของรถโดยใช้ เมธอด update()

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}  
  
thisdict.update({"year": 2018})  
  
print(thisdict)
```

Output

```
{'brand': 'Toyota', 'model': 'Fortuner', 'year': 2018}
```

4.4 การเพิ่มรายการดิกชันนารี

การเพิ่มรายการลงใน dictionary ทำได้โดยใช้คีย์ดัชนีใหม่และกำหนดค่าให้กับ dict:



ตัวอย่างที่ 4.46 การเพิ่มรายการ

```
thisdict =      {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020}  
thisdict["color"] = "red"  
print(thisdict)
```

Output

```
{'brand': 'Toyota', 'model': 'Fortuner', 'year': 2020, 'color': 'red'}
```



ตัวอย่างที่ 4.47 การเพิ่มรายการด้วย update()

```
thisdict =      {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020}  
thisdict.update({"color": "red"})  
print(thisdict)
```

Output

```
{'brand': 'Toyota', 'model': 'Fortuner', 'year': 2020, 'color': 'red'}
```



4.5 การลบรายการติดชั้นnaire

มีหลายวิธีในการลบรายการออกจากติดชั้นnaire



ตัวอย่างที่ 4.48 pop()

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}  
  
thisdict.pop("model")  
print(thisdict)
```

Output

```
{'brand': 'Toyota', 'year': 2020}
```



ตัวอย่างที่ 4.49 popitem()

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}  
  
thisdict.popitem()  
print(thisdict)
```

Output

```
{'brand': 'Toyota', 'model': 'Fortuner'}
```



ตัวอย่างที่ 4.50 คีย์เวิร์ด **del** เลือกลบบางรายการ

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}  
  
del thisdict["model"]  
print(thisdict)
```

Output

```
{'brand': 'Toyota', 'year': 2020}
```



ตัวอย่างที่ 4.51 คีย์เวิร์ด `del` ลบทั้งหมด

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}  
  
del thisdict  
  
print(thisdict)  
#จะทำให้เกิดข้อผิดพลาดเนื่องจากไม่มี "thisdict" อีกต่อไป
```

```
Traceback (most recent call last):  
  File "C:\Users\HPSurajak\PycharmProjects\FirstApp\main.py", line 7, in <module>  
    print(thisdict)  
NameError: name 'thisdict' is not defined  
  
Process finished with exit code 1
```



ตัวอย่างที่ 4.52 `clear()`

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}  
  
thisdict.clear  
  
print(thisdict)
```

Output

```
{}
```



4.6 การวนซ้ำดิกชันนารี (Loop Dictionaries)

- สามารถวนผ่านพจนานุกรมได้โดยใช้ for วนซ้ำ
- เมื่อวนลูปผ่านพจนานุกรมค่าที่ส่งคืนเป็นคีย์ของพจนานุกรม แต่ก็มีวิธีการคืนค่าเช่นกัน



ตัวอย่างที่ 4.53 พิมพ์ชื่อ dict ทั้งหมด

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}  
  
for x in thisdict:  
    print(x)
```

Output

brand
model
year



ตัวอย่างที่ 4.54 พิมพ์ทุก value ใน dictionary

```
thisdict = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}  
  
for x in thisdict:  
    print(thisdict[x])
```

Output

Toyota
Fortuner
2020



ตัวอย่างที่ 4.55 ใช้ method values() คืนค่า value ใน dictionary:

```
thisdict = {
    "brand": "Toyota",
    "model": "Fortuner",
    "year": 2020
}
for x in thisdict.values():
    print(x)
```

Output

Toyota
Fortuner
2020



ตัวอย่างที่ 4.56 ใช้ keys() method คืน key ใน dictionary:

```
thisdict = {
    "brand": "Toyota",
    "model": "Fortuner",
    "year": 2020
}
for x in thisdict.keys():
    print(x)
```

Output

brand
model
year



ตัวอย่างที่ 4.57 วนซ้ำทั้ง key และ value โดยใช้ items()

```
thisdict = {
    "brand": "Toyota",
    "model": "Fortuner",
    "year": 2020
}
for x,y in thisdict.items():
    print(x,y)
```



Output

```
brand Toyota
model Fortuner
year 2020
```

4.7 การคัดลอกดิกชันนารี (Copy Dictionaries)

ไฟฟอนไม่สามารถทำการคัดลอก dictionary ด้วยการพิมพ์ dict2 = dict1, เพราะว่า :
dict2 จะอ้างอิงถึง dict1, and และทำการเปลี่ยน dict1 โดยอัตโนมัติใน dict2 ทั้งนี้เราสามารถทำการสำเนาโดยใช้ built-in Dictionary เมธอด copy()



ตัวอย่างที่ 4.58 ทำการสำเนาดิกชันนารีด้วย เมธอด copy()

```
thisdict = {
    "brand": "Toyota",
    "model": "Fortuner",
    "year": 2020
}
mydict = thisdict.copy()
print(mydict)

{'brand': 'Toyota', 'model': 'Fortuner', 'year': 2020}

Process finished with exit code 0
```

อีกวิธีหนึ่งในการทำการสำเนา คือ การใช้ฟังก์ชัน dict()



ตัวอย่างที่ 4.59 ทำการสำเนาดิกชันนารีด้วย เมธอด dict()

```
thisdict = {
    "brand": "Toyota",
    "model": "Fortuner",
    "year": 2020
}
mydict = dict(thisdict)
print(mydict)

{'brand': 'Toyota', 'model': 'Fortuner', 'year': 2020}

Process finished with exit code 0
```

4.8 ดิกชันนารีที่ซ้อนกัน (Nested Dictionaries)



ตัวอย่างที่ 4.60 Nested Dictionaries ช้อนกัน 3 ชุด

```
myfamily = {
    "child1" : {
        "name" : "COM",
        "year" : 2010
    },
    "child2" : {
        "name" : "EDU",
        "year" : 2015
    },
    "child3" : {
        "name" : "KSU",
        "year" : 2020
    }
}

print(myfamily)
```

```
{'child1': {'name': 'COM', 'year': 2010}, 'child2': {'name': 'EDU', 'year': 2015}, 'child3': {'name': 'KSU', 'year': 2020}}
Process finished with exit code 0
```

หรือ สามารถ add 3 dictionaries ลง new dictionary ก็ได้:



ตัวอย่างที่ 4.61 สร้าง 3 dictionaries จากนั้นสร้าง dictionary เพื่อเก็บ 3 dictionaries:

```
child1 = {
    "name" : "COM",
    "year" : 2010
}

child2 = {
    "name" : "EDU",
    "year" : 2015
}

child3 = {
    "name" : "KSU",
    "year" : 2020
}
```



```
}
```

```
myfamily = {  
    "child1" : child1,  
    "child2" : child2,  
    "child3" : child3  
}  
  
print(mymemory)
```

```
{'child1': {'name': 'COM', 'year': 2010}, 'child2': {'name': 'EDU', 'year': 2015}, 'child3': {'name': 'KSU', 'year': 2020}}  
Process finished with exit code 0
```

4.9 เมธอดของดิกชันนารี (Dictionary Methods)

Python มีชุดของ built-in methods ที่สามารถใช้ร่วมกับ dictionaries.

ตารางที่ 19 ชุดของ built-in methods

Method	Description
<code>clear()</code>	Removes all the elements from the dictionary
<code>copy()</code>	Returns a copy of the dictionary
<code>fromkeys()</code>	Returns a dictionary with the specified keys and value
<code>get()</code>	Returns the value of the specified key
<code>items()</code>	Returns a list containing a tuple for each key value pair
<code>keys()</code>	Returns a list containing the dictionary's keys
<code>pop()</code>	Removes the element with the specified key
<code>popitem()</code>	Removes the last inserted key-value pair
<code>setdefault()</code>	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
<code>update()</code>	Updates the dictionary with the specified key-value pairs
<code>values()</code>	Returns a list of all the values in the dictionary



ตัวอย่างที่ 4.62 clear()

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
car.clear()
```

```
print(car)
```

output

```
{}
```



ตัวอย่างที่ 4.63 copy()

```
car = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020}  
}  
  
x = car.copy()  
print(x)
```

output

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```



ตัวอย่างที่ 4.64 fromkeys()

```
x = ('key1', 'key2', 'key3')  
y = 0  
thisdict = dict.fromkeys(x, y)  
print(thisdict)
```

output

```
{'key1': 0, 'key2': 0, 'key3': 0}
```



ตัวอย่างที่ 4.65 get()

```
car = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020}  
x = car.get("price", 15000)
```



```
print(x)
```

output

15000



ตัวอย่างที่ 4.66 items()

```
car = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}  
  
x = car.items()  
car["year"] = 2018  
  
print(x)
```

```
dict_items([('brand', 'Toyota'), ('model', 'Fortuner'), ('year', 2018)])
```



ตัวอย่างที่ 4.67 keys()

```
car = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}  
  
x = car.keys()  
car["color"] = "white"  
  
print(x)
```

output

```
dict_keys(['brand', 'model', 'year', 'color'])
```



ตัวอย่างที่ 4.68 pop()

```
car = {  
    "brand": "Toyota",  
    "model": "Fortuner",  
    "year": 2020  
}
```

```
x = car.pop("model")
print(x)
```

output

Fortuner



ตัวอย่างที่ 4.69 popitem()

```
car = {
    "brand": "Toyota",
    "model": "Fortuner",
    "year": 2020
}
x = car.popitem()
print(x)
```

output

('year', 2020)



ตัวอย่างที่ 4.70 setdefault()

```
car = {
    "brand": "Toyota",
    "model": "Fortuner",
    "year": 2020
}
x = car.setdefault("color", "white")
print(x)
```

output

White



ตัวอย่างที่ 4.71 update()

```
car = {
    "brand": "Toyota",
    "model": "Fortuner",
    "year": 2020
}
```



```
car.update({"color": "White"})
print(car)
```

output

```
{'brand': 'Toyota', 'model': 'Fortuner', 'year': 2020, 'color': 'White'}
```



ตัวอย่างที่ 4.72 values()

```
car = {
    "brand": "Toyota",
    "model": "Fortuner",
    "year": 2020
}
x = car.values()
car["year"] = 2018
print(x)
```

output

```
dict_values(['Toyota', 'Fortuner', 2018])
```

5. สรุปหัวยบทะเบียน

การรวมข้อมูล 4 ประเภทในภาษาโปรแกรมไพทอน:

- Lists คือ ข้อมูลชนิดลำดับรายการและเปลี่ยนแปลงได้ อนุญาตให้สมาชิกซ้ำกัน
- Tuple ข้อมูลชนิดลำดับรายการและไม่สามารถเปลี่ยนแปลงได้ อนุญาตให้สมาชิกซ้ำกัน
- Set คือ คอลเลกชันที่ไม่มีการเรียงลำดับและไม่ได้จัดทำด้วย ไม่มีสมาชิกซ้ำกัน
- Dictionarie คือ คอลเลกชันที่ไม่เรียงลำดับและเปลี่ยนแปลงได้ ไม่มีสมาชิกซ้ำกัน
เมื่อเลือกประเภทคอลเลกชันจะเป็นประโยชน์ในการทำความเข้าใจคุณสมบัติของประเภทนั้น ๆ การเลือกประเภทที่เหมาะสมสำหรับข้อมูลหนึ่ง ๆ อาจหมายถึงการคงความหมายไว้และอาจหมายถึงการเพิ่มประสิทธิภาพหรือความปลอดภัย

6. กิจกรรม



1. ให้รับค่าปี พ.ศ. เข้ามา และคำนวณว่าปีนี้ตรงกับปีนักษัตรใด โดยที่
 - เก็บชื่อปีนักษัตรไว้ในแบบลิสต์ เช่น `constellation = ['ชวด', 'ฉลู', ..., 'กุน']`
 - โดยสูตรคำนวณหาปีนักษัตรแบบง่าย เพื่อใช้กับลิสต์โดยหาค่าที่เหลือจากการหารดังนี้

$$\text{ค่าที่เหลือ} = (\text{ปี พ.ศ.} + 5) \% 12$$

- นำค่าที่เหลือไปใช้เป็นเลขลำดับในการอ่านชื่อปืนกษัตรจากลิสต์ที่ได้สร้างไว้ เช่น ถ้าได้ค่าที่เหลือเป็น 1 แสดงว่าตรงกับ ‘ฉลุ’ เป็นต้น

```
constellations = ['ชวด', 'ฉลุ', 'ເຖິງ', 'ໝາລ', 'ມະໂຮງ', 'ມະເສັງ',  
                  'ມະເມີຍ', 'ມະແມ', 'ວອກ', 'ຮກາ', 'ຈອ', 'ກຸນ']

print('ຄ້າຕ້ອງການຫຍຸດ ໃຫ້ໄສຄ່ານໍ້ອຍກວ່າ 1');

while True:  
    year = int(input('ປີ ພ.ສ. >> '))  
    if year < 1:  
        break  
  
    x = (year + 5) % 12  
    c = constellations[x]  
    print('ຕຽບກັບປື້ນັກເຈົ້າ:', c)  
    print()
```

2. เก็บรายชื่อวันสำคัญของเดือนต่างๆ ในแบบลิสต์ 2 มิติ จากนั้นวนลูปเพื่อตรวจสอบว่าในแต่ละเดือนนั้นมีวันสำคัญอะไรบ้าง และแสดงผลออกมา ผลลัพธ์ดังนี้

```

#ให้เดือนแรกเป็นช่องว่าง เพื่อใช้แทนเดือนลำดับที่ 0
#ดังนั้นลำดับของเดือนมกราคมก็จะเป็น 1 ตามที่เราเข้าใจ
months = [ ' ', 'มกราคม', 'กุมภาพันธ์', 'มีนาคม', 'เมษายน', 'พฤษภาคม', 'มิถุนายน',
           'กรกฎาคม', 'สิงหาคม', 'กันยายน', 'ตุลาคม', 'พฤศจิกายน', 'ธันวาคม' ]

#สร้างลิสต์ว่างแบบ 2 มิติ
days = []
for m in range(13):      #[[เดือน 0], [เดือน 1], [เดือน 2], ...[เดือน 12]]
    days.append([])
    for d in range(32):    #[[วันที่ 0, วันที่ 1, วันที่ 2, ..., วันที่ 31], [วันที่ 0, วันที่ 1, วันที่ 2,
                           ..., วันที่ 31], ...]
        days[m].append("")

days[1][1] = 'วันขึ้นไปใหม่'
days[1][16] = 'วันครู'
days[2][14] = 'วันวานเล่นไทน์'
days[4][6] = 'วันจักรี'

```



```
days[4][13] = 'วันสงกรานต์'  
days[7][28] = 'วันเฉลิมพระชนมพรรษา ร.10'  
days[8][12] = 'วันแม่'  
days[10][23] = 'วันปิยมหาราช'  
days[12][5] = 'วันพ่อ'  
days[12][10] = 'วันรัฐธรรมนูญ'  
days[12][25] = 'วันคริสต์มาส'  
days[12][31] = 'วันสิ้นปี'  
  
print('วันสำคัญในเดือนต่างๆ')  
  
for m in range(len(days)):      # วนลูปตามจำนวนเดือน (0 - 12)  
    important_days = ""  
  
    for d in days[m]:          # วนลูปตามจำนวนวันที่ของแต่ละเดือน  
        if d != "":  
            if important_days != "":  
                important_days += ','  
  
            important_days += d  
  
    #ถ้าเดือนนั้นมีวันสำคัญ ก็ให้แสดงออกไป  
    if important_days != "":  
        print(f'เดือน {months[m]} : {important_days}')
```

```
วันสำคัญในเดือนต่างๆ  
เดือน มกราคม : วันขึ้นปีใหม่, วันครู  
เดือน กุมภาพันธ์ : วันวาเลนไทน์  
เดือน เมษายน : วันวัฒนธรรมไทย, วันสงกรานต์  
เดือน กรกฎาคม : วันเฉลิมพระชนมพรรษา ร.10  
เดือน สิงหาคม : วันแม่  
เดือน ตุลาคม : วันปิยมหาราช  
เดือน ธันวาคม : วันพ่อ, วันรัฐธรรมนูญ, วันคริสต์มาส, วันสิ้นปี
```

```
Process finished with exit code 0
```

7. แบบฝึกหัดท้ายบท

- จงเขียนโปรแกรมเพื่อเก็บชื่อและสีของวันทั้ง 7 ในรูปสั้นๆด้วยลิสต์หรือทุเพิล 2 มิติ แล้วใช้ลูป for เพื่ออ่านค่ามาแสดงผล

```

วันอาทิตย์ - สีแดง
วันจันทร์ - สีเหลือง
วันอังคาร - สีชมพู
วันพุธ - สีเขียว
วันพฤหัสบดี - สีส้ม
วันศุกร์ - สีฟ้า
วันเสาร์ - สีม่วง

```

`Process finished with exit code 0`

- จงเขียนโปรแกรมรับข้อมูลทางคีย์บอร์ด แล้วนำมาทำหนดเป็นลิสต์ จากนั้นให้เรียงลำดับจากน้อยไปมาก (สามารถใช้ฟังก์ชัน sort) และจากมากไปน้อย (กรณีต้องนำลิสต์ที่เรียงลำดับจากน้อยไปมากเอาไว้แล้ว มาเรียงแบบย้อนกลับ)

```

จำนวนที่ 1 >>100
จำนวนที่ 2 >>201
จำนวนที่ 3 >>1001
จำนวนที่ 4 >>2001
จำนวนที่ 5 >>4001
เรียงจากน้อยไปมาก: [100, 201, 1001, 2001, 4001]
เรียงจากมากไปน้อย: [4001, 2001, 1001, 201, 100]

```

`Process finished with exit code 0`

- จงเขียนโปรแกรมเพื่อรับข้อมูลจากคีย์บอร์ดเป็นคะแนนของนักเรียนจำนวนหนึ่งจากนั้นพิจารณาให้เกรดโดยยึดເອົາຜູ້ได้คะแนนสูงສุด (top) เป็นเกณฑ์ดังนี้

- ได้เกรด A ถ้าคะแนน \geq top-10
- ได้เกรด B ถ้าคะแนน \geq top-20
- ได้เกรด C ถ้าคะแนน \geq top-30
- ได้เกรด D ถ้าคะแนน \geq top-40
- ได้เกรด F ถ้าคะแนน $<$ top-40

```

คะแนนของนักเรียนคนที่ 1 >>50
คะแนนของนักเรียนคนที่ 2 >>60
คะแนนของนักเรียนคนที่ 3 >>65
คะแนนของนักเรียนคนที่ 4 >>70
คะแนนของนักเรียนคนที่ 5 >>75
คะแนนของนักเรียนคนที่ 6 >>80
-----
```

```

นักเรียนคนที่ 1 ได้เกรด: C
นักเรียนคนที่ 2 ได้เกรด: B
นักเรียนคนที่ 3 ได้เกรด: B
นักเรียนคนที่ 4 ได้เกรด: A
นักเรียนคนที่ 5 ได้เกรด: A
นักเรียนคนที่ 6 ได้เกรด: A

```

`Process finished with exit code 0`

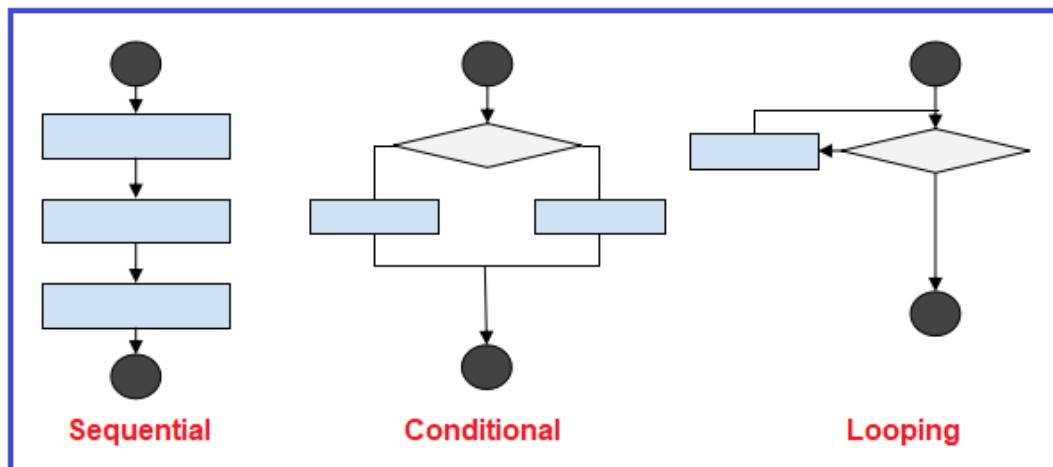


บทที่ 5

คำสั่งควบคุมทิศทางการทำงานของโปรแกรม

ในบทนี้ จะได้เรียนรู้คำสั่นงช้าในภาษาไพทอน ซึ่งจะพูดถึงการควบคุมการทำงานโดยการใช้คำสั่งที่สามารถควบคุมโปรแกรมให้ทำงานช้าๆ ในเงื่อนไขที่กำหนดและเพิ่มความสามารถของการเขียนโปรแกรม ตัวอย่างของการทำงานช้าๆ นั่นพบเห็นได้ทั่วไปในชีวิตประจำวัน เช่น โปรแกรมพยากรณ์สภาพอากาศที่เกิดขึ้นในทุกๆ วัน ดังนั้นแนวคิดเหล่านี้จึงถูกนำมาใช้กับการเขียนโปรแกรมในการเขียนโปรแกรม สามารถกำหนดการทำงานของโปรแกรมได้โดยใช้ คำสั่งควบคุมทิศทางการทำงานของโปรแกรมซึ่งมี 3 รูปแบบ คือ

1. คำสั่งควบคุมแบบลำดับ(Sequence Control Statement)
2. คำสั่งควบคุมแบบมีเงื่อนไข(Condition Control Statement)
3. คำสั่งควบคุมแบบทำซ้ำ(Iteration Control Statement)



ภาพที่ 4 แสดงผังงานรูปแบบคำสั่ง

1. คำสั่งควบคุมแบบลำดับ

การเขียนโปรแกรมควบคุมการทำงานแบบลำดับ เป็นลักษณะการเขียนคำสั่งให้โปรแกรมทำงานแบบทำงานจากบนลงล่าง โดยทำงานจากคำสั่งที่ 1 คำสั่งที่ 2 คำสั่งที่ 3 ไปเรื่อยๆ จนครบ ทุกคำสั่ง ไม่มีการตัดสินใจ ไม่มีการทำซ้ำ และไม่มีคำสั่งกระโดดข้าม ดังตัวอย่าง



ตัวอย่างที่ 5.1 คำนวณพื้นที่สี่เหลี่ยม

```
x = float(input("กรุณาป้อนความยาว = "))
y = float(input("กรุณาป้อนความกว้าง = "))
z = x * y
```

```
print("พื้นที่สี่เหลี่ยมผืนผ้า = ", z)
```

```
กรุณาป้อนความยาว = 5
กรุณาป้อนความกว้าง = 4
พื้นที่สี่เหลี่ยมผืนผ้า = 20.0
```

Process finished with exit code 0



ตัวอย่างที่ 5.2 คำนวณรายได้สุทธิ

```
name = str(input("ชื่อพนักงาน = "))
salary = int(input("เงินเดือน = "))
OT = int(input("จำนวนชั่วโมงที่ทำงานล่วงเวลา = "))
cal_ot = OT * 20
bonus = salary * 0.5
ot_bonus = cal_ot + bonus
salary_total = salary + ot_bonus
print ("ค่าทำงานล่วงเวลา %d * 20 = %.2f" %(OT,cal_ot), "บาท")
print ("ค่า bonus %d * 0.5 = %.2f" %(salary,bonus), "บาท")
print ("รวม OT และ bonus %.2f + %.2f = %.2f" %(cal_ot,bonus,ot_bonus),"บาท")
print ("รายได้สุทธิ %d + %.2f = %.2f" %(salary,ot_bonus,salary_total), "บาท")
```

```
ชื่อพนักงาน = lawan
เงินเดือน = 10000
จำนวนชั่วโมงที่ทำงานล่วงเวลา = 20
ค่าทำงานล่วงเวลา 20 * 20 = 400.00 บาท
ค่า bonus 10000 * 0.5 = 5000.00 บาท
รวม OT และ bonus 400.00 + 5000.00 = 5400.00 บาท
รายได้สุทธิ 10000 + 5400.00 = 15400.00 บาท
```

Process finished with exit code 0

2. คำสั่งควบคุมแบบมีเงื่อนไข(Condition Control Statement)

■ คำสั่ง if

รูปแบบ

```
if เงื่อนไข:
    คำสั่งต่างๆ
```



ตัวอย่างที่ 5.3 if statement:

```
a = 33  
b = 200  
if b > a:  
    print("b is greater than a")
```

output

```
b is greater than a
```

ในตัวอย่างนี้เราใช้ตัวแปรสองตัว คือ a และ b ซึ่งใช้เป็นส่วนหนึ่งของคำสั่ง if เพื่อทดสอบว่า b มากกว่า a หรือไม่ เมื่อ a เป็น 33 และ b คือ 200 เรารู้ว่า 200 มีค่ามากกว่า 33 ดังนั้นเราจะพิมพ์ "b มากกว่า a"



ตัวอย่างที่ 5.4 if statement:

```
var = 100  
if ( var == 100 ):  
    print ("Value of expression is 100")  
    print ("Good bye!")
```

output

```
Value of expression is 100  
Good bye!
```



ตัวอย่างที่ 5.5 รับค่าตัวเลขมา 2 ครั้งแล้วตรวจสอบว่าเป็นเลขคู่หรือเลขคี่ (ถ้าหาร 2 ลงตัวแสดงว่าเป็นเลขคู่):

```
odd = 'เลขคี่'  
even = 'เลขคู่'  
  
n = int(input('จำนวนที่ #1: '))  
s = odd  
if n % 2 == 0:  
    s = even  
  
print(f'{n} เป็น {s}')
```

```
n = int(input("จำนวนที่ #2: "))
s = odd
if n % 2 == 0:
    s = even

print(f'{n} เป็น {s}'")
```

จำนวนที่ #1: 77

77 เป็น เลขคี่

จำนวนที่ #2:

คำสั่ง if...else

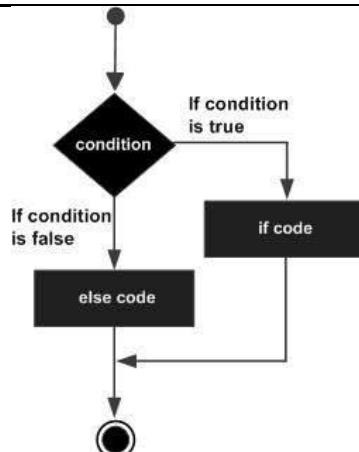
คำสั่ง if...else ควบคุมการทำงานแบบมีเงื่อนไข 2 ทิศทาง

If เงื่อนไข:

คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่กำหนด

else:

คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขไม่ตรงที่กำหนด



ภาพที่ 5 แสดงผังงานคำสั่ง if - else



ตัวอย่างที่ 5.6 เปรียบเทียบค่าตัวแปร a และ b

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

output

b is not greater than a



ตัวอย่างที่ 5.7 รับข้อมูล Password และ Confirm Password และตรวจสอบว่าตรงกันหรือไม่

```
pw1 = input('Password: ')
pw2 = input('Confirm Password: ')

if pw1 != pw2:
    print('Password not match!')
else:
    print('\nPassword OK')
```

Output

```
Password: 1234
Confirm Password: 1234

Password OK

Process finished with exit code 0
```

```
Password: 4321
Confirm Password: 1234
Password not match!

Process finished with exit code 0
```

ภายในบล็อกคำสั่ง if หรือ else ต้องมีอย่างน้อย 1 คำสั่ง แต่หากยังมีพร้อมที่จะกำหนดคำสั่งไดๆ ในขณะนั้น เช่น อยู่ระหว่างทดสอบ ก็อาจใช้คีย์เวิร์ด pass เพื่อข้ามบล็อกคำสั่งนั้นไปก่อน เช่น

```
x = -1
if x >= 0:
    print ('OK')
else:
    pass
```

■ คำสั่ง if...elif...else

ถ้าเงื่อนไขที่เราต้องการตรวจสอบสามารถแยกได้หลายกรณี แต่มีเพียงกรณีเดียวเท่านั้นที่ถูกเลือก เพื่อดำเนินการ ลักษณะเช่นนี้สามารถตรวจสอบด้วยคำสั่ง elif ร่วมกับ if ได้ดังรูปแบบต่อไปนี้

```
If เงื่อนไขที่ 1:
    คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่ 1
elif เงื่อนไขที่ 2:
    คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่ 2
...
elif เงื่อนไขที่ n:
    คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่ n
```



ตัวอย่างที่ 5.8 รับตัวเลขทางคีย์บอร์ดเป็นตัวเลข 0-4 และใช้ if...elif...else ตรวจสอบแล้วแสดงผล

```
print("Please enter the values from 0 to 4")
x=int(input("Enter a number: "))

if x==0:
    print("You entered:", x)
elif x==1:
    print("You entered:", x)
elif x==2:
    print("You entered:", x)
elif x==3:
    print("You entered:", x)
elif x==4:
    print("You entered:", x)
else:
    print("Beyond the range than specified")
```

Output

Please enter the values from 0 to 4

Enter a number: 4

You entered: 4



ตัวอย่างที่ 5.9 รับตัวเลขทางคีย์บอร์ดเป็นจำนวนประดุ ทีมเจ้าบ้านและทีมเยือน และใช้ if...elif ตรวจสอบว่าใครเป็นผู้ชนะ

```
home_goals = int(input('ประดุที่ทีมเจ้าบ้านทำได้: '))
guest_goals = int(input('ประดุที่ทีมเยือนทำได้: '))

if home_goals > guest_goals:
    print('ผล: ทีมเจ้าบ้านชนะ')
elif home_goals < guest_goals:
    print('ผล: ทีมเยือนชนะ')
elif home_goals == guest_goals:
    print('ผล: เสมอทั้งนั้น')
```

ประดุที่ทีมเจ้าบ้านทำได้: 5

ประดุที่ทีมเยือนทำได้: 3

ผล: ทีมเจ้าบ้านชนะ

Process finished with exit code 0



เปลี่ยนหน่วยขนาดไฟล์ให้เหมาะสม โดยรับข้อมูลเข้ามาในหน่วย ไบต์ และเปรียบเทียว่าสามารถเปลี่ยนเป็นหน่วยใหญ่ที่สุดได้ โดยจำนวนไบต์ของแต่ละหน่วยเป็นดังตาราง

หน่วย	จำนวนไบต์
KB	1,024
MB	1,048,576
GB	1,073,741,824
TB	1,099,511,627,776



ตัวอย่างที่ 5.10 รับค่าจากคีย์บอร์ดแล้วแปลงหน่วยข้อมูล

```
size = float(input('ขนาดของไฟล์ (ไบต์) >> '))
unit = ""

if size >= 1_099_511_627_776: #ถ้าขนาด 1,099,511,627,776 ขึ้นไป ให้แปลงเป็น
    หน่วย TB
    size /= 1_099_511_627_776
    unit = 'TB'
elif size >= 107_3741_824:      #ถ้าขนาด 107,3741,824 ขึ้นไป ให้แปลงเป็นหน่วย GB
    size /= 1_073_741_824
    unit = 'GB'
elif size >= 1_048_576:         #ถ้าขนาด 1,048,576 ขึ้นไป ให้แปลงเป็นหน่วย MB
    size /= 1_048_576
    unit = 'MB'
elif size >= 1_024:            #ถ้าขนาด 1,024 ขึ้นไป ให้แปลงเป็นหน่วย KB
    size /= 1_024
    unit = 'KB'
elif size < 1_024:             #ถ้าขนาดน้อยกว่า 1,024 ให้เป็นหน่วย Byte เช่นเดิม
    unit = 'Byte(s)'

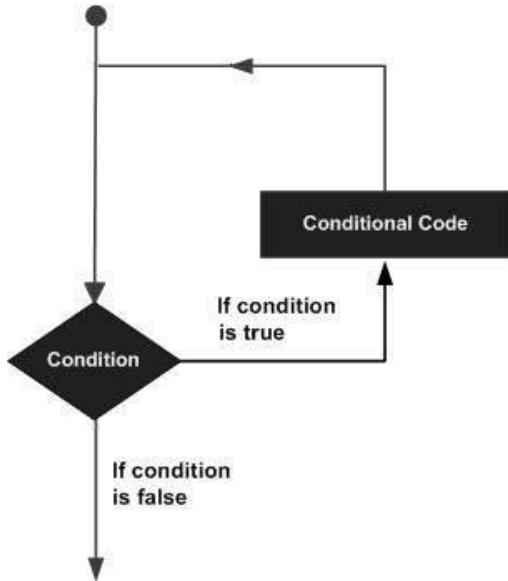
print(f'แปลงเป็นหน่วยที่เหมาะสมได้ประมาณ: {format(size, ".2f")}{unit}')
```

```
ขนาดของไฟล์ (ไบต์) >> 2222222222
แปลงเป็นหน่วยที่เหมาะสมได้ประมาณ: 2.07 GB
```

```
Process finished with exit code 0
```

3. คำสั่งควบคุมแบบทำซ้ำ (Iteration Control Statement)

ในภาษาไพทอน มีคำสั่งทำซ้ำอยู่ 2 คำสั่ง คือ while loops และ for loops



ภาพที่ 6 แสดงผังงานคำสั่งแบบทำซ้ำ

3.1 คำสั่งวนซ้ำด้วย while loop

While loop เราสามารถเรียกใช้ชุดคำสั่งได้ติดๆ กันโดยไม่ต้องมีจังหวะใดๆ ระหว่างคำสั่ง

while เงื่อนไข:

 คำสั่งต่างๆ



ตัวอย่างที่ 5.11 คำสั่งทำซ้ำแบบ While

```
i = 1
while i < 6:
    print(i)
    i += 1
```

output

```
1
2
3
4
5
```



■ The break Statement



ตัวอย่างที่ 5.12 คำสั่ง break เราสามารถหยุดลูปได้เมื่อเงื่อนไข while จะเป็นจริง:

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

output

```
1
2
3
```

■ The continue Statement

continue statement สามารถหยุดการวนซ้ำ และดำเนินการต่อในขั้นตอนต่อไป:



ตัวอย่างที่ 5.13 ทำซ้ำต่อไปถ้าค่า i = 3

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

output

```
1
2
4
5
6
```

■ The else Statement

คำสั่ง else สามารถเรียกใช้บล็อกโค้ด เมื่อเงื่อนไขไม่เป็นจริง:



ตัวอย่างที่ 5.14 พิมพ์ข้อความเมื่อเงื่อนไขเป็นเท็จ:

```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")
```

output

```
1
2
3
4
5
i is no longer less than 6
```

3.2 คำสั่งวนซ้ำด้วย for loop



ตัวอย่างที่ 5.15 พิมพ์ fruit ใน fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

output

```
apple
banana
cherry
```



■ Looping Through a String



ตัวอย่างที่ 5.16 วนลูปสตริง

```
for x in "banana":  
    print(x)
```

output

```
b  
a  
n  
a  
n  
a
```

■ The break Statement



ตัวอย่างที่ 5.17 break Statement

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)  
    if x == "banana":  
        break
```

output

```
apple  
banana
```



ตัวอย่างที่ 5.18 break Statement

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    if x == "banana":  
        break  
    print(x)
```

output

```
apple
```

■ The continue Statement



ตัวอย่างที่ 5.19 continue

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        continue
    print(x)
```

output

apple
cherry

■ The range() Function

ในการวนซ้ำชุดรหัสตามจำนวนครั้งที่กำหนด เราสามารถใช้ฟังก์ชัน `range()` ส่งคืนลำดับของตัวเลขโดยเริ่มจาก 0 ตามค่าเริ่มต้นและเพิ่มขึ้นทีละ 1 (โดยค่าเริ่มต้น) และสิ้นสุดที่ตัวเลขที่ระบุ



ตัวอย่างที่ 5.20 ใช้ฟังก์ชัน range()

```
for x in range(6):
    print(x)
```

output

0
1
2
3
4
5



หมายเหตุ: ช่วง (6)ไม่ใช่ค่า 0 ถึง 6 แต่เป็นค่า 0 ถึง 5

ฟังก์ชัน `range()` มีค่าเริ่มต้นเป็น 0 เป็นค่าเริ่มต้นอย่างไรก็ตามสามารถระบุค่าเริ่มต้นได้โดยการเพิ่มพารามิเตอร์: `range(2, 6)`ซึ่งหมายถึงค่าตั้งแต่ 2 ถึง 6 (แต่ไม่รวม 6):



ตัวอย่างที่ 5.21 ใช้พารามิเตอร์เริ่มต้น:

```
for x in range(2, 6):
    print(x)
```

output

```
2
3
4
5
```

ฟังก์ชัน range () มีค่าเริ่มต้นที่จะเพิ่มลำดับทีละ 1 อย่างไรก็ตามสามารถระบุค่าส่วนเพิ่มได้โดยการเพิ่มพารามิเตอร์ที่สาม: range (2, 30, 3) :



ตัวอย่างที่ 5.22 เพิ่มลำดับด้วย 3 (ค่าเริ่มต้นคือ 1):

```
for x in range(2, 30, 3):
    print(x)
```

output

```
2
5
8
11
14
17
20
23
26
29
```

▪ Else in for Loop

คำหลัก else ใน for loop ระบุลักษณะของรหัสที่จะดำเนินการเมื่อ loop เสร็จสิ้น:



ตัวอย่างที่ 5.23 พิมพ์ตัวเลขตั้งแต่ 0 ถึง 5 และพิมพ์ข้อความเมื่อการวนซ้ำสิ้นสุดลง:

```
for x in range(6):
    print(x)
else:
    print("Finally finished!")
```

output

```
0
1
2
3
4
5
```

Finally finished!

■ Nested Loops

ลูปที่ซ้อนกัน คือ การวนซ้ำภายในลูป "วงใน" จะดำเนินการหนึ่งครั้งสำหรับการวนซ้ำของ "วงนอก" แต่ละครั้ง:



ตัวอย่างที่ 5.24 Nested Loops

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adj:
    for y in fruits:
        print(x, y)
```

Output

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```



4. กิจกรรม

4.1 ให้ตรวจสอบว่าจะต้องสุ่มกี่ครั้ง จึงจะได้เลขที่มีค่าตั้งแต่ 0.9 ขึ้นไป

```
import random

x = 0
times = 0

while x < 0.9:          #ถ้าได้ค่าที่น้อยกว่า 0.9 ต้องวนลูปต่อไป
    x = random.random()  #เลขสุ่มที่ได้จาก random() จะอยู่ระหว่าง 0 - 1
    times += 1

print('ต้องสุ่มทั้งหมด {times} ครั้ง เพื่อให้ได้ค่าตั้งแต่ 0.9 ขึ้นไป')
```

ต้องสุ่มทั้งหมด 32 ครั้ง เพื่อให้ได้ค่าตั้งแต่ 0.9 ขึ้นไป

Process finished with exit code 0

4.2 เขียนโปรแกรมเพื่อรับค่ารหัสจากผู้ใช้ ถ้ารหัสคือ 1234 ด้วยฟังก์ชัน input() หากใส่รหัสผิดให้ใส่ใหม่จนกว่าจะถูกต้อง โดยใช้ลูป while ในการรับข้อมูล

```
validCode = False

while not validCode:      #ถ้าตัวแปร validCode ยังเป็น false ก็ให้วนลูปต่อไป
    code = input('กรุณาระบุรหัส >> ')
    if code == '1234':
        validCode = True  #ถ้าใส่รหัสถูก เปลี่ยนค่า validCode เป็น true เพื่อออกจากลูป
while

print('คุณใส่รหัสถูกต้อง')
```

กรุณาระบุรหัส >> 1111
กรุณาระบุรหัส >> 1234
คุณใส่รหัสถูกต้อง

Process finished with exit code 0



4.3 แยกตัวเลขของมาทีละตัว เช่น 2520 แยกเป็น 2,5,2,0

```

n = int(input('กรุณาใส่ตัวเลขจำนวนเต็ม >> '))
remain = n
digit = 0
result = ""

#จะใช้เทคนิคการนำตัวเลขที่คัดแยกในลูปก่อนๆ นี้
#มาวางต่อท้ายเลขที่คัดแยกได้ในลูปนี้ โดยคั่นด้วยเครื่องหมาย ','
while remain != 0:
    digit = remain % 10
    if result == "":
        result = str(digit)
    else:
        result = str(digit) + ', ' + result

    remain = remain // 10

print('ตัวเลขที่แยกได้คือ: ', result)

```

กรุณาใส่ตัวเลขจำนวนเต็ม >> 2520
ตัวเลขที่แยกได้คือ: 2, 5, 2, 0

Process finished with exit code 0

4.4 สร้างเกมทายตัวเลข

- เมื่อเริ่มเกมให้สุ่มตัวเลข 0-99
- ใช้loop while หรือ for ก็ได้ เพื่อแสดง input ให้ผู้เล่นทายตัวเลขที่ต้องการทาย
- ถ้าผู้เล่นใส่เลขที่มากกว่าค่าที่สุ่ม ให้แจ้งว่า น้อยกว่านี้
- ถ้าผู้เล่นใส่เลขที่น้อยกว่าค่าที่สุ่ม ให้แจ้งว่า มากกว่านี้
- ทายได้ไม่เกิน 7 ครั้ง วนลูปไปจนกว่าจะชนะ ทายครบบังทีไม่ถูกถือว่าแพ้

```

import random

win = False
number = random.randint(0, 100)
count = 1
max_guess = 7
message = ""

```



```
while not win:
```

```
    guess_num = int(input(
```

```
        f'หมายเลขที่: {count} กรุณาใส่ตัวเลข 0 - 99 >> '))
```

```
    if guess_num > number:
```

```
        message = 'น้อยกว่านี้'
```

```
    elif guess_num < number:
```

```
        message = 'มากกว่านี้'
```

```
    elif guess_num == number:
```

```
        message = 'ถูกต้อง'
```

```
        win = True
```

```
    print(message)
```

```
    if (not win) and (count == max_guess):
```

```
        print(f'\nคุณพยายาม {max_guess} ครั้งแล้วไก่เงยบินเลิก')
```

```
        break
```

```
    count += 1
```

```
หมายเลข: 1 กรุณาใส่ตัวเลข 0 - 99 >> 10
```

```
มากกว่านี้
```

```
หมายเลข: 2 กรุณาใส่ตัวเลข 0 - 99 >> 15
```

```
มากกว่านี้
```

```
หมายเลข: 3 กรุณาใส่ตัวเลข 0 - 99 >> 30
```

```
มากกว่านี้
```

```
หมายเลข: 4 กรุณาใส่ตัวเลข 0 - 99 >> 50
```

```
มากกว่านี้
```

```
หมายเลข: 5 กรุณาใส่ตัวเลข 0 - 99 >> 90
```

```
น้อยกว่านี้
```

```
หมายเลข: 6 กรุณาใส่ตัวเลข 0 - 99 >> 80
```

```
มากกว่านี้
```

```
หมายเลข: 7 กรุณาใส่ตัวเลข 0 - 99 >> 85
```

```
น้อยกวานี้
```

```
คุณพยายาม 7 ครั้งแล้ว
```

```
ไก่เงยบินเลิก
```

```
Process finished with exit code 0
```

4.5 ใช้ลูป for วนรับตัวเลข 4 จำนวนจากผู้ใช้ พร้อมหาผลรวมและค่าเฉลี่ย

```
sum = 0
```

```
for i in range(1, 5):
```

```
    num = int(input(f'จำนวนที่ {i}: '))
```

```
    sum += num
```

```
print('ผลรวม:', sum)
```

```
print('ค่าเฉลี่ย:', sum / 4)
```

```

จำนวนที่ 1: 5
จำนวนที่ 2: 10
จำนวนที่ 3: 15
จำนวนที่ 4: 20
ผลรวม: 50
ค่าเฉลี่ย: 12.5

Process finished with exit code 0

```

4.6 ใช้ลูป for เพื่อคำนวณหาเลขยกกำลัง หากได้ผลลัพธ์เป็นจำนวนเต็มก็จะตัดเศษทิ้ง

```

import math

base = float(input('เลขฐาน: '))
power = int(input('เลขชี้กำลัง (จำนวนเต็มบวก): '))

if power >= 0:
    result = 1
    for i in range(0, power):
        result *= base
    """
    เนื่องจากเลขฐานและผลลัพธ์เป็นชนิด float ดังนั้นหากใส่หรือได้ผลลัพธ์เป็นเลขจำนวน
    เต็ม
    เมื่อแสดงเป็นเลขที่มีทศนิยม(.0) ซึ่งเราอาจปรับแต่งการแสดงผลใหม่
    ให้ตรงตามข้อมูลจริง ด้วยการตรวจสอบดังต่อไปนี้
    """

    #ถ้าหารด้วย 1 ลงตัว แสดงว่าเป็นจำนวนเต็ม (หรือทศนิยมเป็น .0) หรือใช้วิธีอื่นก็ได้
    if result % 1 == 0:
        result = math.trunc(result)

    if base % 1 == 0:
        base = math.trunc(base)

    print(f'{base} ** {power} = {format(result, ",")}')
else:
    print('กรุณาใส่เลขชี้กำลังเป็นจำนวนเต็มบวก')

Process finished with exit code 0

```



4.7 สร้างตารางสูตรคูณจากการใช้ลูป for ซ้อนกันสองชั้น

```
text = 'ตารางสูตรคูณ\n'
text += '*****\n'

for i in range(1, 6):      #ลูปชั้นนอก (i) สำหรับตัวเลขซึ่งจัดเรียงในแนวตั้ง

    for j in range(1, 11):    #ลูปชั้นใน (j) สำหรับเลขตัวคูณ
        if j == 1:
            text += " "
        elif i * j < 10:
            text += ' '
        else:
            text += ' '
        text += str(i * j)

    text += '\n'               #ต่อครบทุก j ให้ขึ้น一行ใหม่

print(text)
```

```
ตารางสูตรคูณ
*****
1   2   3   4   5   6   7   8   9   10
2   4   6   8   10  12  14  16  18  20
3   6   9   12  15  18  21  24  27  30
4   8   12  16  20  24  28  32  36  40
5   10  15  20  25  30  35  40  45  50
```

```
Process finished with exit code 0
```



5. แบบฝึกหัดท้ายบท



5.1 จงเขียนโปรแกรมคำนวณหาดัชนีมวลกาย (Body Mass Index: BMI)

สูตร $BMI = \frac{\text{น้ำหนักตัว}}{\text{ส่วนสูง}^2}$

โดยน้ำหนักมีหน่วยเป็นกิโลกรัม และส่วนสูงมีหน่วยเป็นเมตร แล้วเทียบภาวะน้ำหนักตัวดังนี้

BMI	ภาวะน้ำหนักตัว
> 30	โรคอ้วนอันตราย
≥ 25	โรคอ้วน
≥ 23	น้ำหนักเกิน
≥ 18.5	สมส่วน
< 18.5	น้ำหนักต่ำกว่าเกณฑ์

Output

```
น้ำหนัก (กิโลกรัม) >>55
ส่วนสูง (เซ็นติเมตร) >>159
ค่า BMI คือ: 21.76

ภาวะน้ำหนักตัว: สมส่วน

Process finished with exit code 0
```

5.2 จงเขียนโปรแกรมคำนวณภาษีเงินได้บุคคลธรรมดาแบบง่าย โดยที่โจทย์กำหนดให้

รับค่าจ้างเป็นรายเดือนให้คุณด้วย 12 จะได้เป็นเงินเดือนสุทธิ ($30,000 \times 12$)

- หักค่าลดหย่อนส่วนตัว อัตรา 4% สูงสุดไม่เกิน 60,000 บาท
 - หักค่าประกันสังคมลดหย่อนภาษีได้ 5% สูงสุดไม่เกิน 9,000 บาท
- (ประกันสังคมคิดในอัตรา 5% ของเงินเดือน ทุกเดือน) $(30,000 \times 5/100) \times 12 = 15,000$ แต่กฎหมายให้สูงสุดไม่เกิน 9,000 บาท

อัตราภาษีเงินได้บุคคลธรรมดาใหม่	
เงินได้สุทธิต่อปี	อัตราภาษีเงินได้บุคคลธรรมดา
0-150,000 บาท	ได้รับการยกเว้น
150,001-300,000 บาท	5%
300,001-500,000 บาท	10%
500,001-750,000 บาท	15%
750,001-1,000,000 บาท	20%
1,000,001-2,000,000 บาท	25%
2,000,001-5,000,000 บาท	30%
5,000,001 บาทขึ้นไป	35%



Output

```
เงินเดือน >>30000
ภาษีที่ต้องชำระคือ: 7,050.00

Process finished with exit code 0
```

5.3 จงเขียนโปรแกรมเพื่อรับค่าข้อมูลจากคีย์บอร์ดเป็นตัวเลข 0-9 เพียงตัวเดียว จากนั้นแสดงคำอ่านเป็นเลขตัวนั้นพร้อมคำอ่านเป็นเลขไทย โดยอาจใช้ if - elif ในการเปรียบเทียบ เช่น

```
If gigit == 0:
...
elif digit == 1:
...
...
กรุณาใส่ตัวเลข 0-9 เพียงตัวเดียว >>9
9 อ่านว่า: เก้า
เลขไทยคือ: ๙

Process finished with exit code 0
```

5.4 จงเขียนโปรแกรมเพื่อรับค่าข้อมูลจากคีย์บอร์ด ให้รับข้อมูลตัวเลข 0-9 จากนั้นตรวจสอบว่าตัวเลขใดที่มีค่ามากที่สุดและผลรวมของตัวเลข

```
กรุณาใส่ตัวเลขจำนวนเต็ม >> 1235
ตัวเลขที่มีค่ามากที่สุดคือ: 5
ผลรวมของเลขโดดทั้งหมดคือ: 11

Process finished with exit code 0
```

5.5 ถ้าลิฟต์รับน้ำหนักของนักศึกษาไม่เกิน 6 คน ไม่เกิน 450 กิโลกรัม ให้เขียนโปรแกรมเพื่อรับค่าข้อมูลน้ำหนักจากคีย์บอร์ด และถ้าน้ำหนักเกินกว่าที่กำหนดให้แสดงคำแจ้งเตือน ดังตัวอย่างผลลัพธ์

```

ถ้าไม่มีใครเข้าไปในลิฟต์อีก ให้ใส่น้ำหนักเป็น 0
น้ำหนักของคนที่: 1 >>60
น้ำหนักของคนที่: 2 >>70
น้ำหนักของคนที่: 3 >>80
น้ำหนักของคนที่: 4 >>90
น้ำหนักของคนที่: 5 >>100
น้ำหนักของคนที่: 6 >>120
น้ำหนักเกินกำหนด!!!
น้ำหนักของคนที่: 6 >>0
น้ำหนักรวม: 400.0

Process finished with exit code 0

```

5.6 จงสร้างสมาชีกใหม่จากการเรียงลำดับตัวเลขให้ได้รูปแบบเหมือน pattern A และ B โดยใช้ลูป ดังภาพ

1	2	3
1	2	
1		
----- Pattern B -----		
		1
	2	1
	3	2
4	3	2
5	4	3
4	3	2
5	4	3

```

Process finished with exit code 0

```



บทที่ 6

ฟังก์ชันและโมดูล

ในการพัฒนาโปรแกรมขนาดใหญ่และมีซับช้อนมากขึ้น ก็มีความจำเป็นต้องแบ่งขั้นตอนการทำงาน成แต่ละอย่างออกเป็นส่วนย่อยๆ ให้มีหน้าที่การทำงานอย่างเดียวบางหนึ่ง เพื่อเรียกใช้งานในแต่ละส่วน โดยไม่ต้องเขียนโค้ดซ้ำๆ เรียกว่า ฟังก์ชัน (Function) หมายถึง ส่วนของโค้ดหรือโปรแกรมที่ทำงานเพื่อวัตถุประสงค์บางอย่าง ในภาษาไพทอนสามารถสร้างฟังก์ชันเองเพื่อให้ทำงานที่ต้องการในการเขียนโปรแกรมเรามักจะแยกโค้ดที่มีการทำงานเหมือนๆ กันเป็นฟังก์ชันเอาไว้ และเรียกใช้ฟังก์ชันนั้นซ้ำๆ ซึ่งเป็นแนวคิดของการนำโค้ดกลับมาใช้ใหม่ (Code reuse) นี้เป็นรูปแบบของการประยุกต์ใช้ฟังก์ชันในภาษาไพทอน ในบทนี้จะเกี่ยวข้อง ได้แก่ พารามิเตอร์ อาร์กิวเมนต์ และบีต้า และโมดูล

1. ลักษณะของฟังก์ชัน ในภาษาไพทอนแบ่งฟังก์ชันออกเป็น 2 ประเภทหลักๆ คือ

1.1 Pre-defined function (หรือ Built in function) ฟังก์ชันกลุ่มนี้ถูกสร้างไว้ล่วงหน้าจากผู้พัฒนาไพทอนแล้ว และเป็นส่วนหนึ่งของตัวภาษา จึงสามารถเรียกใช้งานได้ทันที เช่น print(), int(), input(), range() เป็นต้น

1.2 User-defined function ฟังก์ชันที่ผู้เขียนโปรแกรมพัฒนาขึ้น ตามวัตถุประสงค์ของตนเองโดยแบ่งฟังก์ชันออกเป็น 2 ประเภทดังนี้

1.2.1 ฟังก์ชันแบบไม่ส่งค่ากลับ (void function) ฟังก์ชันแบบนี้จะกำหนดเพียงการกระทำ แต่ไม่ส่งค่าใดๆ ออกจากฟังก์ชัน

1.2.2 ฟังก์ชันแบบส่งค่ากลับ (value-returning function) ฟังก์ชันแบบนี้จะกำหนดเพียงการกระทำ เมื่อได้ผลลัพธ์ออกมาจะ เราจะส่งค่าที่ได้กลับไปให้ส่วนที่เรียกใช้ฟังก์ชันนั้นๆ ด้วย

2. การสร้างฟังก์ชันแบบไม่ส่งค่ากลับ

ในไพทอนมีการกำหนดฟังก์ชันโดยใช้คำสำคัญ def :

รูปแบบ

```
def ชื่อฟังก์ชัน(พารามิเตอร์) :  
    คำสั่งต่างๆ
```

- Def มาจากคำว่า define เป็นคีย์เวิร์ดในการประกาศเพื่อสร้างฟังก์ชัน
- ชื่อฟังก์ชันมีหลักเกณฑ์คล้ายกับการตั้งชื่อตัวแปร เช่น display , calculate, show_message เป็นต้น
 - พารามิเตอร์(parameter) ข้อมูลที่นำมาใช้ในฟังก์ชัน แต่อาจไม่มีก็ได้ ตามความจำเป็นของงานที่จะทำ

- คำสั่งต่างๆที่กำหนดการกระทำภายในฟังก์ชัน โดยทุกคำสั่งจะต้องเย็บโค้ดเข้าไปให้อยู่ในเดียวกันกับคำสั่งในบล็อก เช่น

```
def my_function():
    print("Hello from a function")
```

- การเรียกใช้ฟังก์ชันให้ใช้ชื่อฟังก์ชันตามด้วย ()



ตัวอย่างที่ 6.1 การเรียกใช้ฟังก์ชัน

```
def my_function():
    print("Hello from a function")
```

```
my_function()
```

Output

```
Hello from a function
```

3. พารามิเตอร์หรืออาร์กิวเม้นต์

พารามิเตอร์ คือ ตัวแปรที่แสดงอยู่ในวงเล็บในนิยามฟังก์ชัน ถ้ามีมากกว่า 1 ตัวให้คั่นด้วย ,
อาร์กิวเม้นต์ คือ ค่าที่ส่งไปยังฟังก์ชันเมื่อถูกเรียกใช้ หรือค่าของพารามิเตอร์ อาร์กิวเม้นต์ถูก
ระบุไว้หลังชื่อฟังก์ชันภายใต้วงเล็บ สามารถเพิ่มอาร์กิวเม้นต์ได้มากเท่าที่ต้องการเพียงคั่นด้วย
เครื่องหมาย , เงื่อนไขของพารามิเตอร์ และอาร์กิวเม้นต์ สามารถใช้สำหรับสิ่งเดียวกัน: ข้อมูลที่
ส่งผ่านไปยังฟังก์ชัน จำนวนอาร์กิวเม้นต์จะต้องเท่ากับจำนวนของพารามิเตอร์

3.1 การจัดลำดับของอาร์กิวเม้นต์(Ordering Arguments)

3.1.1 ลำดับของอาร์กิวเม้นต์ ต้องตรงกับลำดับของของพารามิเตอร์ที่ส่งข้อมูลระหว่างกัน
เมื่อจัดลำดับอาร์กิวเม้นต์ภายใต้ฟังก์ชันหรือการเรียกใช้ฟังก์ชันอาร์กิวเม้นต์จะต้องเกิดขึ้นตามลำดับ
เช่น:

1. Formal positional arguments
2. *args
3. Keyword arguments
4. **kwargs

ตัวอย่างต่อไปนี้มีฟังก์ชัน 1 อาร์กิวเม้นต์ (fname) เมื่อฟังก์ชันถูกเรียกใช้เราจะส่งต่อชื่อจริงซึ่งใช้
ในฟังก์ชันเพื่อพิมพ์ชื่อเต็ม:



ตัวอย่างที่ 6.2 การเรียกใช้ฟังก์ชัน 1 อาร์กิวเมนต์

```
def my_function(fname):  
    print(fname + " Piriya")  
  
my_function("Lawan")  
my_function("Prim")  
my_function("Poom")
```

Output

Lawan Piriya
Prim Piriya
Poom Piriya

โดยค่าเริ่มต้นจะต้องเรียกใช้ฟังก์ชันด้วยจำนวนอาร์กิวเมนต์ที่ถูกต้อง หมายความว่าถ้าฟังก์ชันคาดหวัง 2 อาร์กิวเมนต์ต้องเรียกใช้ฟังก์ชันด้วย 2 อาร์กิวเมนต์ไม่มากและไม่น้อยกว่านี้



ตัวอย่างที่ 6.3 ฟังก์ชันนี้ต้องการ 2 อาร์กิวเมนต์ และรับ 2 อาร์กิวเมนต์:

```
def my_function(fname, lname):  
    print(fname + " " + lname)  
  
my_function("Lawan", "Piriya")
```

Output

Lawan Piriya

- Arguments, * args

หากไม่ทราบจำนวนอาร์กิวเมนต์ที่จะถูกส่งไปยังฟังก์ชันให้เพิ่ม a * ก่อนชื่อพารามิเตอร์ ในนิยามฟังก์ชัน วิธีนี้ฟังก์ชันจะได้รับ tuple ของอาร์กิวเมนต์ และสามารถเข้าถึงรายการตามลำดับ:



ตัวอย่างที่ 6.3 หากไม่ทราบจำนวนอาร์กิวเมนต์ให้เพิ่ม a *หน้าชื่อพารามิเตอร์:

```
def my_function(*kids):  
    print("The youngest child is " + kids[2])  
  
my_function("Malee", "Manee", "Mana")
```

Output

The youngest child is Mana



ตัวอย่างที่ 6.4 arguments, * args

```
def some_args(arg_1, arg_2, arg_3):
    print("arg_1:", arg_1)
    print("arg_2:", arg_2)
    print("arg_3:", arg_3)
args = ("Malee", "Manee", "Mana")
some_args(*args)
```

Output

arg_1: Malee
arg_2: Manee
arg_3: Mana



ตัวอย่างที่ 6.5 arguments, * args

```
def some_args(arg_1, arg_2, arg_3):
    print("arg_1:", arg_1)
    print("arg_2:", arg_2)
    print("arg_3:", arg_3)
my_list = [2, 3]
some_args(1, *my_list)
```

Output

arg_1: 1
arg_2: 2
arg_3: 3



ตัวอย่างที่ 6.6 **kwargs

```
def some_kwargs(kwarg_1, kwarg_2, kwarg_3):
    print("kwarg_1:", kwarg_1)
    print("kwarg_2:", kwarg_2)
    print("kwarg_3:", kwarg_3)
kwargs = {"kwarg_1": "Malee", "kwarg_2": "Manee", "kwarg_3": "Mana"}
some_kwargs(**kwargs)
```

Output

kwarg_1: Malee
kwarg_2: Manee
kwarg_3: Mana



ตัวอย่างที่ 6.7 *args

```
def multiply(*args):
```

```
    z = 1
```

```
    for num in args:
```

```
        z *= num
```

```
    print(z)
```

```
multiply(4, 5)
```

```
multiply(10, 9)
```

```
multiply(2, 3, 4)
```

```
multiply(3, 5, 10, 6)
```

Output

```
20
```

```
90
```

```
24
```

```
900
```

- Keyword Arguments

สามารถส่งอาร์กิวเม้นต์ด้วยไวยากรณ์ `key = value` (ชื่อพารามิเตอร์ = อาร์กิวเม้นต์) วิธีนี้ลำดับในอาร์กิวเม้นต์ไม่มีความสำคัญ



ตัวอย่างที่ 6.8 keyword Arguments

```
def my_function(child3, child2, child1):
```

```
    print("The youngest child is " + child3)
```

```
my_function(child1="Emmy", child2="Tob", child3="Lin")
```

Output

```
The youngest child is Lin
```

- Keyword Arguments, **kwargs

หากไม่ทราบจำนวน Keyword Argument ที่จะถูกส่งไปยังฟังก์ชันให้เพิ่มเครื่องหมายดอกจั่นสองตัว: `**ก่อนชื่อพารามิเตอร์` ในชื่อกำหนดฟังก์ชัน



ตัวอย่างที่ 6.9 **kwargs

```
def print_values(**kwargs):
    for key, value in kwargs.items():
        print("The value of {} is {}".format(key, value))

print_values(
    name_1="Prim",
    name_2="Poom",
    name_3="Par",
    name_4="Peach",
    name_5="Pony",
    name_6="Paint"
)
```

Output

The value of name_1 is Prim
 The value of name_2 is Poom
 The value of name_3 is Par
 The value of name_4 is Peach
 The value of name_5 is Pony
 The value of name_6 is Paint

ด้วยวิธีนี้ฟังก์ชันจะได้รับ dictionary ของอาร์กิวเมนต์และสามารถเข้าถึงรายการตามลำดับ:



ตัวอย่างที่ 6.10 หากไม่ทราบจำนวน Keyword Argument ให้เพิ่ม **หน้าชื่อพารามิเตอร์:

```
def my_function(**kid):
    print("His last name is " + kid["lname"])
my_function(fname = "Poompat", lname = "Piriya")
```

Output

His last name is Piriya

- **Default Parameter Value**

หากเราเรียกใช้ฟังก์ชันโดยไม่มีอาร์กิวเมนต์จะใช้ค่าเริ่มต้น(default):

- รูปแบบ ชื่อพารามิเตอร์ = ค่าเดิม



- หากในฟังก์ชันเดียวกัน มีพารามิเตอร์ทั้งแบบมีค่าดีฟอลต์ และแบบไม่มีค่าดีฟอลต์ ต้องจัดให้ตัวที่ไม่มีค่าดีฟอลต์อยู่ก่อนเสมอ



ตัวอย่างที่ 6.11 ค่าดีฟอลต์พารามิเตอร์

```
def my_function(country = "Thailand"):
    print("I am from " + country)

my_function("Lao")
my_function("Cambodia")
my_function()
my_function("Myanmar")
```

Output

```
I am from Lao
I am from Cambodia
I am from Thailand
I am from Myanmar
```

• Passing a List as an Argument

สามารถส่งประเภทข้อมูลของอาร์กิวเมนต์ ไปยังฟังก์ชัน (string, number, list, dictionary เป็นต้น) และจะถือว่าเป็นประเภทข้อมูลเดียวกันภายในฟังก์ชัน เช่น ถ้าส่ง List เป็นอาร์กิวเมนต์มันจะยังคงเป็น List เมื่อถึงฟังก์ชัน:



ตัวอย่างที่ 6.12 ส่งประเภทข้อมูลของอาร์กิวเมนต์ไปยังฟังก์ชัน

```
def my_function(food):
    for x in food:
        print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Output

```
apple
banana
cherry
```

4. พังก์ชันแบบส่งค่ากลับ

หากต้องการให้พังก์ชันส่งคืนค่าให้ใช้คำสั่ง `return`:

รูปแบบ

`def` ชื่อพังก์ชัน (พารามิเตอร์) :

คำสั่งต่างๆ ภายในพังก์ชัน
`return` ข้อมูลที่จะส่งกลับ



ตัวอย่างที่ 6.13 คำสั่ง `return`:

```
def my_function(x):
    return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Output

15
25
45

พังก์ชันไม่สามารถเป็นค่าว่างได้ แต่สามารถนิยามพังก์ชันแบบปัสเมื่อหาได้ โดย `pass` statement ให้ใส่คำสั่ง `pass` เพื่อหลีกเลี่ยงข้อผิดพลาด

```
def myfunction():
    pass
```

5. การเรียกใช้พังก์ชันแบบ Recursion

ไฟฟอนยังยอมรับการเรียกซ้ำของพังก์ชันซึ่งหมายความว่าพังก์ชันที่กำหนดไว้สามารถเรียกใช้ตัวเองได้ การเรียกซ้ำเป็นแนวคิดทางคณิตศาสตร์ และการเขียนโปรแกรมทั่วไป หมายความว่า พังก์ชันเรียกตัวเอง สิ่งนี้มีประโยชน์ในความหมายที่สามารถวนลูปข้อมูลเพื่อให้ได้ผลลัพธ์

นักเขียนโปรแกรมควรระมัดระวังอย่างยิ่งกับการเรียกซ้ำเนื่องจากอาจเป็นเรื่องง่ายที่จะเขียนพังก์ชันที่ไม่มีวันยุติหรือใช้หน่วยความจำหรือพลังงานโปรแกรมมากเกินไป อย่างไรก็ตามเมื่อเขียนซ้ำอย่างถูกต้องอาจเป็นวิธีการเขียนโปรแกรมที่มีประสิทธิภาพ



ในตัวอย่างนี้ tri_recursion () เป็นฟังก์ชันที่เราได้กำหนดไว้เพื่อเรียกตัวเองว่า ("recurse") เราใช้ตัวแปร k เป็นข้อมูลซึ่งจะลดลง (-1) ทุกครั้งที่เราเรียกคืน การเรียกซ้ำจะสิ้นสุดเมื่อเงื่อนไขไม่เกิน 0 (กล่าวคือเมื่อเป็น 0)



ตัวอย่างที่ 6.14 การเรียกซ้ำ (Recursion)

```
def tri_recursion(k):
    if(k > 0):
        result = k + tri_recursion(k - 1)
        print(result)
    else:
        result = 0
    return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```

Output

Recursion Example Results

1
3
6
10
15
21

6. ฟังก์ชันแบบแอลมป์ดา (Lampda function)

นอกจากการสร้างฟังก์ชันด้วยคำสั่ง def ที่กล่าวมาแล้วยังมีฟังก์ชันแบบ แอลมป์ดา โดยมีลักษณะดังนี้

- ฟังก์ชันแอลมป์ดาเป็นฟังก์ชันที่ไม่มีชื่อ อาจเรียกได้ว่า anonymous function
- ฟังก์ชันแอลมป์ดาสามารถรับอาร์กิวเมนต์จำนวนเท่าใดก็ได้ แต่สามารถมีได้เพียงนิพจน์เดียว

รูปแบบ

```
lambda arguments : expression
```



ตัวอย่างที่ 6.15 นิพจน์ลูกดำเนินการและผลลัพธ์จะถูกส่งกลับ: เพิ่ม 10 ในอาร์กิวเม้นต์ a และส่งกลับผลลัพธ์: พิ้งก์ชัน Lambda สามารถรับอาร์กิวเม้นต์จำนวนเท่าใดก็ได้:

```
x = lambda a : a + 10
print(x(5))
```

Output

15



ตัวอย่างที่ 6.16 คูณอาร์กิวเม้นต์ a ด้วยอาร์กิวเม้นต์ b และส่งกลับผลลัพธ์:

```
x = lambda a, b : a * b
print(x(5, 6))
```

Output

30



ตัวอย่างที่ 6.17 สรุปอาร์กิวเม้นต์ a, b และ c และส่งกลับผลลัพธ์:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

Output

13

เหตุใดจึงต้องใช้พิ้งก์ชันแลมบ์ดา เพราะแลมบ์ดาจะทำงานได้ดีกว่าเมื่อใช้เป็นพิ้งก์ชันที่ไม่ระบุตัวตนในพิ้งก์ชันอื่น สมมติว่าต้องการนิยามพิ้งก์ชันที่ใช้อาร์กิวเม้นต์เดียว และอาร์กิวเม้นต์นั้นจะคูณด้วยจำนวนที่ไม่รู้จัก:

```
def myfunc(n):
    return lambda a : a * n
```



ตัวอย่างที่ 6.18 ใช้นิยามพิ้งก์ชันนี้เพื่อสร้างพิ้งก์ชันที่จะเพิ่มจำนวนที่ส่งเป็นสองเท่า:

```
def myfunc(n):
    return lambda a : a * n
```



```
mydoubler = myfunc(2)
print(mydoubler(11))
```

Output

22

หรือใช้คำจำกัดความของฟังก์ชันเดียวกันเพื่อสร้างฟังก์ชันที่มีค่าเป็น 3 เท่าของจำนวนที่ส่งเข้ามา:



ตัวอย่างที่ 6.19 สร้างฟังก์ชันที่มีค่าเป็น 3 เท่า

```
def myfunc(n):
    return lambda a : a * n
mytrippler = myfunc(3)
print(mytrippler(11))
```

Output

33

หรือใช้นิยามฟังก์ชันเดียวกันเพื่อสร้างฟังก์ชันทั้ง 2 ในโปรแกรม:



ตัวอย่างที่ 6.20 นิยามฟังก์ชันเดียวกันเพื่อสร้างฟังก์ชัน

```
def myfunc(n):
    return lambda a : a * n
mydoubler = myfunc(2)
mytrippler = myfunc(3)
print(mydoubler(11))
print(mytrippler(11))
```

Output

22

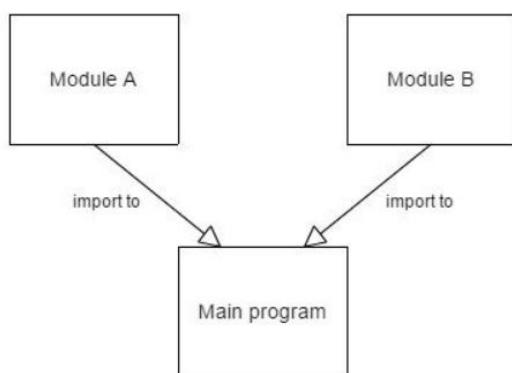
33



ใช้ฟังก์ชันแลนบ์ดาเมื่อต้องใช้ฟังก์ชันที่ไม่ระบุชื่อในช่วงเวลาสั้น ๆ

7. การสร้างและใช้โมดูล

โมดูล (Module) คือ ไฟล์หรือส่วนของโปรแกรมที่ใช้สำหรับกำหนดตัวแปร พังก์ชันหรือคลาสโดยแบ่งย่อยอีกหน่วยหนึ่งจากโปรแกรมหลัก และในโมดูลยังสามารถประกอบไปด้วยคำสั่ง ประมวลผลการทำงานได้ ยกตัวอย่างเช่น เมื่อเขียนโปรแกรมในภาษาไพทอนอาจจะมีพังก์ชันสำหรับทำงานและจัดการกับตัวเลขเป็นจำนวนมาก และในขณะเดียวกัน ไม่ต้องการให้โปรแกรมหลักนั้นมีขนาดใหญ่เกินไป นั่นหมายความว่าสามารถนำพังก์ชันเหล่านี้มาสร้างเป็นโมดูล และในการใช้งานนั้น จะต้องนำเข้ามาในโปรแกรมโดยวิธีที่เรียกว่า Import



จะเห็นว่าโมดูลก็คือการแยกส่วนของโปรแกรมออกไปเป็นอีks่วนและสามารถเรียกใช้ได้เมื่อต้องการ หรือกล่าวอีกนัยหนึ่ง โมดูลก็เหมือนไลบรารีของฟังก์ชันและคลาสต่างๆ นั่นเป็นเพราะว่าเมื่อ โปรแกรมมีขนาดใหญ่ สามารถแบ่งส่วนต่างๆ ของโปรแกรมออกเป็นโมดูลย่อยๆ เพื่อให้ง่ายต่อการ จัดการและการใช้งาน ในภาษาไพทอนโมดูลที่ถูกสร้างขึ้นมาจะเป็นไฟล์ในรูปแบบ `module_name.py` และนอกจากนี้ไพทอนยังมี Built-in module เป็นจำนวนมาก เช่น `math` เป็น โมดูลเกี่ยวกับฟังก์ชันทางคณิตศาสตร์ หรือ `random` เป็นโมดูลเพื่อจัดการและสุ่มตัวเลข เป็นต้น

7.1 การสร้างโมดูลในภาษา Python

ในการสร้างโมดูลในภาษาไพทอนต้องนำโค้ดของโปรแกรม โดยที่ไว้แล้วจะประกอบไปด้วย ตัวแปร พังก์ชัน หรือคลาส ไปรวมไว้ในไฟล์ใหม่ที่ไม่ใช่ไฟล์หลักของโปรแกรม ในรูปแบบ `module_name.py` โดยที่ `module_name` นั้นเป็นชื่อของโมดูลเพื่อนำไปใช้งานในการเขียน โปรแกรม โดยการเรียกใช้ด้วยคำสั่ง `import` ต่อไปมาดูตัวอย่างการสร้างโมดูลในภาษาไพทอน

```

# number.py
def factorial(n): # return factorial value of n
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

def fibonacci(n): # return Fibonacci series up to n
  
```



```
result = []
a, b = 0, 1
while b < n:
    result.append(b)
    a, b = b, a + b
return result
```

ในตัวอย่าง เป็นการสร้างโมดูลโดยไฟล์ของโมดูลนั้นมีชื่อว่า number.py นั้นหมายความว่า โมดูลนี้มีชื่อว่า number ซึ่งนี้เป็นสิ่งที่เราจะใช้สำหรับเรียกใช้งานโมดูลในการเขียนโปรแกรม ภายใน โมดูลประกอบไปด้วย 2 ฟังก์ชันที่ทำงานเกี่ยวกับตัวเลข ฟังก์ชัน factorial() เป็นฟังก์ชันสำหรับหาค่า Factorial ของตัวเลขจำนวนเต็ม n ซึ่งเป็น Recursive function และฟังก์ชัน fibonacci() ใช้หา ลำดับของ Fibonacci จนถึงจำนวนเต็ม n

7.2 การนำเข้าโมดูลด้วยคำสั่ง import

หลังจากที่เราได้สร้างโมดูลไปแล้ว ต่อไปจะเป็นการนำโมดูลดังกล่าวมาใช้งาน ในภาษาไฟฟอนนั้นจะใช้คำสั่ง import เพื่อนำเข้าโมดูลเพื่อนำมาใช้งานในโปรแกรม มาดูตัวอย่างการใช้งาน โมดูล number ในตัวอย่างก่อนหน้า นี้เป็นโค้ดของโปรแกรม



ตัวอย่างที่ 7.1 การนำเข้าโมดูลด้วยคำสั่ง import

```
import number
print('5! = ', number.factorial(5))
print(number.fibonacci(100))
```

Output

```
5! = 120
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

7.2.1 การนำเข้าโมดูลด้วยคำสั่ง from...import

ในการใช้งานคำสั่ง import นั้นจะเป็นการนำเข้าออบเจ็คทั้งหมดในโมดูลเข้ามายัง โปรแกรม และการใช้งานฟังก์ชันหรือออบเจ็คภายในโมดูลจะต้องนำหน้าด้วยชื่อโมดูลเสมอ ในภาษาไฟฟอนนั้นมีคำสั่ง from import สำหรับนำเข้าข้อมูลบางส่วนภายในโมดูล และสามารถใช้งานออบเจ็คได้โดยตรงโดยไม่ต้องมี Prefix ชื่อของโมดูล มาดูตัวอย่างการใช้งาน



ตัวอย่างที่ 7.2 การนำเข้าโมดูลด้วยคำสั่ง from...import

```
from number import factorial
print('5! = ', factorial(5))
print('3! = ', factorial(3))
```

$5! = 120$

$3! = 6$

ในตัวอย่าง เป็นการใช้งานคำสั่ง from ... import เพื่อนำเข้าฟังก์ชันภายในโมดูล number จะเห็นได้ว่าเราได้นำเข้าเพียงฟังก์ชัน factorial() เข้ามาในโปรแกรมและในตอนใช้งานนั้นสามารถใช้ได้โดยที่ไม่ต้องใช้ Prefix ในการเรียกใช้

```
from number import factorial, fibonacci
# or
from number import *
```



ตัวอย่างที่ 7.3 นำเข้าโมดูล random เพื่อสร้างตัวเลขสุ่ม

```
import random

for i in range(10):
    print(random.randint(1, 25))
```

Output

```
18
5
22
8
4
25
13
3
2
15
```



ตัวอย่างที่ 7.4 นำเข้าโมดูล random และ math

```
import random
import math

for i in range(5):
    print(random.randint(1, 25))

print(math.pi)
```

Output

```
5
```



```
17  
19  
17  
2  
3.141592653589793
```

7.2.2 ไมดูลนามแฝง

สามารถแก้ไขชื่อของโมดูลและฟังก์ชันภายในไฟล์โดยใช้คำสำคัญ **as** หากต้องการเปลี่ยนชื่อเนื่องจากได้ใช้ชื่อเดียวกันสำหรับอย่างอื่นในโปรแกรมแล้วโมดูลอื่นที่ import ที่ใช้ชื่อนั้นเช่นกัน หรืออาจต้องการยกชื่อที่ยาวขึ้นซึ่งใช้บ่อยมาก โครงสร้างของคำสั่งนี้มีลักษณะดังนี้:

```
import [module] as [another_name]
```



ตัวอย่างที่ 7.5 ไมดูลนามแฝง

```
import math as m  
  
print(m.pi)  
print(m.e)
```

Output
3.141592653589793
2.718281828459045

8. กิจกรรม

8.1 จงเขียนโปรแกรมเพื่อสร้างฟังก์ชันเดียวเพื่อรับพารามิเตอร์เป็นจำนวนหลักที่จะอกรางวัลจากนั้นส่งค่ากลับออกໄປ

```
import random

def rand_num(length):
    result = ""
    for n in range(0, length):
        r = random.randrange(0, 9)
        result += str(r)

    return result
#-----
print(
    'รางวัลที่ 1:', rand_num(6), '\n',
    'รางวัลเลขหน้า 3 ตัว:', rand_num(3), ", ", rand_num(3), '\n',
    'รางวัลเลขท้าย 3 ตัว:', rand_num(3), ", ", rand_num(3), '\n',
    'รางวัลเลขท้าย 2 ตัว:', rand_num(2), '\n',
    end=""
)
```

C:\Users\HPSurajak\PycharmProjects\

```
รางวัลที่ 1: 746527
รางวัลเลขหน้า 3 ตัว: 644 006
รางวัลเลขท้าย 3 ตัว: 515 136
รางวัลเลขท้าย 2 ตัว: 00
```

Process finished with exit code 0

8.2 จงเขียนโปรแกรมเพื่อสร้างฟังก์ชันสำหรับตรวจสอบค่าวันเดือนปีว่าตรงตามหลักการหรือไม่โดยประเด็นสำคัญที่ต้องพิจารณาดังนี้

- อนุญาตให้ใช้เครื่องหมายแบ่งวันเดือนปี อย่างไดอย่างหนึ่งระหว่าง / , - , หรือ . เท่านั้น
- รูปแบบที่ยอมรับ คือ ต้องเรียงลำดับและอยู่ในช่วงวันที่ (1-31) เดือน (1-12) ปี ค.ศ. (1-9999)

```
#ฟังก์ชันสำหรับตรวจสอบว่าเป็นปี leap year หรือไม่ ซึ่งเงื่อนไขคือ
#(หารด้วย 400 ลงตัว) หรือ (หารด้วย 4 ลงตัว แต่หารด้วย 100 ไม่ลงตัว)
def leapyear(year):
```



```
if year % 400 == 0 or (year % 4 == 0 and year % 100 != 0):
    return True
else:
    return False

#ตรวจสอบว่าวันเดือนปี โดยคัดแยกแต่ละส่วนออกจากกัน
def check_date(dd_mm_yyyy):
    datestr = str(dd_mm_yyyy)
    sep = ""
    if datestr.count('/') != 0:
        sep = '/'
    elif datestr.count('-') != 0:
        sep = '-'
    elif datestr.count('.') != 0:
        sep = '.'
    else:
        return 'Error: ต้องแบ่งวันเดือนปีด้วย / หรือ - หรือ .'

    d = datestr.split(sep)
    for (i, x) in enumerate(d):
        if d[i].isdigit():
            d[i] = int(x)
        else:
            return 'Error: วันเดือนปีต้องเป็นตัวเลข'

    #d[0] วันที่, d[1] เดือน, d[2] ปี
    if d[0] not in range(1, 31):
        return 'Error: วันที่ต้องอยู่ระหว่าง 1-31'
    elif d[1] not in range(1, 12):
        return 'Error: เดือนต้องอยู่ระหว่าง 1-12'
    elif d[2] not in range(1, 9999):
        return 'Error: ปีต้องอยู่ระหว่าง 1-9999'
    elif d[1] == 2:
        if d[0] > 28 and not leapyear(d[2]):
            return f'Error: เดือนกุมภาพันธ์ปี {d[2]} มี 28 วัน'
        elif d[0] > 29 and leapyear(d[2]):
            return f'Error: เดือนกุมภาพันธ์ปี {d[2]} มี 29 วัน'
    elif d[0] == 31 and d[1] not in [1, 3, 5, 7, 8, 10, 12]:
```

```

return 'Error: เดือนนี้มีเพียง 30 วัน'

return 'OK: เป็นวันเดือนปีที่ถูกต้อง'

#-----
dt = ['20-02-2002', '20/20/2020', '20.10.2010', '31.4.2000', '29/2/2018']
for x in dt:
    print(f'{x} => {check_date(x)}')

C:\Users\HPSurajak\PycharmProjects\pythonProject
20-02-2002 => OK: เป็นวันเดือนปีที่ถูกต้อง
20/20/2020 => Error: เดือนต้องอยู่ระหว่าง 1-12
20.10.2010 => OK: เป็นวันเดือนปีที่ถูกต้อง
31.4.2000 => Error: วันที่ต้องอยู่ระหว่าง 1-31
29/2/2018 => Error: เดือนกุมภาพันธ์ปี 2018 มี 28 วัน

Process finished with exit code 0

```

8.3 จะเขียนโปรแกรมเพื่อสร้างฟังก์ชันคำนวณปริมาตรของรูปทรง ซึ่งมีพารามิเตอร์เป็น แລมบ์ดา โดยหากเราจะคำนวณปริมาตรเป็นรูปทรงใดๆ เมื่อเราเรียกฟังก์ชันก็จะระบุวิธีการในแบบแລมบ์ดาแล้วกำหนดเป็นอาร์กิวเมนต์ให้กับ แต่เนื่องจากรูปทรงแต่ละแบบจะใช้จำนวนข้อมูลที่แตกต่างดังนั้นก็จะมีพารามิเตอร์อีก

```

def shape_volume(*dimension, func):
    return func(*dimension)

#-----
sphere = shape_volume(10, func=lambda r: (4 / 3) * 3.14 * (r ** 3))
print('sphere volume =', format(sphere, '.2f'))

cuboid = shape_volume(10, 20, 30, func=lambda w, l, h: w * l * h)
print('cuboid volume =', cuboid)

cylinder = shape_volume(10, 20, func=lambda r, h: 3.14 * (r ** 2) * h)
print('cylinder volume =', cylinder)

sphere volume = 4186.67
cuboid volume = 6000
cylinder volume = 6280.0

Process finished with exit code 0

```



8.4 จงเขียนโปรแกรมพื้อฝึก ตอน และแสดงยอดคงเหลือ ดำเนินการอย่างต่อเนื่องจนกว่าผู้ใช้จะกดยกเลิก ขั้นตอนดังนี้

- สร้างเลขสุ่มเพื่อสมมุติว่าเป็นค่าคงเหลือเริ่มแรก
- แสดงเมนูให้เลือกว่าจะฝาก ถอน และแสดงยอดคงเหลือ หรือ ยกเลิก
 1. ถ้าเลือก แสดงยอดคงเหลือ ก็อ่านค่ามาแสดง
 2. ถ้าเลือกยกเลิกก็หยุดการทำงาน
 3. ถ้าเลือก ฝาก หรือ ถอน ให้รับข้อมูลเป็นจำนวนเงิน และนำข้อมูลที่ได้ไปดำเนินการ หลังจากนั้นก็แสดงยอดคงเหลือ
 4. หลังจากแสดงยอดคงเหลือให้นำเมนูกลับมาแสดงอีก และทำซ้ำเช่นนี้ไปเรื่อยๆ

```
import random

balance = random.randint(10_000, 50_000)
menu = 1
amount = 0

def select_menu():
    global menu
    print('\nกรุณาเลือกเมนู')
    menu = input('1: ฝาก, 2: ถอน, 3: ถามยอด, 0: ยกเลิก >>')
    if not menu.isdigit() or int(menu) not in range(1, 4):
        exit(0)
    else:
        menu = int(menu)

    if menu in range(1, 3):
        get_amount()
    elif menu == 3:
        show_balance()

def get_amount():
    global amount
    a = input('จำนวนเงิน >>')
    if not a.isdigit():
        print('ต้องระบุเป็นเลขจำนวนเต็ม')
        get_amount()
    else:
        amount = int(a)
```

```

if menu == 1:
    deposit()
elif menu == 2:
    withdraw()

def deposit():
    global balance
    balance += amount
    show_balance()

def withdraw():
    global balance
    if amount <= balance:
        balance -= amount
        show_balance()
    else:
        print('ยอดคงเหลือไม่เพียงพอ')

def show_balance():
    print('ยอดคงเหลือ: ', format(balance, ","))
    select_menu()

#---- ทดสอบ -----
show_balance()
select_menu()

```

```

ยอดคงเหลือ:  20,395
กรุณาเลือกเมนู
1: ฝาก, 2: ถอน, 3: ถามยอด, 0: ยกเลิก >>1
จำนวนเงิน >>500
ยอดคงเหลือ:  20,895

กรุณาเลือกเมนู
1: ฝาก, 2: ถอน, 3: ถามยอด, 0: ยกเลิก >>2
จำนวนเงิน >>300
ยอดคงเหลือ:  20,595

กรุณาเลือกเมนู
1: ฝาก, 2: ถอน, 3: ถามยอด, 0: ยกเลิก >>3
ยอดคงเหลือ:  20,595

Process finished with exit code 0

```



9. สรุปท้ายบท

ฟังก์ชัน (Function) คือ ส่วนของโค้ดหรือโปรแกรมที่ทำงานเพื่อวัตถุประสงค์บางอย่าง ในภาษาไพทอนสามารถสร้างฟังก์ชันเองเพื่อให้ทำงานที่ต้องการ ในการเขียนโปรแกรมเรามักจะแยกโค้ดที่มีการทำงานเหมือนๆ กันเป็นฟังก์ชันเอาไว้ และเรียกใช้ฟังก์ชันนั้นๆ ซึ่งเป็นแนวคิดของการนำโค้ดกลับมาใช้ใหม่ (Code reuse) นี้เป็นรูปแบบของการประกาศฟังก์ชันในภาษาไพทอน

โมดูล (Module) คือ ไฟล์หรือส่วนของโปรแกรมที่ใช้สำหรับกำหนดตัวแปร ฟังก์ชัน หรือคลาส โดยแบ่งย่อยอีกหน่วยหนึ่งจากโปรแกรมหลัก และในโมดูลยังสามารถประกอบไปด้วยคำสั่ง ประมวลผลการทำงานได้ และเรียกใช้งานได้เมื่อต้องการ นำเข้าโมดูลด้วยสั่ง import และคำสั่ง from import

10. แบบฝึกหัดท้ายบท

10.1 จงสร้างฟังก์ชันในลักษณะดังต่อไปนี้

- random_lowercase_letters(length) เพื่อสร้างอักษร a-z แบบสุ่ม ให้ได้จำนวนตามค่าของพารามิเตอร์ length
- random_uppercase_letters(length) เพื่อสร้างอักษร A-Z แบบสุ่ม ให้ได้จำนวนตามค่าของพารามิเตอร์ length
- random_digits(length) เพื่อสร้างตัวเลข 0-9 แบบสุ่ม ให้ได้จำนวนตามค่าของพารามิเตอร์ length (ตัวแรกต้องไม่เป็น 0)

```
random_lowercase_letters(5) => hwfcz  
random_uppercase_letters(5) => FHQ  
random_digits(5) => 2655  
  
Process finished with exit code 0
```

10.2 จากรูปทรงข้อที่ 4 สร้างฟังก์ชันคำนวณปริมาตรของรูปทรง ซึ่งมีพารามิเตอร์เป็นแลมบ์ดา ให้เปลี่ยนมาเป็นฟังก์ชันคำนวณหาพื้นที่ของรูปทรงแทน โดยมีรูปแบบเป็น shap_area(*.dimension, func)

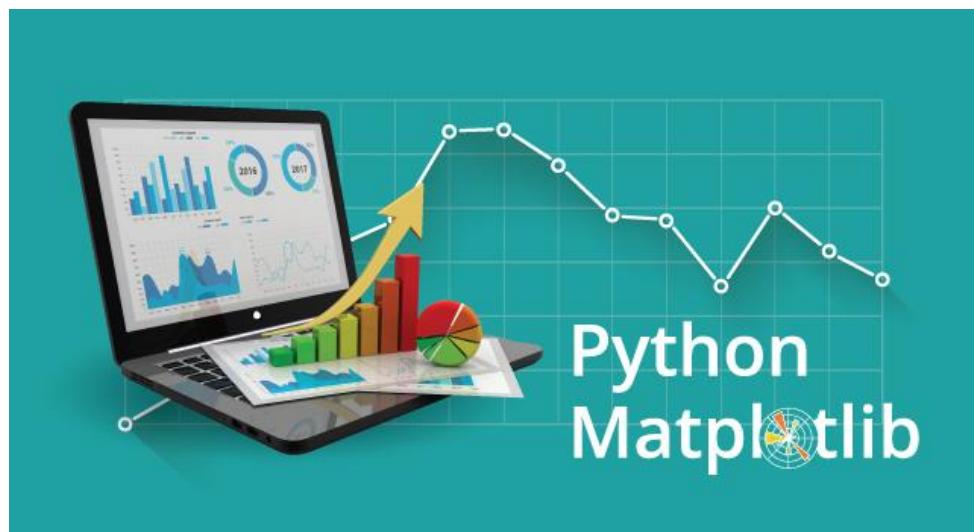
```
circle area (radius: 10) = 314.00  
rect area (width:10, height: 20) = 200  
triangle area (base: 10, height: 20) = 100.0  
  
Process finished with exit code 0
```



บทที่ 7

การสร้างกราฟด้วย Matplotlib

การเขียนโปรแกรมนั้นหลายครั้งเราจะต้องทำงานกับข้อมูลเชิงตัวเลขอยู่เสมอ เช่น การทำเว็บช้อปปิ้งออนไลน์ ก็จะต้องมีข้อมูลราคาของสินค้า, ยอดการสั่งซื้อ หรือจำนวนรายรับที่เข้ามาเป็นของเรา และบางครั้งเราก็อยากที่จะนำตัวเลขเหล่านั้นมาลงพล็อทเป็นกราฟเพื่อดูแนวโน้ม หรือภาพรวมเพื่อนำไปวิเคราะห์อะไรบางอย่าง วันนี้เราจะ 알아จะมานำเสนอการพล็อทกราฟด้วยภาษาไพทอน



1. Matplotlib คืออะไร

Matplotlib เป็นไลบรารีการพล็อทกราฟระดับต่ำในไพทอนที่ทำหน้าที่เป็นยูทิลิตี้การแสดงกราฟ สร้างขึ้นโดย John D. Hunter เป็นโอเพนซอร์สและสามารถใช้งานได้อย่างอิสระ ส่วนใหญ่เขียนด้วยไพทอนบางกลุ่ม เช่นด้วย C, Objective-C และ Javascript สำหรับความเข้ากันได้ของแพลตฟอร์ม

2. Matplotlib Codebase อยู่ที่ไหน

ซอฟต์แวร์โค้ดสำหรับ Matplotlib อยู่ที่ที่เก็บ github <https://github.com/matplotlib/matplotlib>



3. การติดตั้ง Matplotlib (Installation of Matplotlib)

หากมี Python และ PIP ติดตั้งอยู่ในระบบแล้ว การติดตั้ง Matplotlib นั้นง่ายมาก ติดตั้งโดยใช้คำสั่งนี้:

```
C:\Users\Your Name>pip install matplotlib
```

หากคำสั่งนี้ล้มเหลว ให้ติดตั้ง Matplotlib ผ่านโปรแกรมอื่น เช่น Anaconda, Spyder เป็นต้น

4. การนำเข้า Matplotlib (Import Matplotlib)

เมื่อติดตั้ง Matplotlib แล้ว ให้นำเข้าในแอปพลิเคชันโดยเพิ่ม import module statement:

```
Import Matplotlib
```

ตอนนี้ Matplotlib ถูกนำเข้าและพร้อมใช้งานแล้ว:

5. การตรวจสอบเวอร์ชัน Matplotlib (Checking Matplotlib Version)

สร้างไฟล์ Python ใหม่โดยใส่คำสั่งดังนี้:



ตัวอย่างที่ 7.1 ตรวจสอบเวอร์ชัน Matplotlib

```
import matplotlib
print(matplotlib.__version__)
```

Output

3.2.1

หมายเหตุ: ใช้อักษรขีดล่างสองตัว (underscore characters) ใน __version__.

6. การสร้างกราฟด้วย Matplotlib

- 6.1 Matplotlib Pyplot
- 6.2 Matplotlib Plotting
- 6.3 Matplotlib Markers
- 6.4 Matplotlib Line
- 6.5 Matplotlib Labels
- 6.6 Matplotlib Grid
- 6.7 Matplotlib Subplots
- 6.8 Matplotlib Scatter

- 6.9 Matplotlib Bars
- 6.10 Matplotlib Histograms
- 6.11 Matplotlib Pie Charts

6.1 Matplotlib Pyplot

Pyplot ยูทิลิตี้ Matplotlib ส่วนใหญ่อยู่ภายใต้โมดูลชื่อ pyplot และมักจะนำเข้าภายใต้ชื่อ plt

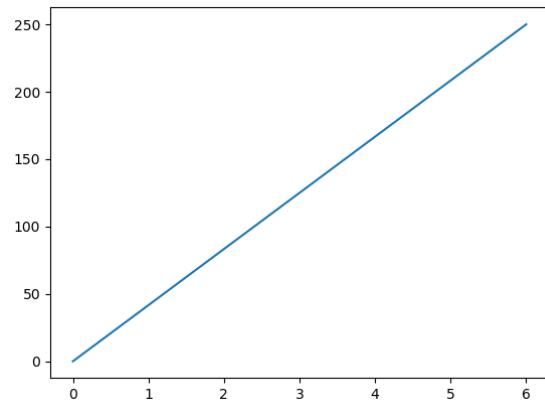
```
import matplotlib.pyplot as plt
```

ตอนนี้ Pyplot package สามารถอ้างถึงด้วย plt



ตัวอย่างที่ 7.2 ลากเส้นในโค้ดограмจากตำแหน่ง (0,0) ไปยังตำแหน่ง (6,250)

```
import matplotlib.pyplot as plt  
import numpy as np  
  
xpoints = np.array([0, 6])  
ypoints = np.array([0, 250])  
  
plt.plot(xpoints, ypoints)  
plt.show()
```





6.2 Matplotlib Plotting

6.2.1 กราฟจุด x และ y

ฟังก์ชัน `plot()` ที่ใช้ในการวาดจุด (เครื่องหมาย) ในแผนภาพ โดยค่าเริ่มต้นฟังก์ชัน `plot()` จะลากเส้นจากจุดหนึ่งไปยังอีกจุดหนึ่ง ฟังก์ชันใช้พารามิเตอร์เพื่อบรุจุดในไดอะแกรม

พารามิเตอร์1 เป็นอาร์เรย์ที่มีจุดที่แกน x (x-axis)

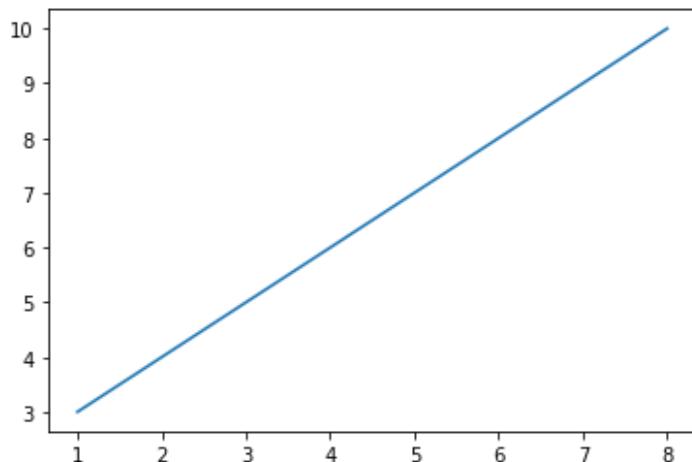
พารามิเตอร์2 เป็นอาร์เรย์ที่มีจุดที่แกน y (y-axis)

หากเราต้องการวาดกราฟเส้นจาก $(1, 3)$ ถึง $(8, 10)$ เราต้องส่งสองอาร์เรย์ $[1, 8]$ และ $[3, 10]$ ไปยังฟังก์ชันกราฟ



ตัวอย่างที่ 7.3 ลากเส้นในไดอะแกรมจากตำแหน่ง $(1, 3)$ ไปยังตำแหน่ง $(8, 10)$

```
import matplotlib.pyplot as plt  
import numpy as np  
  
xpoints = np.array([1, 8])  
ypoints = np.array([3, 10])  
  
plt.plot(xpoints, ypoints)  
plt.show()
```



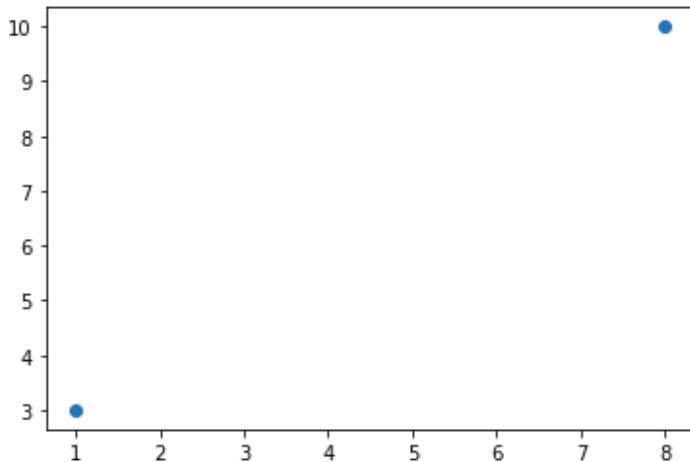
6.2.2 กราฟแบบไม่มีเส้น (Plotting Without Line)

หากต้องการกราฟเฉพาะตัวทำเครื่องหมาย(Marker) สามารถใช้พารามิเตอร์ สัญลักษณ์*_shortcut string notation* 'o' ซึ่งหมายถึง 'rings'



ตัวอย่างที่ 7.4 วัดจุดสองจุดในแผนภาพ จุดหนึ่งอยู่ที่ตำแหน่ง (1, 3) และจุดหนึ่งอยู่ในตำแหน่ง (8, 10):

```
import matplotlib.pyplot as plt  
import numpy as np  
  
xpoints = np.array([1, 8])  
ypoints = np.array([3, 10])  
  
plt.plot(xpoints, ypoints, 'o')  
plt.show()
```





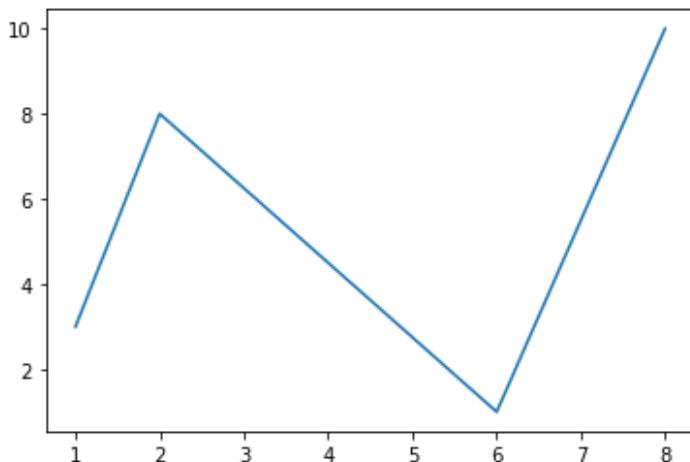
6.2.3 กราฟแบบหลายจุด(Multiple Points)

สามารถสร้างกราฟจุดได้มากเท่าที่ต้องการ เพียงตรวจสอบให้แน่ใจว่ามีจำนวนจุดเท่ากันในทั้งสองแกน



ตัวอย่างที่ 7.5 ลากเส้นโดยอัตโนมัติจากตำแหน่ง (1, 3) ถึง (2, 8) จากนั้นถึง (6, 1) และสุดท้ายไปยังตำแหน่ง (8, 10):

```
import matplotlib.pyplot as plt  
import numpy as np  
  
xpoints = np.array([1, 2, 6, 8])  
ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(xpoints, ypoints)  
plt.show()
```



6.2.4 Default X-Points

หากเราไม่ระบุจุดในแกน x จุดเหล่านี้จะได้รับค่าเริ่มต้น 0, 1, 2, 3 ฯลฯ ขึ้นอยู่กับความยาวของจุด y ดังนั้น หากเราใช้ดังตัวอย่าง 7.5 และไม่พิ้งจุด x ไว้ โค้ดจะมีลักษณะดังตัวอย่าง ดังนั้น หากเราเอาตัวอย่างเดียวกันกับด้านบน และไม่พิ้งจุด x ไว้ โค้ดจะมีลักษณะดังตัวอย่างที่ 7.6 นี้

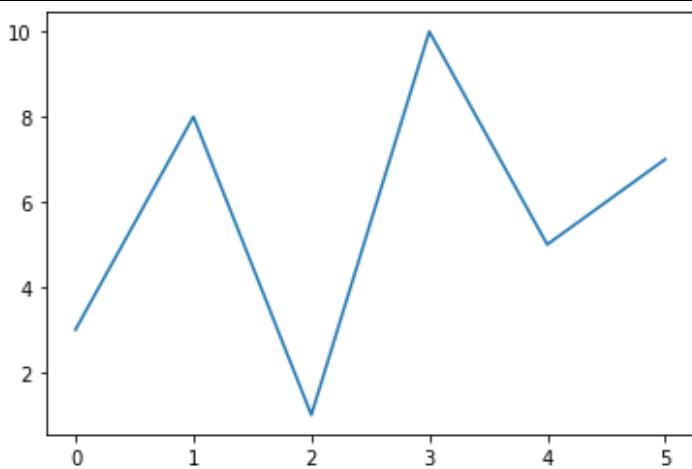


ตัวอย่างที่ 7.6 การกราฟโดยไม่มีจุด x:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10, 5, 7])

plt.plot(ypoints)
plt.show()
```



จุด x ในตัวอย่างด้านบน คือ [0, 1, 2, 3, 4, 5]

6.3 Matplotlib Markers

สามารถใช้อาร์กิวเมนต์คำหลัก marker เพื่อเน้นแต่ละจุดด้วยเครื่องหมายที่ระบุ:

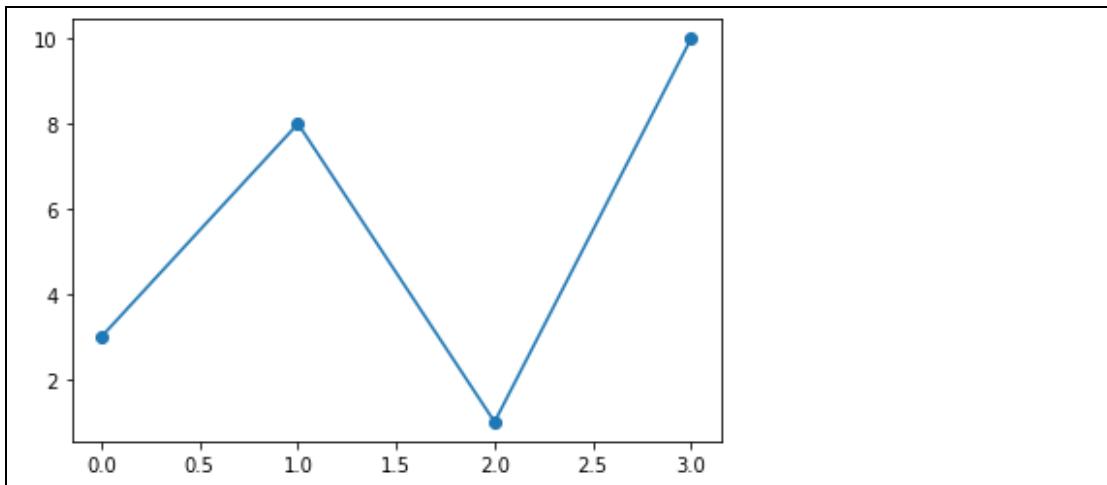


ตัวอย่างที่ 7.8 ทำเครื่องหมายแต่ละจุดด้วยวงกลม:

```
import matplotlib.pyplot as plt
import numpy as np

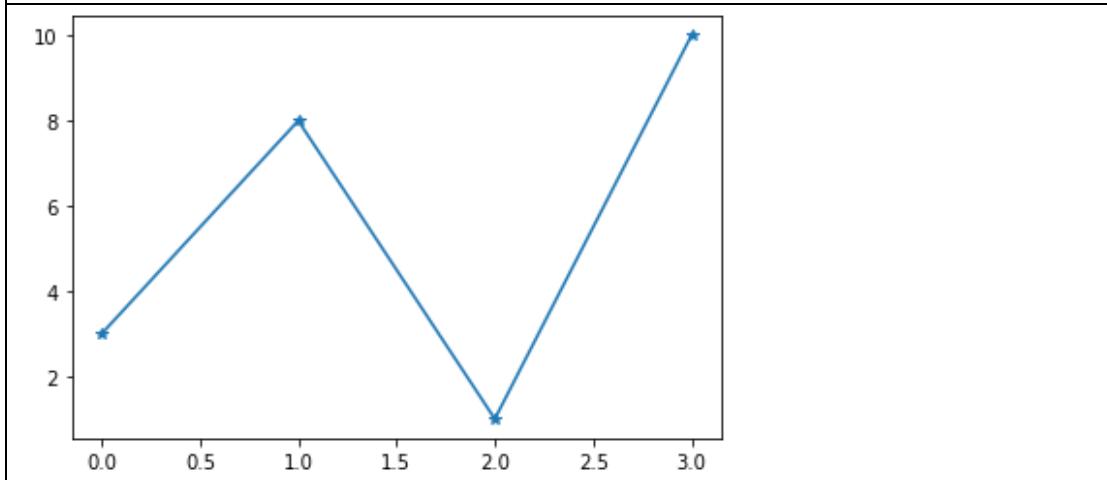
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o')
plt.show()
```



ตัวอย่างที่ 7.7 ทำเครื่องหมายแต่ละจุดด้วยดาว:

```
...  
plt.plot(ypoints, marker = '*')  
...
```



เครื่องหมายอ้างอิง (Marker Reference)

ตารางที่ 20 เครื่องหมายอ้างอิง

Marker	Description	Marker	Description
'o'	Circle	'H'	Hexagon
'*'	Star	'h'	Hexagon
'.'	Point	'v'	Triangle Down
','	Pixel	'^'	Triangle Up
'x'	X	<td>Triangle Left</td>	Triangle Left
'X'	X (filled)	<td>Triangle Right</td>	Triangle Right

Marker	Description	Marker	Description
'+'	Plus	'1'	Tri Down
'P'	Plus (filled)	'2'	Tri Up
's'	Square	'3'	Tri Left
'D'	Diamond	'4'	Tri Right
'd'	Diamond (thin)	' '	Vline
'p'	Pentagon	'_'	Hline

6.3.2 รูปแบบสตริง fmt (Format Strings fmt)

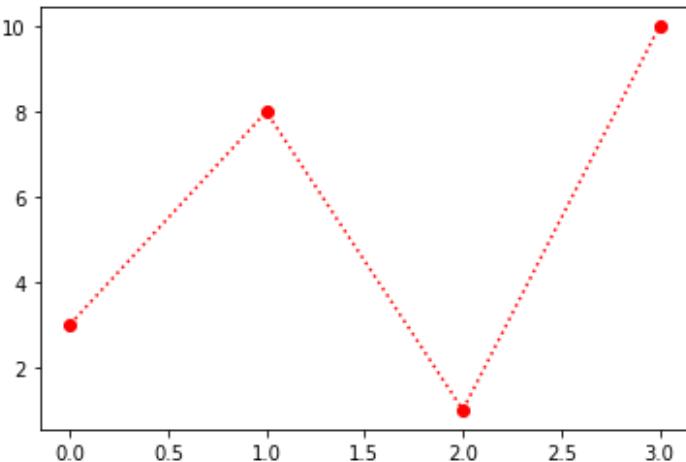
สามารถใช้พารามิเตอร์ **shortcut string notation** เพื่อบรรบตัวทำเครื่องหมายพารามิเตอร์นี้เรียกอีกอย่างว่า fmt และเขียนด้วยไวยากรณ์นี้:

marker|line|color



ตัวอย่างที่ 7.9 ทำเครื่องหมายแต่ละจุดด้วยวงกลม:

```
import matplotlib.pyplot as plt
import numpy as np
y whole = np.array([3, 8, 1, 10])
plt.plot(y whole, 'o:r')
plt.show()
```





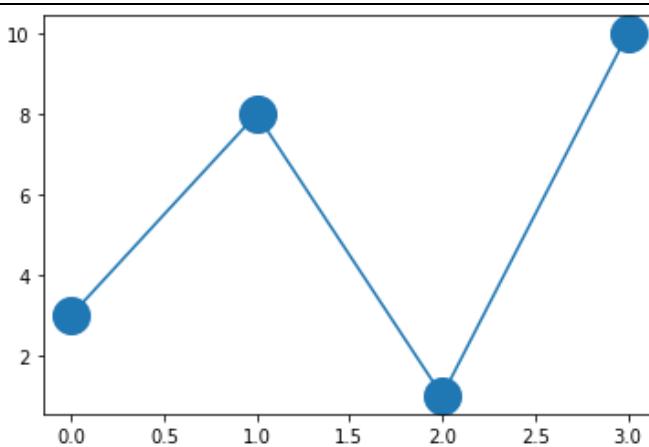
6.3.1 ขนาดมาร์กเกอร์ (Marker Size)

สามารถใช้อาร์กิวเมนต์คำลักษณ์ `markersize` หรือเวอร์ชันที่สั้นกว่า `ms` เพื่อกำหนดขนาดของเครื่องหมาย:



ตัวอย่างที่ 7.10 กำหนดขนาดของเครื่องหมายเป็น 20:

```
import matplotlib.pyplot as plt  
import numpy as np  
y whole = np.array([3, 8, 1, 10])  
plt.plot(y whole, marker = 'o', ms = 20)  
plt.show()
```



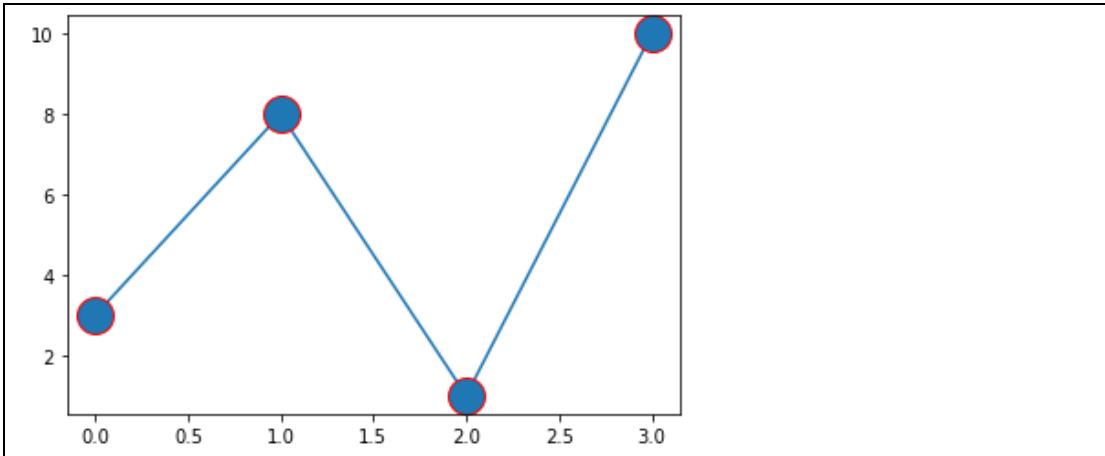
6.3.3 สีมาร์กเกอร์ (Marker Color)

สามารถใช้อาร์กิวเมนต์คำลักษณ์ `markeredgecolor` หรือค่าที่สั้นกว่า `mec` เพื่อกำหนดสีของขอบของเครื่องหมาย:



ตัวอย่างที่ 7.11 ตั้งค่าสี EDGE เป็นสีแดง:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
y whole = np.array([3, 8, 1, 10])  
  
plt.plot(y whole, marker = 'o', ms = 20, mec = 'r')  
plt.show()
```



ใช้ทั้งอาร์กิวเมนต์ `mec` และ `mfc` เพื่อกำหนดสีของマークเกอร์ทั้งหมด:

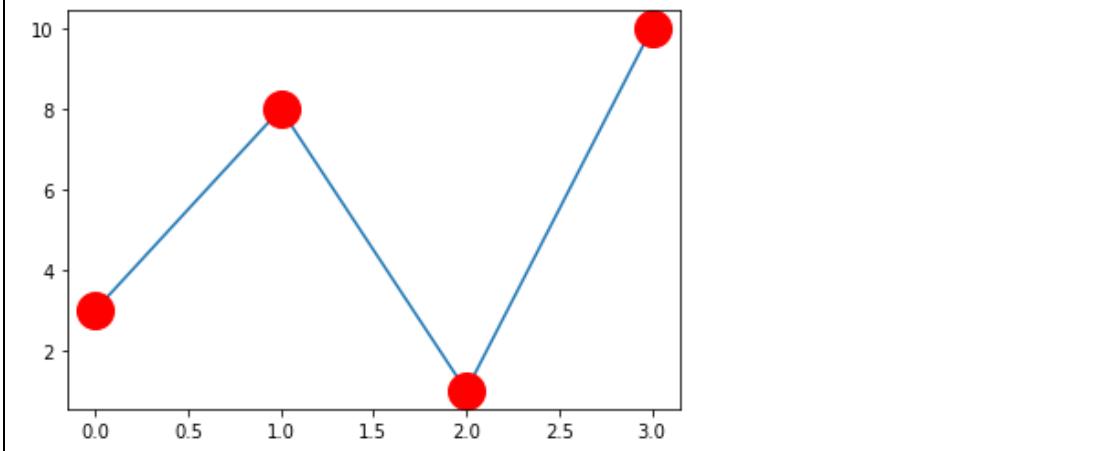


ตัวอย่างที่ 7.12 กำหนดสีของทั้งขอบและหน้าเป็นสีแดง:

```
import matplotlib.pyplot as plt
import numpy as np

y whole = np.array([3, 8, 1, 10])

plt.plot(y whole, marker = 'o', ms = 20, mec = 'r', mfc = 'r')
plt.show()
```



สามารถใช้ค่าสีฐานลับแทน :

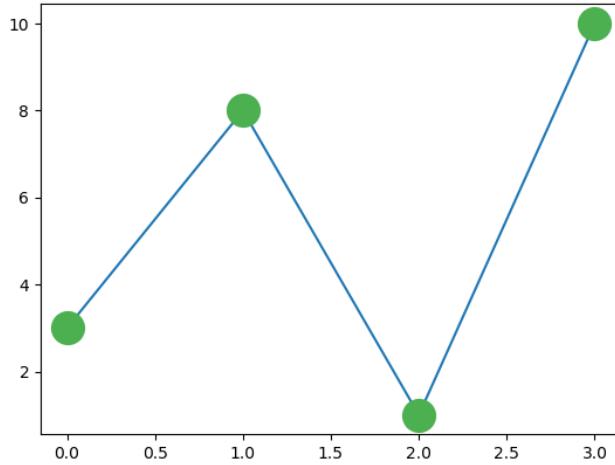


ตัวอย่างที่ 7.13 ทำเครื่องหมายแต่ละจุดด้วยสีเขียวที่สวยงาม:

...

```
plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc = '#4CAF50')
```

...



Colors HEX

Hex Calculator

#ff0000

rgb(255, 0, 0)
hsl(0, 100%, 50%)

R: ff

G: 00

B: 00

[Use this color in our Color Picker](#)

■ เลขฐานสิบหกสี

ค่าสีฐานสิบหกยังได้รับการสนับสนุนในเบราว์เซอร์ทั้งหมด

■ ระบบสีฐานสิบหกด้วย: #RRGGBB

RR (สีแดง), GG (สีเขียว) และ BB (สีน้ำเงิน) เป็นจำนวนเต็มฐานสิบหกระหว่าง 00 ถึง FF ซึ่งระบุความเข้มของสี

ตัวอย่างเช่น #0000FF จะแสดงเป็นสีน้ำเงิน เนื่องจากองค์ประกอบสีน้ำเงินถูกตั้งค่าเป็นค่าสูงสุด (FF) และองค์ประกอบอื่นๆ ถูกตั้งค่าเป็น 00

6.4 กราฟเส้น (Matplotlib Line)

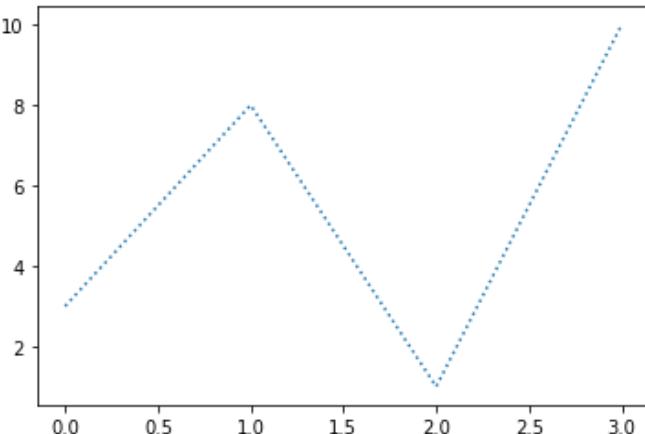
6.4.1 รูปแบบเส้น (Linestyle)

สามารถใช้คีย์เวิร์ดอาร์กิวเมนต์ linestyle หรือ shorter ໄ้เพื่อเปลี่ยนรูปแบบของเส้นที่ลงจุด:



ตัวอย่างที่ 7.14 ใช้เส้นปะ

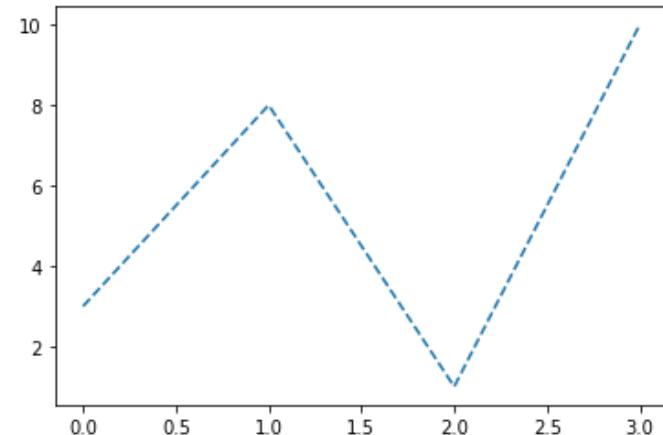
```
import matplotlib.pyplot as plt  
import numpy as np  
  
y whole = np.array([3, 8, 1, 10])  
  
plt.plot(y whole, linestyle = 'dotted')  
plt.show()
```





ตัวอย่างที่ 7.15 ใช้เส้นปะ

```
plt.plot(ypoints, linestyle = 'dashed')
```



ไวยากรณ์ที่สั้นลง ลักษณะเส้นสามารถเขียนในรูปแบบที่สั้นกว่าได้:

linestyle เขียนได้เป็น ls.

dotted เขียนได้เป็น :.

dashed เขียนได้เป็น --.

ลักษณะเส้นสามารถเลือกรูปแบบได้ดังนี้:

ตารางที่ 21 รูปแบบลักษณะเส้น

Style	Or
'solid' (default)	'-'
'dotted'	':'
'dashed'	'--'
'dashdot'	'-.'
'None'	" or ''



สีเส้น สามารถใช้อาร์กิวเมนต์คำลักษณ์ color หรือค่าที่สั้นกว่า c เพื่อกำหนดสีของเส้น:

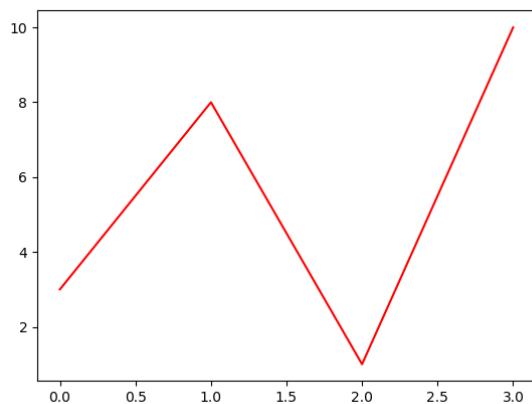


ตัวอย่างที่ 7.16 กำหนดสีเส้นเป็นสีแดง:

```
import matplotlib.pyplot as plt
import numpy as np

y whole points = np.array([3, 8, 1, 10])

plt.plot(y whole points, color = 'r')
plt.show()
```

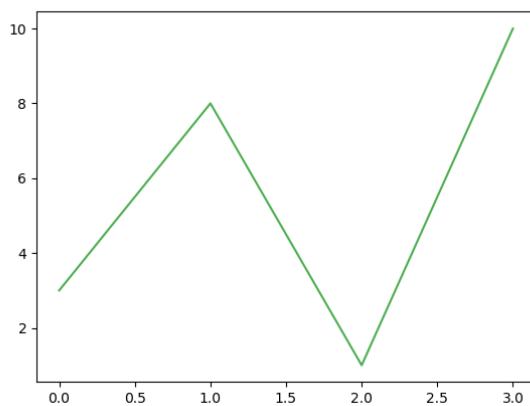


สามารถใช้ค่าสีฐานสิบหก :



ตัวอย่างที่ 7.17 กราฟด้วยเส้นสีเขียวที่สวยงาม:

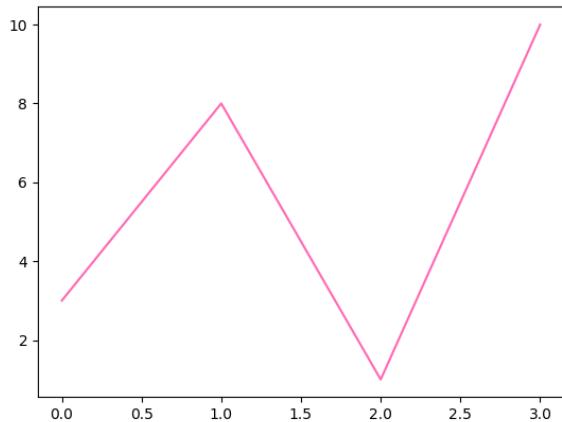
```
...
plt.plot(y whole points, c = '#4CAF50')
...
```





ตัวอย่างที่ 7.18 กราฟด้วยสีชื่อ "hotpink":

```
...  
plt.plot(ypoints, c = 'hotpink')  
...
```



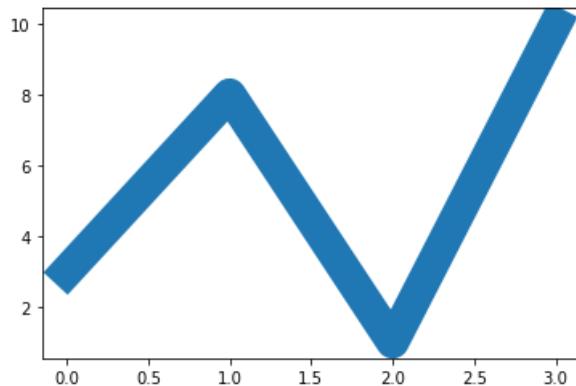
6.4.2 กำหนดความกว้างของเส้น

สามารถใช้อาร์กิวเมนต์คีย์เวิร์ด linewidth หรือค่าที่สั้นกว่า lw เพื่อเปลี่ยนความกว้างของบรรทัด ค่าเป็น floating number , in points:



ตัวอย่างที่ 7.19 กราฟที่มีเส้นกว้าง 20.5pt:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, linewidth = '20.5')  
plt.show()
```



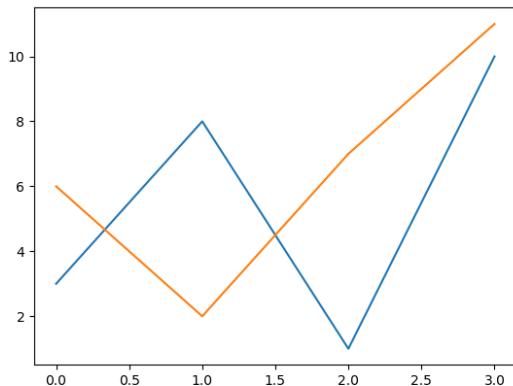
6.4.3 สร้างกราฟแบบหลายบรรทัด

สามารถสร้างกราฟเส้นได้มากเท่าที่ต้องการโดยเพียงแค่เพิ่มฟังก์ชัน plt.plot()



ตัวอย่างที่ 7.20 วัดสองบรรทัดโดยระบบ plt.plot() ฟังก์ชันสำหรับแต่ละบรรทัด:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
y1 = np.array([3, 8, 1, 10])  
y2 = np.array([6, 2, 7, 11])  
  
plt.plot(y1)  
plt.plot(y2)  
  
plt.show()
```



สามารถกราฟหลายบรรทัดได้ด้วยการเพิ่มจุดสำหรับแกน x และ y สำหรับแต่ละบรรทัดใน plt.plot() ฟังก์ชันเดียวกัน (ในตัวอย่างข้างต้น เราระบุเฉพาะจุดบนแกน y เท่านั้น หมายความว่าจุดบนแกน x มีค่าเริ่มต้น (0, 1, 2, 3)) ค่า x และ y- เป็นคู่:

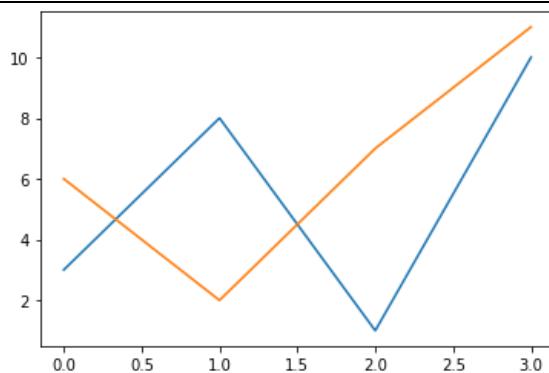


ตัวอย่างที่ 7.21 วาดเส้นสองเส้นโดยระบุค่าจุด x และจุด y สำหรับทั้งสองเส้น:

```
import matplotlib.pyplot as plt
import numpy as np

x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])
x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])

plt.plot(x1, y1, x2, y2)
plt.show()
```



6.5 Matplotlib Labels and Title

6.5.1 การสร้างป้ายกำกับสำหรับกราฟ (Create Labels for a Plot)

ไฟลอนสามารถใช้ฟังก์ชัน xlabel() และ ylabel() เพื่อตั้งค่าป้ายกำกับสำหรับแกน x และ แกน y



ตัวอย่างที่ 7.22 เพิ่มป้ายกำกับให้กับแกน x และ y

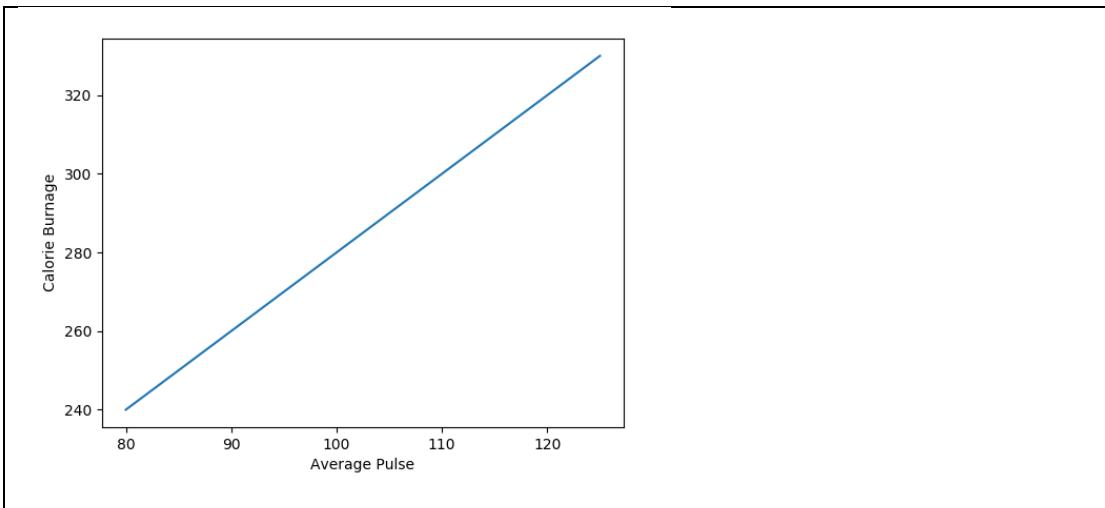
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```



6.5.2 สร้างชื่อเรื่องสำหรับกราฟ (Create a Title for a Plot)



ตัวอย่างที่ 7.23 สร้างชื่อเรื่องสำหรับกราฟ

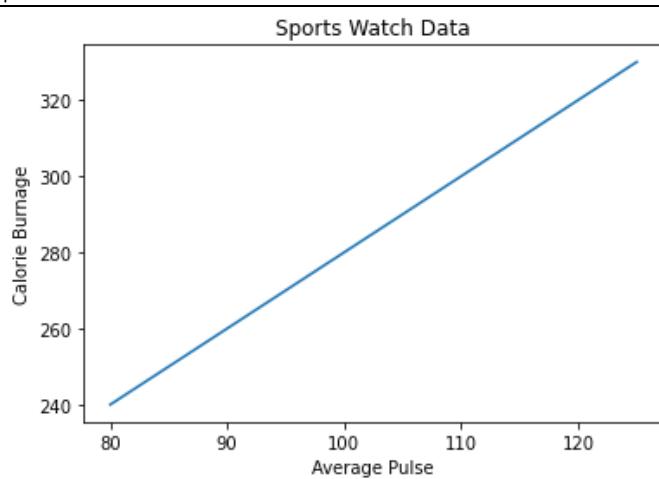
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```





6.5.3 ตั้งค่าคุณสมบัติแบบอักษรสำหรับชื่อเรื่องและป้ายกำกับ

สามารถใช้ fontdict พารามิเตอร์ใน xlabel(), ylabel() และ title() เพื่อตั้งค่าคุณสมบัติฟอนต์สำหรับชื่อเรื่องและป้ายกำกับ



ตัวอย่างที่ 7.24 ตั้งค่าคุณสมบัติแบบอักษรสำหรับชื่อเรื่องและป้ายกำกับ

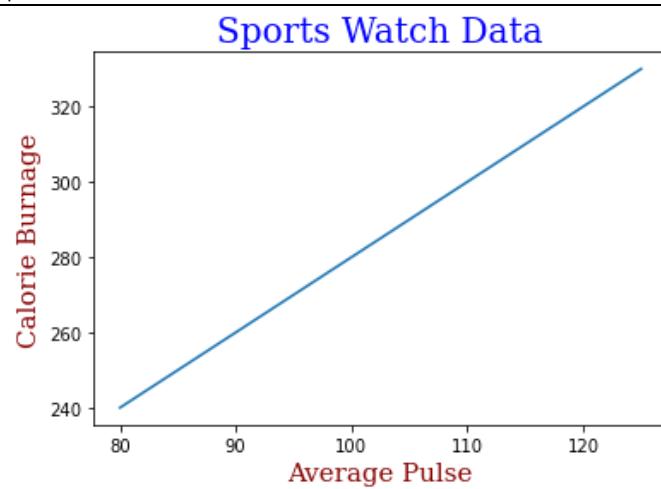
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

plt.plot(x, y)
plt.show()
```



6.5.4 ตำแหน่งชื่อเรื่อง

สามารถใช้พารามิเตอร์ loc ใน title() เพื่อจัดตำแหน่งหัวเรื่อง Legal values คือ 'left', 'right', and 'center' Default value คือ 'center'.



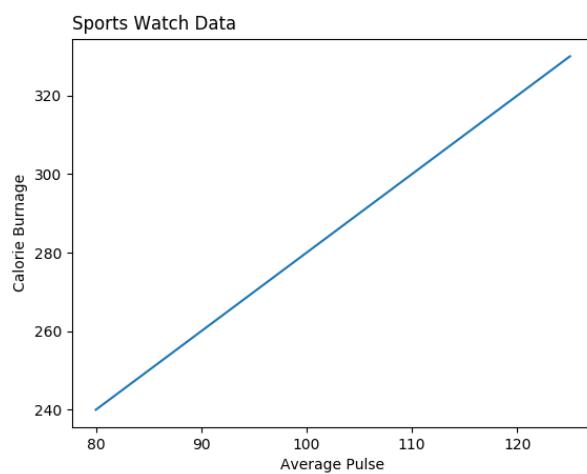
ตัวอย่างที่ 7.25 วางแผนชื่อเรื่องไปทางซ้าย

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data", loc = 'left')
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)
plt.show()
```



6.6 เส้นกริดของกราฟ (Matplotlib Grid)

6.6.1 เพิ่มเส้นกริด (Add Grid Lines to a Plot)

ด้วย Pyplot สามารถใช้ grid() พงก์ชันนี้ เพื่อเพิ่มเส้นกริดให้กับกราฟ

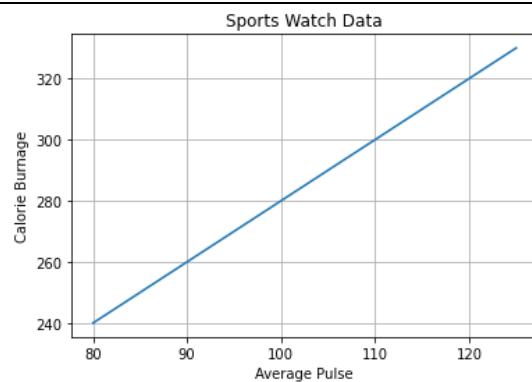


ตัวอย่างที่ 7.26 เพิ่มเส้นกริดให้กับกราฟ:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.plot(x, y)
plt.grid()
plt.show()
```



6.6.2 ระบุเส้นกริดที่จะแสดง

สามารถใช้ axis พารามิเตอร์ในฟังก์ชัน grid() เพื่อระบุเส้นกริดที่จะแสดง
Legal values คือ: 'x', 'y' และ 'ทั้ง' ค่าเริ่มต้นคือ 'ทั้งสอง'



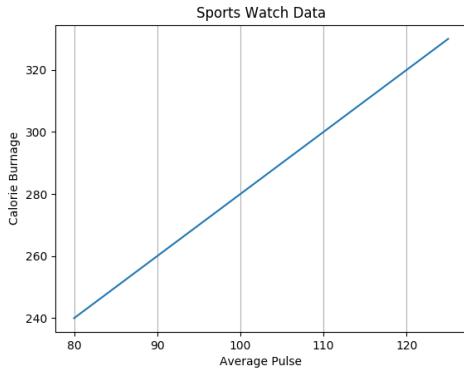
ตัวอย่างที่ 7.27 แสดงเส้นกริดสำหรับแกน x เท่านั้น

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.plot(x, y)
plt.grid('x')
plt.show()
```

```
plt.grid(axis = 'x')
plt.show()
```

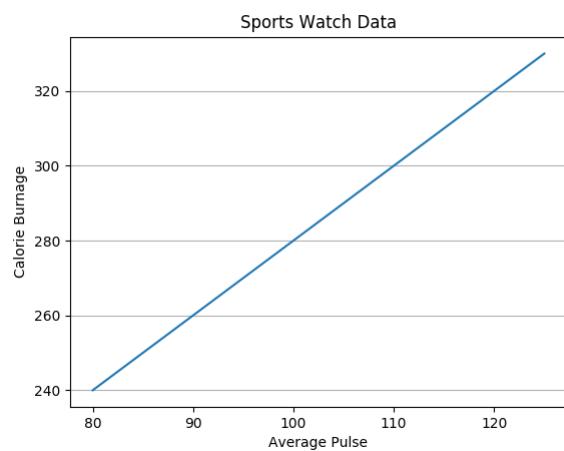


ตัวอย่างที่ 7.28 แสดงเส้นกริดสำหรับแกน y เท่านั้น

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.plot(x, y)
plt.grid(axis = 'y')
plt.show()
```





6.6.3 ตั้งค่าคุณสมบัติของเส้นสำหรับ Grid

สามารถตั้งค่าคุณสมบัติเส้นของเส้นตารางได้ดังนี้:

```
grid (color = ' color ', linestyle = ' linestyle ', linewidth = number )
```

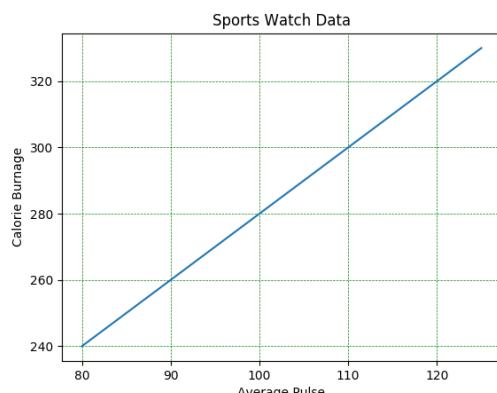


ตัวอย่างที่ 7.29 ตั้งค่าคุณสมบัติของเส้นสำหรับ Grid

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.plot(x, y)
plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)
plt.show()
```



6.7 กราฟย่อย (Matplotlib Subplots)

6.7.1 แสดงหลายกราฟ (Display Multiple Plots)

ด้วยฟังก์ชัน `subplots()` สามารถวาดหลายกราฟในรูปเดียว:



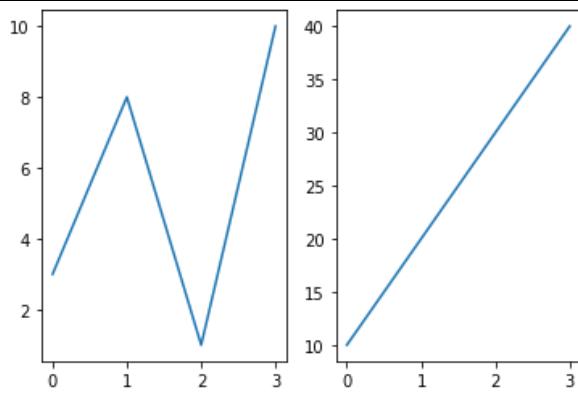
ตัวอย่างที่ 7.30 วาด 2 กราฟ

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
```

```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.show()
```



6.7.2 พังก์ชันของกราฟย่อย (The subplots() Function)

พังก์ชัน subplots() อธิบายถึง layout ของ figure เลย์เอาต์ถูกรักษาไว้เป็น常 แล้วคอลัมน์ ซึ่งแสดงโดยอาร์กิวเมนต์แรก และตัวที่สอง อาร์กิวเมนต์ที่สามแสดงถึงดัชนีของกราฟ ปัจจุบัน

```
plt.subplot(1, 2, 1)
#the figure has 1 row, 2 columns, and this plot is the first plot.
```

```
plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, and this plot is the second plot.
```



ตัวอย่างที่ 7.31 วาด 2 Plots

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
```



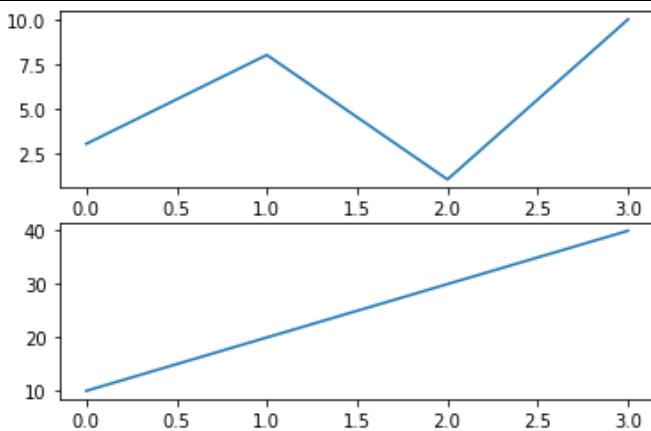
```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 1, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 1, 2)
plt.plot(x,y)

plt.show()
```



ตัวอย่างที่ 7.32 จำนวน 6 Plots

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
```

```

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

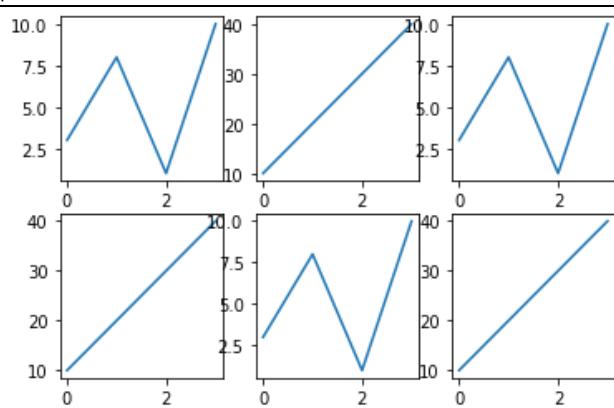
plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()

```





6.7.3 หัวเรื่อง (Title)

title() function:



ตัวอย่างที่ 7.33 หัวเรื่อง (Title)

```
import matplotlib.pyplot as plt
import numpy as np

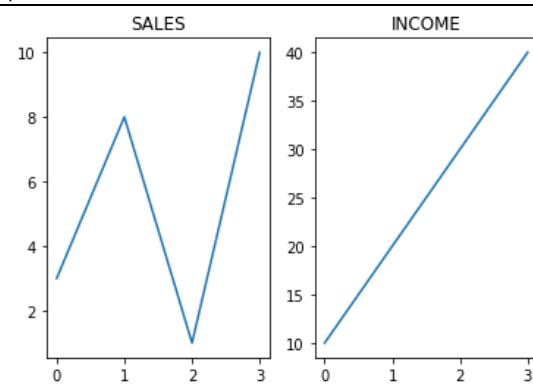
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.show()
```



6.7.4 ชื่อหัวเรื่อง (Super Title)

สามารถเพิ่มชื่อให้กับภาพทั้งหมดด้วยฟังก์ชัน `suptitle()`



ตัวอย่างที่ 7.34 เพิ่มชื่อเรื่องให้กับ figure

```
import matplotlib.pyplot as plt  
import numpy as np
```

```
#plot 1:
```

```
x = np.array([0, 1, 2, 3])  
y = np.array([3, 8, 1, 10])
```

```
plt.subplot(1, 2, 1)
```

```
plt.plot(x,y)
```

```
plt.title("SALES")
```

```
#plot 2:
```

```
x = np.array([0, 1, 2, 3])
```

```
y = np.array([10, 20, 30, 40])
```

```
plt.subplot(1, 2, 2)
```

```
plt.plot(x,y)
```

```
plt.title("INCOME")
```

```
plt.show()
```





6.8 กราฟแบบกระจาย (Matplotlib Scatter)

6.8.1 การสร้างกราฟกระจาย (Creating Scatter Plots)

ด้วย Pyplot สามารถใช้ฟังก์ชัน scatter() นี้ เพื่อวาดกราฟแบบกระจาย scatter() แปลงฟังก์ชันหนึ่งจุดสำหรับแต่ละสังเกต มันต้องการสองอาร์เรย์ที่มีความยาวเท่ากัน หนึ่งอาร์เรย์สำหรับค่าของแกน x และอีกชุดสำหรับค่าบนแกน y:

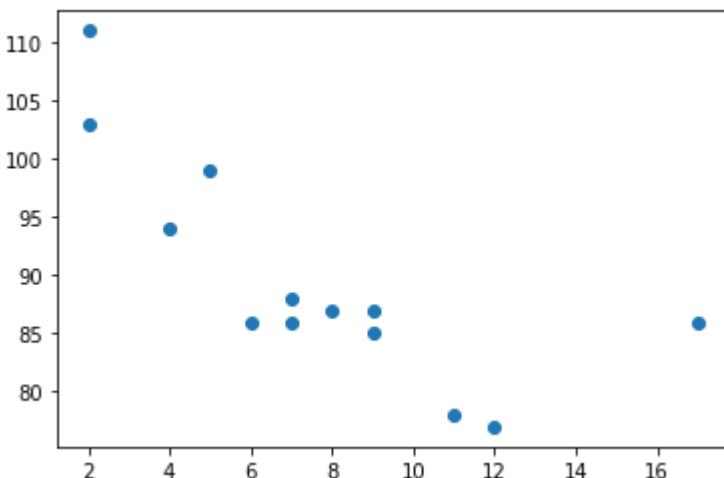


ตัวอย่างที่ 7.35 กราฟกระจายอย่างง่าย:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```



การสังเกตในตัวอย่างข้างต้นเป็นผลจากการถ่าย 13 คันที่ผ่านไปมา แกน X แสดงอายุรถ แกน Y แสดงความเร็วของรถเมื่อผ่าน มีความสัมพันธ์ใด ๆ ระหว่างการสังเกตหรือไม่ ดูเหมือนว่ายิ่งรถใหม่เท่าไหร่ ก็ยิ่งขับเร็วขึ้น แต่นั่นอาจเป็นเรื่องบังเอิญ เพราะเราจดทะเบียนรถเพียง 13 คันเท่านั้น

6.8.2 เปรียบเทียบกราฟ (Compare Plots)

ในตัวอย่างข้างต้น ดูเหมือนว่าจะมีความสัมพันธ์ระหว่างความเร็วและอายุ แต่ถ้าเรากราฟข้อสังเกตจากวันอื่นด้วยล่ะ กราฟกระจายจะบอกอะไรเราอีกไหม



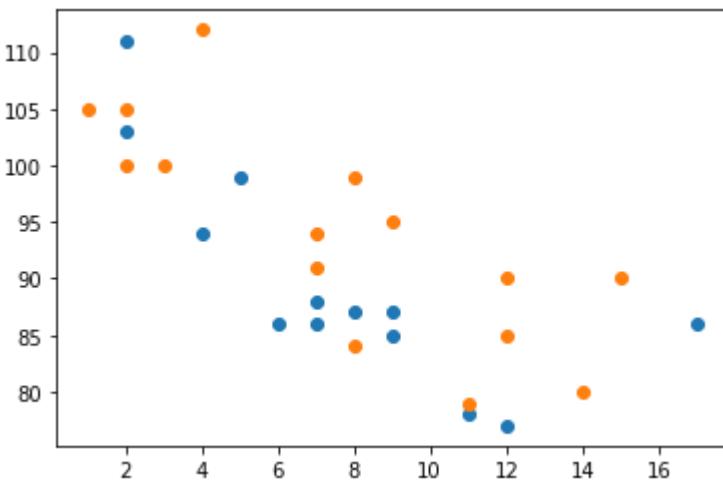
ตัวอย่างที่ 7.36 วัด 2 พร็อตในภาพเดียวกัน

```
import matplotlib.pyplot as plt
import numpy as np

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()
```



6.8.4 สี(Colors)

สามารถกำหนดสีของสำหรับแต่ละกราฟกระจายด้วย อาร์กิวเมนต์ color หรือ c :



ตัวอย่างที่ 7.38 กำหนดสี

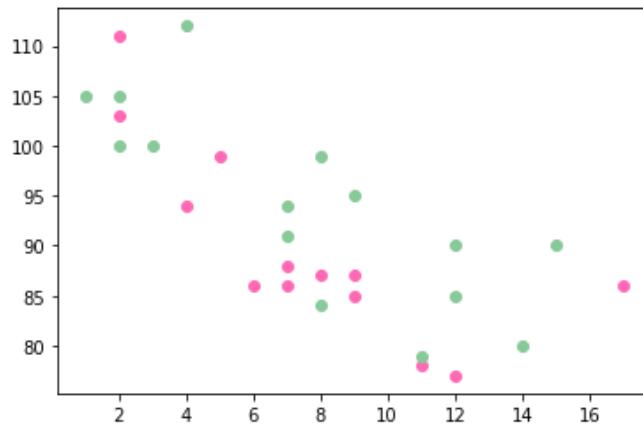
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
```



```
plt.scatter(x, y, color = 'hotpink')

x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color = '#88c999')
plt.show()
```



6.8.3 สีแต่ละจุด (Color Each Dot)

สามารถกำหนดสีเฉพาะสำหรับแต่ละจุดได้โดยใช้อาร์เรย์ของสีเป็นค่าสำหรับ c อาร์กิวเมนต์

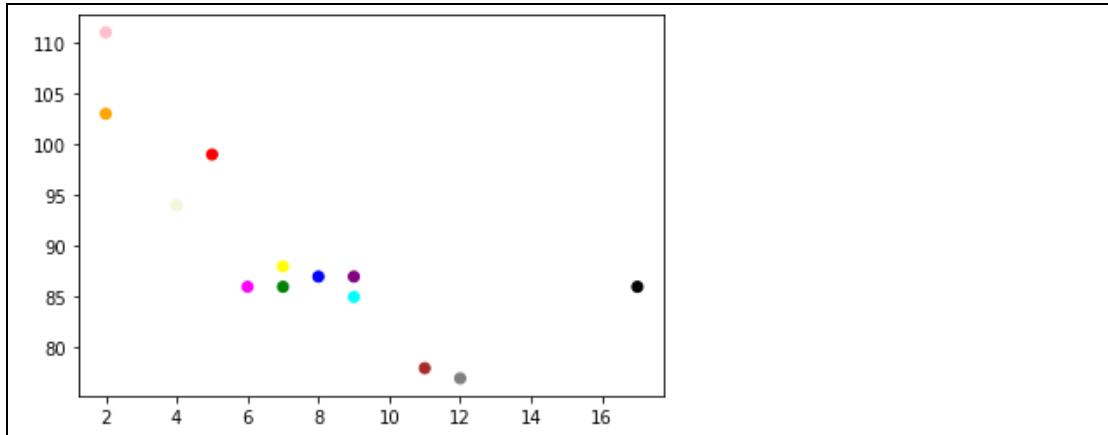


ตัวอย่างที่ 7.37 กำหนดสีของเครื่องหมาย象:

```
import matplotlib.pyplot as plt
import numpy as np

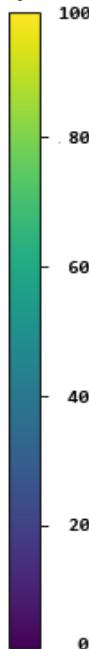
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors =
np.array(["red","green","blue","yellow","pink","black","orange","purple","beige","brown",
"gray","cyan","magenta"])

plt.scatter(x, y, c=colors)
plt.show()
```



6.8.5 แผนที่สี (ColorMap)

โมดูล Matplotlib มีแผนที่สีจำนวนมาก แผนผังสีเป็นเหมือนรายการสี โดยแต่ละสีมีค่าที่อยู่ในช่วงตั้งแต่ 0 ถึง 100 นี้คือตัวอย่างของ colormap:



แผนที่สีนี้เรียกว่า 'viridis' และจะเห็นว่ามีตั้งแต่ 0 ซึ่งเป็นสีม่วง และสูงสุด 100 ซึ่งเป็นสีเหลือง

6.8.6 วิธีใช้แผนที่สี (ColorMap)

สามารถระบุ colormap ด้วยอาร์กิวเม้นต์ของคีย์เวิร์ด cmap ด้วยค่าของ colormap ในกรณี 'viridis' ซึ่งเป็นหนึ่งใน colormap ที่มีอยู่แล้วใน Matplotlib

นอกจากนี้ ต้องสร้างอาร์เรย์ที่มีค่า (ตั้งแต่ 0 ถึง 100) หนึ่งค่าสำหรับแต่ละจุดในแผนภาพ กระจาย:



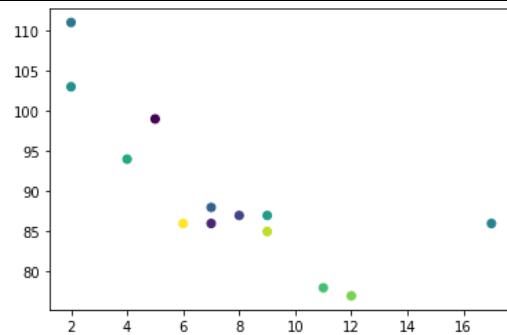
ตัวอย่างที่ 7.38 สร้างอาร์เรย์สี และระบุแผนผังสีในกราฟแบบกระจาย

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')

plt.show()
```



สามารถรวม colormap ไว้ในภาพวาดได้โดยการใส่คำสั่ง plt.colorbar():

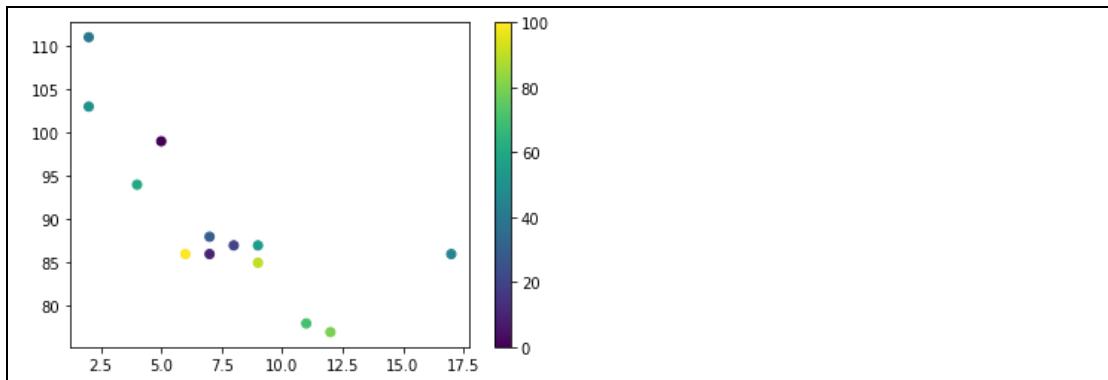


ตัวอย่างที่ 7.39 รวมแผนผังสีจริง:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')
plt.colorbar()
plt.show()
```



6.8.7 ขนาด (Size)

สามารถเปลี่ยนขนาดของจุดด้วย `s` อาร์กิวเม้นต์ เช่นเดียวกับสี ตรวจสอบให้แน่ใจว่า อาร์เรย์สำหรับขนาดมีความยาวเท่ากับอาร์เรย์สำหรับแกน `x` และ `y`:



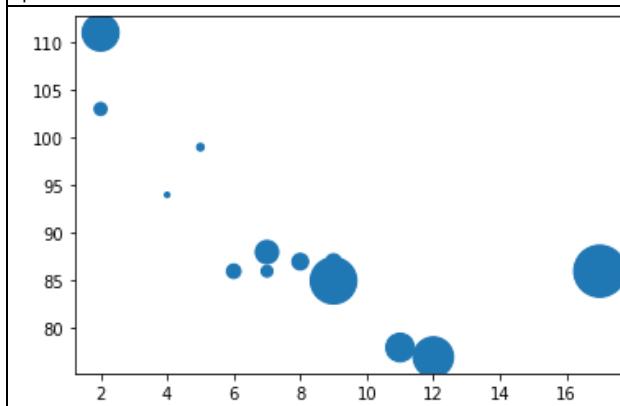
ตัวอย่างที่ 7.40 กำหนดขนาดของ Marker

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes)

plt.show()
```



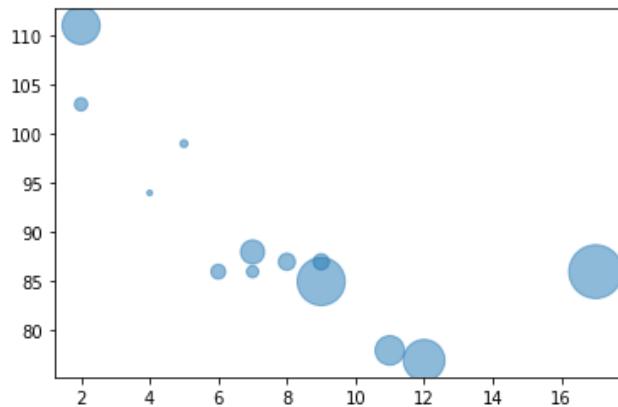
6.8.9 ปรับความโปร่งใสของจุดด้วย Alpha

สามารถปรับความโปร่งใสของจุดด้วย `alpha` อาร์กิวเม้นต์ เช่นเดียวกับสี ตรวจสอบให้แน่ใจว่า อาร์เรย์สำหรับขนาดมีความยาวเท่ากับอาร์เรย์สำหรับแกน `x` และ `y`:



ตัวอย่างที่ 7.41 กำหนดขนาดของ Marker

```
import matplotlib.pyplot as plt  
import numpy as np  
  
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])  
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])  
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])  
  
plt.scatter(x, y, s=sizes, alpha=0.5)  
  
plt.show()
```



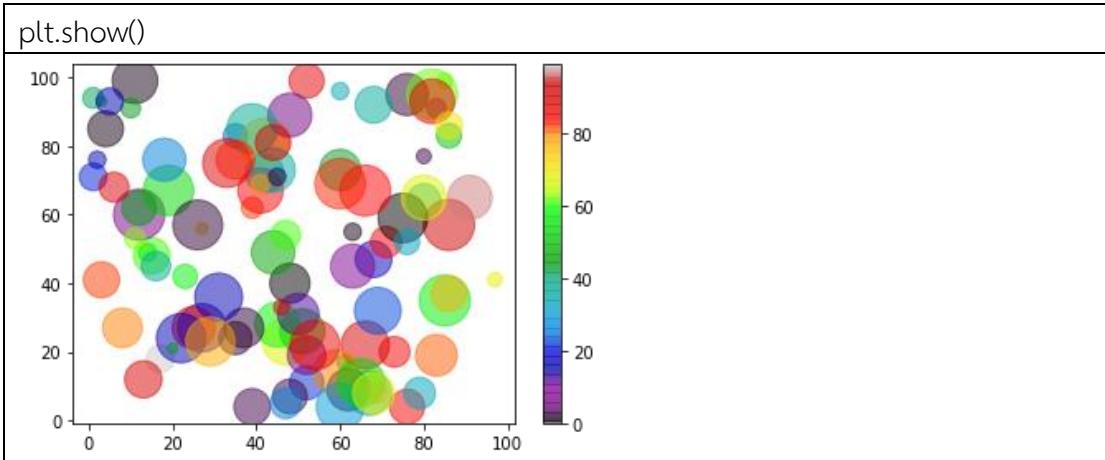
6.8.8 รวมขนาดสี และปรับความโปร่งแสง (Combine Color Size and Alpha)

สามารถรวม colormap กับขนาดต่างๆ บนจุดต่างๆ ได้ วิธีนี้จะทำให้เห็นภาพได้ดีที่สุดหากจุดนั้นโปร่งใส:



ตัวอย่างที่ 7.42 สร้างอาร์เรย์สุ่มที่มีค่า 100 ค่าสำหรับจุด x จุด y สี และขนาด:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
x = np.random.randint(100, size=(100))  
y = np.random.randint(100, size=(100))  
colors = np.random.randint(100, size=(100))  
sizes = 10 * np.random.randint(100, size=(100))  
  
plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')  
plt.colorbar()
```



6.9 กราฟแท่ง (Matplotlib Bars)

6.9.1 การสร้างกราฟแท่ง (Creating Bars)

ด้วย Pyplot สามารถใช้ `bar()` ฟังก์ชันนี้เพื่อวาดกราฟแท่ง:

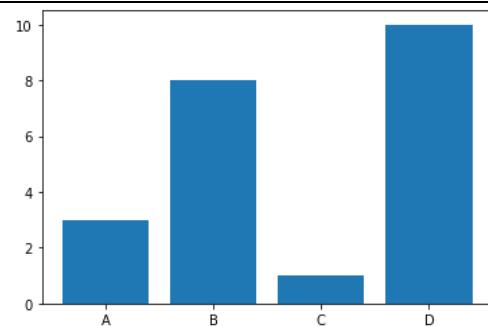


ตัวอย่างที่ 7.43 วาด 4 แท่ง

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```



ฟังก์ชัน `bar()` ใช้อาร์กิวเม้นต์ที่อธิบายถึงรูปแบบของแท่ง ประเภทและค่า และแสดงโดยครั้งแรก และครั้งที่สองอาร์กิวเม้นต์อาร์เรย์

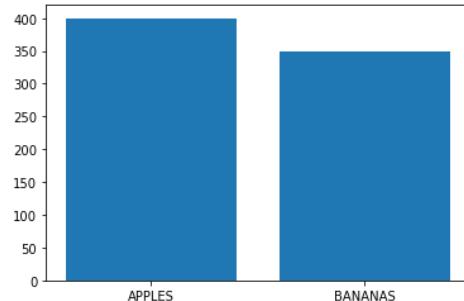


ตัวอย่างที่ 7.44 แสดงกราฟโดยใช้ฟังก์ชันbar()

```
x = ["APPLES", "BANANAS"]
```

```
y = [400, 350]
```

```
plt.bar(x, y)
```



6.9.2 แบบแนวโน้ม (Horizontal Bars)

หากต้องการให้แบบแสดงแนวโน้มแทนที่จะเป็นแนวตั้ง ให้ใช้ barh() ฟังก์ชัน:



ตัวอย่างที่ 7.44 แสดงแนวโน้มโดยใช้ barh() ฟังก์ชัน

```
import matplotlib.pyplot as plt
```

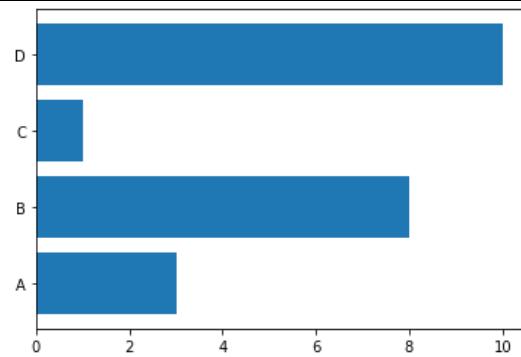
```
import numpy as np
```

```
x = np.array(["A", "B", "C", "D"])
```

```
y = np.array([3, 8, 1, 10])
```

```
plt.barh(x, y)
```

```
plt.show()
```



6.9.3 สีบาร์ (Bar Color)

`bar()` และ `barh()` ใช้ keyword argument `color` ในการตั้งค่าสีของบาร์นี้:

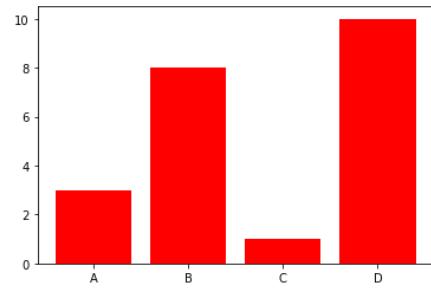


ตัวอย่างที่ 7.45 กำหนดสีบาร์

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "red")
plt.show()
```



6.9.4 ชื่อสี (Color Names)

140 supported color names.

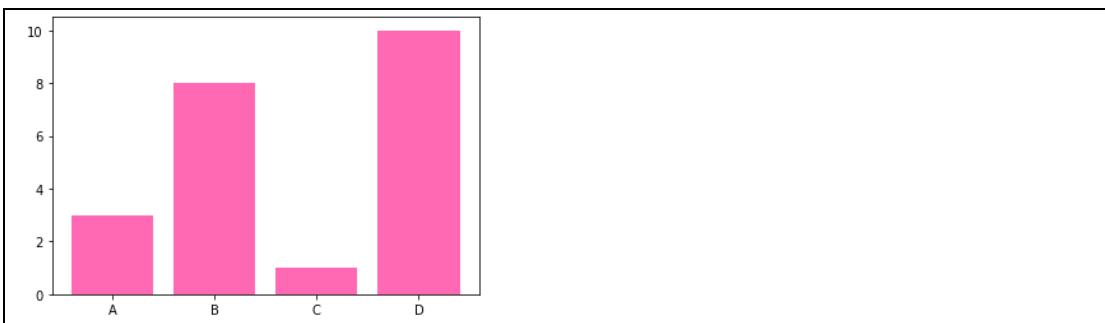


ตัวอย่างที่ 7.46 วาด 4 "hot pink" bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "hotpink")
plt.show()
```



Hexadecimal color values:

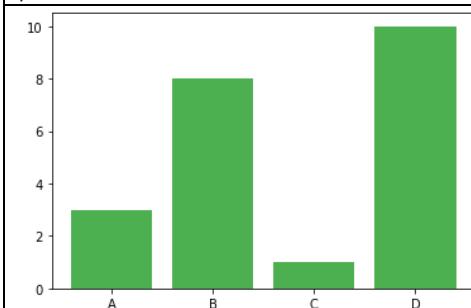


ตัวอย่างที่ 7.47 กำหนดสี

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "#4CAF50")
plt.show()
```



6.9.5 ความกว้างของบาร์ (Bar Width)

bar() ใช้ keyword argument ใช้ width กำหนดความกว้างของบาร์นี้:



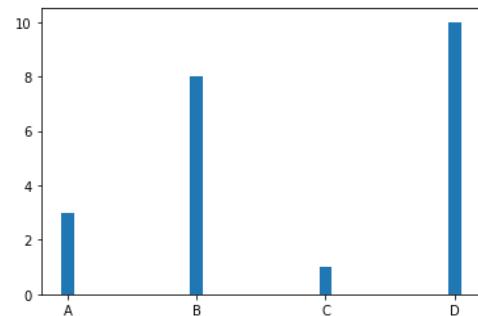
ตัวอย่างที่ 7.48 กำหนดความกว้างของบาร์

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, width = 0.1)
```

`plt.show()`



6.9.6 ความสูงของกราฟแท่ง (Bar Height)

`barh()` keyword argument ใช้ `height` ในการตั้งค่าความสูงของบาร์นี้:

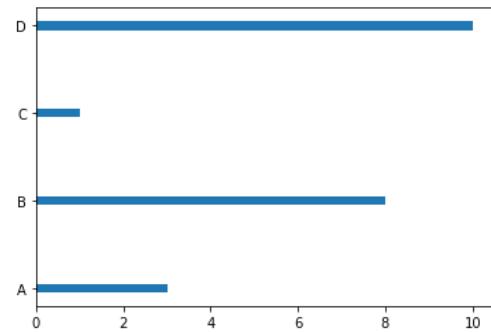


ตัวอย่างที่ 7.49 การตั้งค่าความสูงของบาร์

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y, height = 0.1)
plt.show()
```

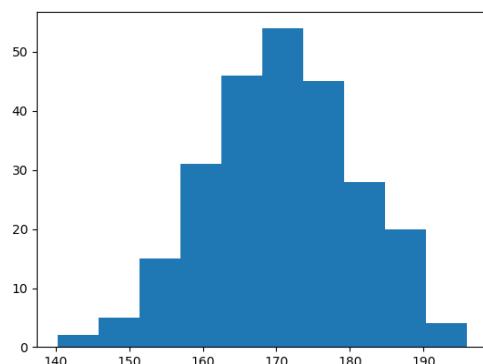




6.10 กราฟฮิสโตแกรม (Matplotlib Histograms)

ฮิสโตแกรม (Histogram) คือ กราฟที่แสดงการแจกแจงความถี่ เป็นกราฟแสดงจำนวนการสังเกตภายในแต่ละช่วงที่กำหนด

ตัวอย่าง: สมมติว่าขอส่วนสูง 250 คน อาจได้ฮิสโตแกรมดังนี้:



สามารถอ่านได้จากฮิสโตแกรมที่มีประมาณ:

- 2 คน จาก 140 ถึง 145 ซม.
- 5 คน จาก 145 ถึง 150 ซม.
- 15 คน จาก 150 ถึง 155 ซม.
- 31 คน จาก 155 ถึง 160 ซม.
- 46 คน จาก 160 ถึง 165 ซม.
- 53 คน จาก 165 ถึง 170 ซม.
- 45 คน จาก 170 ถึง 175 ซม.
- 28 คน จาก 175 ถึง 180 ซม.
- 21 คน จาก 180 ถึง 185 ซม.
- 4 คน จาก 185 ถึง 190 ซม.

ใน Matplotlib เราใช้ `hist()` ฟังก์ชันนี้เพื่อสร้างฮิสโตแกรม `hist()` ฟังก์ชันจะใช้อาร์เรย์ของตัวเลขที่จะสร้าง histogram ที่อาร์เรย์จะถูกส่งเข้าไปในฟังก์ชันเป็นอาร์กิวเมนต์ สำหรับความเรียบง่ายที่เราใช้ NumPy เพื่อสุ่มสร้างอาร์เรย์ 250 ค่าที่ค่าที่จะมีสมาชิรอบ 170 และค่าเบี่ยงเบนมาตรฐานคือ 10 เรียนรู้เพิ่มเติมเกี่ยวกับการเพริ่งกระจายข้อมูลปกติของเรามาเครื่องการเรียนรู้การสอน



ตัวอย่างที่ 7.50 การกระจายข้อมูลปกติโดย NumPy:

```
import numpy as np

x = np.random.normal(170, 10, 250)

print(x)
```



```
[167.97816638 165.59752866 170.06186351 185.65553575 162.53825022
160.88791821 168.61535818 170.93671253 171.78852016 158.64457565
162.89034241 175.44635359 163.3709988 190.39259732 170.79390239
157.34436471 176.76822238 173.37260082 179.88386044 175.09334983
173.3256915 171.4292424 175.93681281 174.51572628 181.71623835
173.16477852 162.57422102 166.85468569 162.70904959 157.6134545
152.86285453 183.20048224 182.33231245 160.3262779 178.17799359
163.33546989 172.49571261 183.71227142 170.59508339 168.59538635
153.66476861 177.44366012 159.96385244 162.48458593 167.18222644
178.99334013 178.73517227 169.41083679 166.56481704 162.53829422
177.50446414 147.72584854 182.5002121 168.02587019 175.08310702
192.39118217 151.08357437 160.06391897 157.25903396 168.31419312
180.14079101 178.00184425 156.61643124 177.30451507 179.44136886
175.65219871 169.90356117 150.07903334 175.15335537 165.81746927
163.78403953 174.25802357 138.52849889 157.83950763 186.1912743
155.16769367 167.49916157 177.4208725 157.26425144 193.57474206
175.39931106 172.18885402 171.17648493 160.51796929 173.12135053
159.64559997 166.5286084 152.10280171 173.92868803 169.31075589
154.34201899 166.49077636 162.64396245 176.67186441 176.38641517
188.36033933 174.67732451 175.42620278 153.8898978 178.05498938
176.3877616 177.89027251 162.9272838 171.89669141 167.10435088
170.11671125 174.30951697 172.61997823 143.7045856 180.33440808
184.99187759 153.7203447 170.88836706 153.2083838 163.34441351
171.18819515 171.0606658 168.26002382 161.63060545 189.0628932
165.54841869 177.37277406 169.72297871 177.86676484 159.48693921
174.10395721 171.33306509 169.94534752 170.76910515 172.85680069
161.77452231 181.09585404 172.8647386 159.80751307 164.27769644
178.64580352 165.77137321 155.78445162 171.48139737 155.27875328
167.69091014 177.63835581 173.53818522 181.63577112 140.04129089
170.95930762 157.48096932 168.47953815 173.59096908 160.19319469
172.68100912 188.27583021 178.11736552 173.23594248 177.91043154
170.63848444 174.80047962 168.48234775 171.62218008 156.66496134
183.10231832 167.23630192 160.88114948 173.06397737 147.60672611
165.2312894 190.3144802 175.32689326 167.75607443 170.35780366
172.19120832 184.38038353 162.96288263 180.29388294 171.44668935
157.02420931 173.86019552 164.1667486 167.33519 175.57612776
169.03922597 173.90499208 177.69090296 202.96387996 184.73947454
165.7632941 165.83326431 179.76501336 169.04316974 150.17933302
170.74665139 180.11376951 165.30293746 170.21348735 169.54797649
168.60419686 164.16072919 181.37913409 180.34109526 173.44256898
155.44728657 184.4280803 173.88744693 168.00225288 162.92228642
175.70231503 175.34551401 177.8456072 173.71854883 181.88671688
165.1544812 171.19431998 168.65636765 177.52461484 175.50825954
166.99630696 187.29160171 173.89325429 186.35748907 177.07255973
182.01033009 169.4934946 175.63455438 170.14610917 170.59243489
190.12347292 172.32339628 169.83297769 168.88185427 167.22813838
166.99268328 186.10401361 168.87422539 163.92086061 169.7333586
170.35709897 176.68323059 163.86585927 188.98468626 176.14343936
178.33498359 188.84552348 179.60725544 172.35763374 158.23782957]
```

6.11 กราฟวงกลม (Matplotlib Pie Charts)

ด้วย Pyplot สามารถใช้ฟังก์ชัน pie() เพื่อวาดแผนภูมิวงกลม:



ตัวอย่างที่ 7.51 แผนภูมิวงกลมอย่างง่าย:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```



แผนภูมิวงกลม สำหรับแต่ละค่าในอาร์เรย์ (ในกรณี [35, 25, 25, 15]) โดยค่าเริ่มต้น การกราฟของ จะเริ่มต้นจากแกน x และเลื่อนทวนเข็มนาฬิกา :

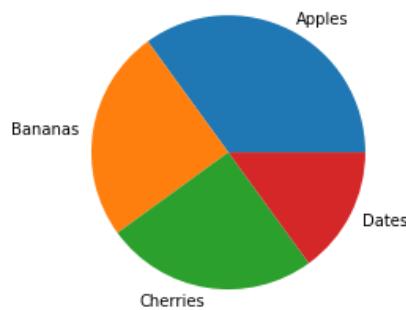
6.11.1 ป้ายกำกับ (Labels)

เพิ่มป้ายกำกับให้กับแผนภูมิวงกลมด้วยพารามิเตอร์ label พารามิเตอร์ label ต้องเป็นอาร์เรย์กับฉลากสำหรับแต่ละส่วน (wedge):



ตัวอย่างที่ 7.52 pie chart ธรรมชาติ

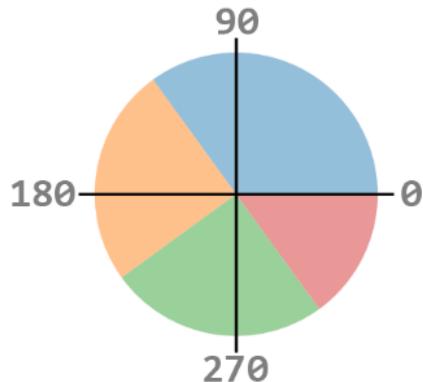
```
import matplotlib.pyplot as plt  
import numpy as np  
  
y = np.array([35, 25, 25, 15])  
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]  
  
plt.pie(y, labels = mylabels)  
plt.show()
```



6.11.1 มุมเริ่มต้น (Start Angle)

ตั้งที่กล่าวไว้ มุมเริ่มต้นเริ่มต้นอยู่ที่แกน x แต่สามารถเปลี่ยนมุมเริ่มต้นได้โดยการระบุ startangle พารามิเตอร์

พารามิเตอร์ Startangle ที่กำหนดไว้กับมุมในองศาที่มุมเริ่มต้นเป็น 0:

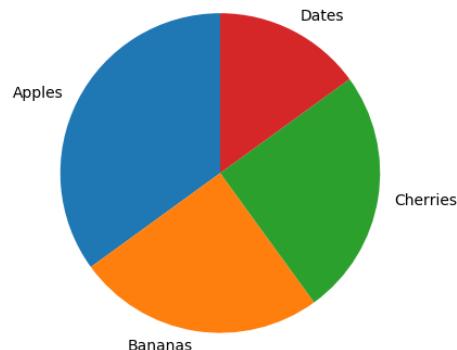


ตัวอย่างที่ 7.52 เริ่มส่วนแรกที่ 90 องศา

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels, startangle = 90)
plt.show()
```



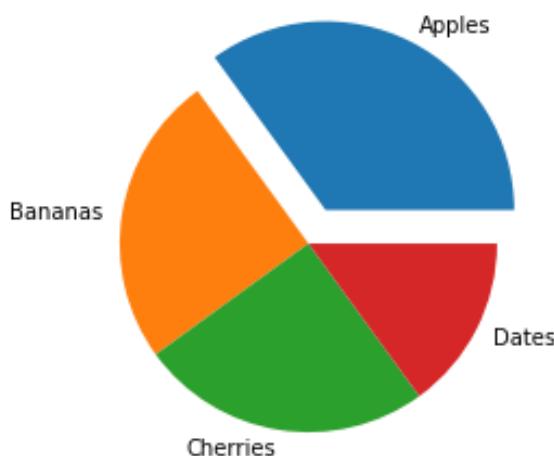


6.11.2 การแยกส่วน (Explode)



ตัวอย่างที่ 7.54 เริ่มส่วนแรกที่ 90 องศา

```
import matplotlib.pyplot as plt  
import numpy as np  
  
y = np.array([35, 25, 25, 15])  
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]  
myexplode = [0.2, 0, 0, 0]  
  
plt.pie(y, labels = mylabels, explode = myexplode)  
plt.show()
```



6.11.3 เจ้า (Shadow)

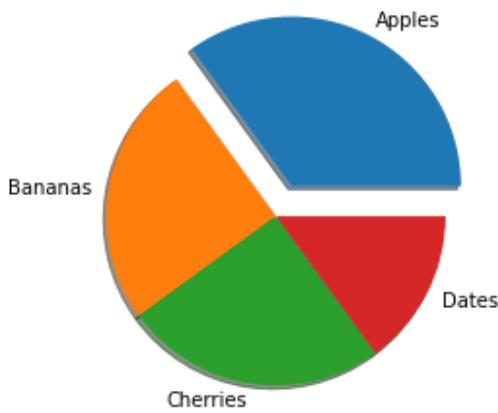
เพิ่มเงาให้กับกราฟวงกลมโดยตั้งค่าพารามิเตอร์ shadows เป็นTrue:



ตัวอย่างที่ 7.55 เพิ่มเงา:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
y = np.array([35, 25, 25, 15])  
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]  
myexplode = [0.2, 0, 0, 0]
```

```
plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)
plt.show()
```



6.11.4 สี (Colors)

Colors พารามิเตอร์ถ้าระบุต้อง เป็นอาร์เรย์ที่มีค่าสำหรับแต่ละส่วน:

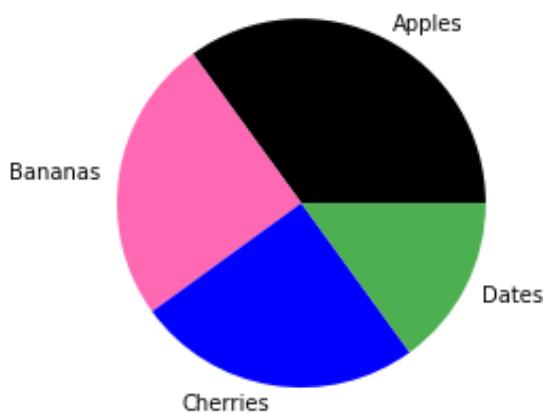


ตัวอย่างที่ 7.56 ระบุสีใหม่สำหรับ wedge แต่ละส่วน:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]

plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```





ใช้ [Hexadecimal color values](#), [140 supported color names](#) อื่นๆ หรือ shortcuts:

- 'r' - Red
- 'g' - Green
- 'b' - Blue
- 'c' - Cyan
- 'm' - Magenta
- 'y' - Yellow
- 'k' - Black
- 'w' - White

6.11.5 คำอธิบาย (Legend)

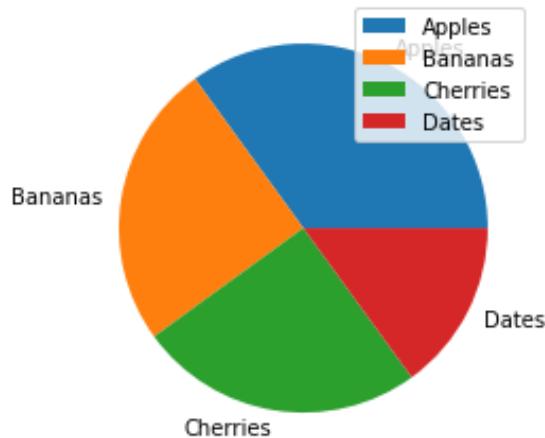


ตัวอย่างที่ 7.57 เพิ่มคำอธิบาย(legend)

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend()
plt.show()
```

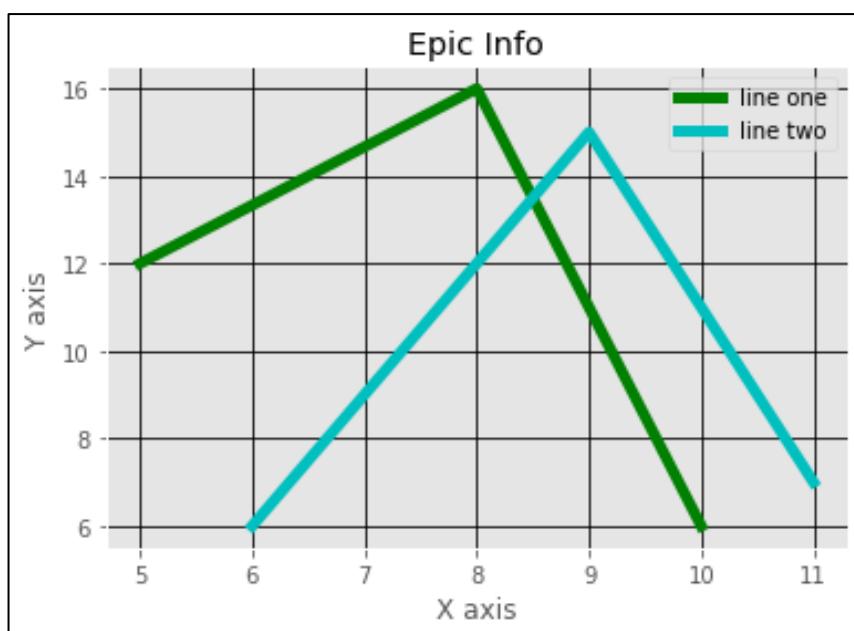


7. สรุปท้ายบท

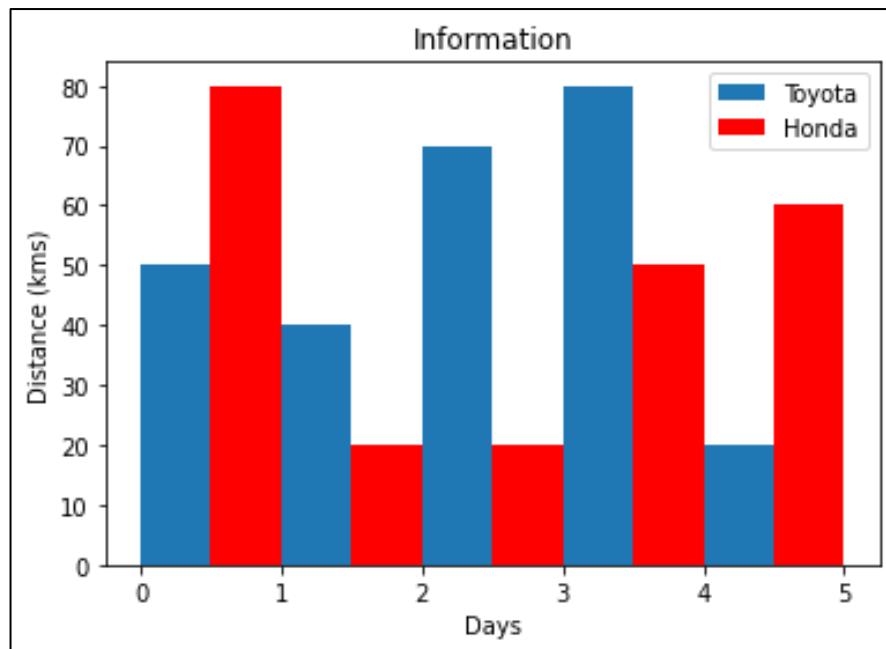
Matplotlib.pyplot เป็นไลบรารี ที่ใช้สำหรับกราฟิก 2 มิติในภาษาโปรแกรมไฟทอน สามารถใช้ในสคริปต์ไฟทอน, เชล์, เซิร์ฟเวอร์เว็บแอปพลิเคชัน และชุดเครื่องมือส่วนต่อประสานกราฟิกกับผู้ใช้อื่น ๆ มีชุดเครื่องมือหลายอย่างที่พร้อมใช้งานเพื่อขยายการทำงานของ python matplotlib บางส่วนเป็นการดาวน์โหลดแยกต่างหากบางรายการ สามารถจัดส่งพร้อมกับซอฟต์แวร์สโคดmatplotlib แต่มีการอ้างอิงภายนอก

8. แบบฝึกหัดท้ายบท

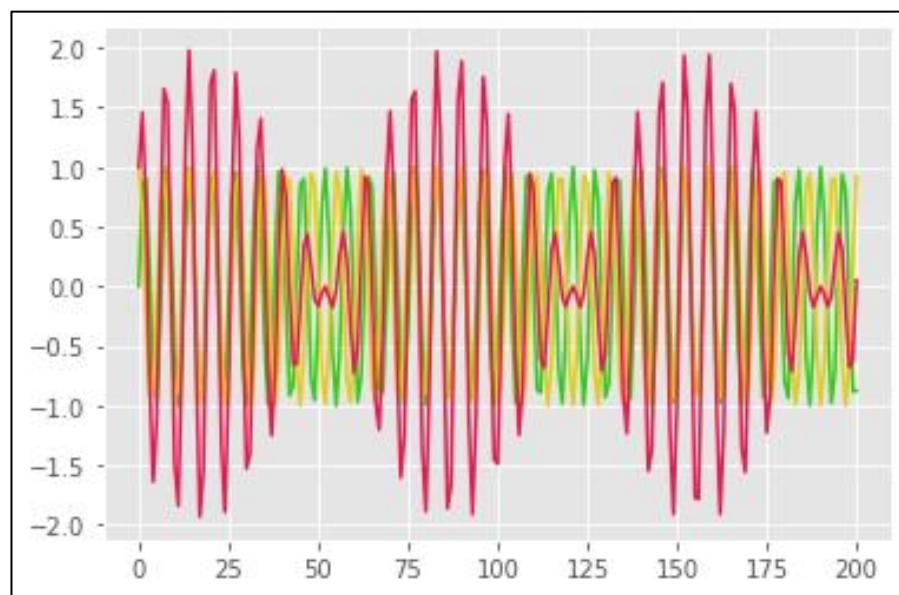
8.1 จงเขียนโปรแกรมเพื่อหาผลลัพธ์ Matplotlib Line ดังภาพ



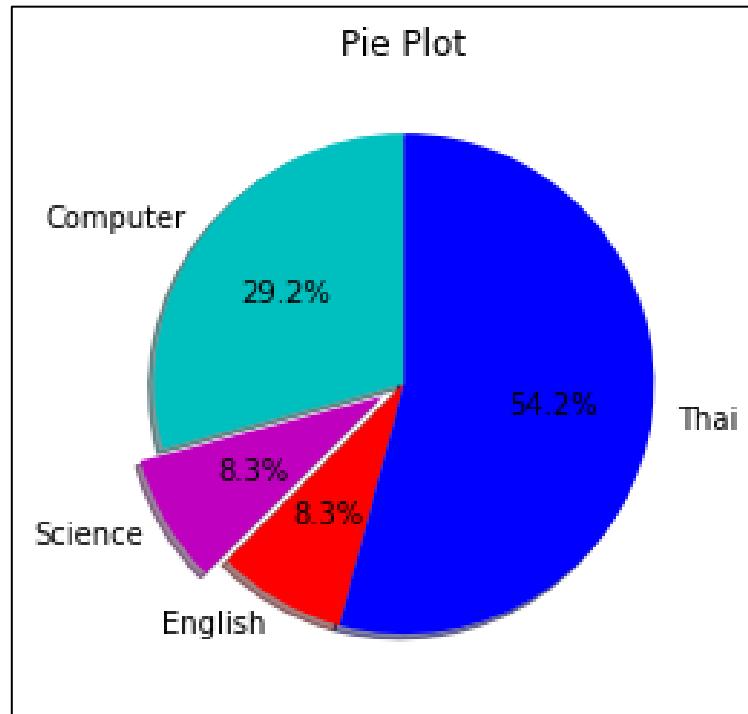
8.2 จงเขียนโปรแกรมเพื่อเพื่อหาผลลัพธ์ Matplotlib Bars เปรียบเทียบระหว่างระยะทางรถยนต์สองคัน Toyota และ Honda ในช่วงเวลา 5 วัน ดังภาพ



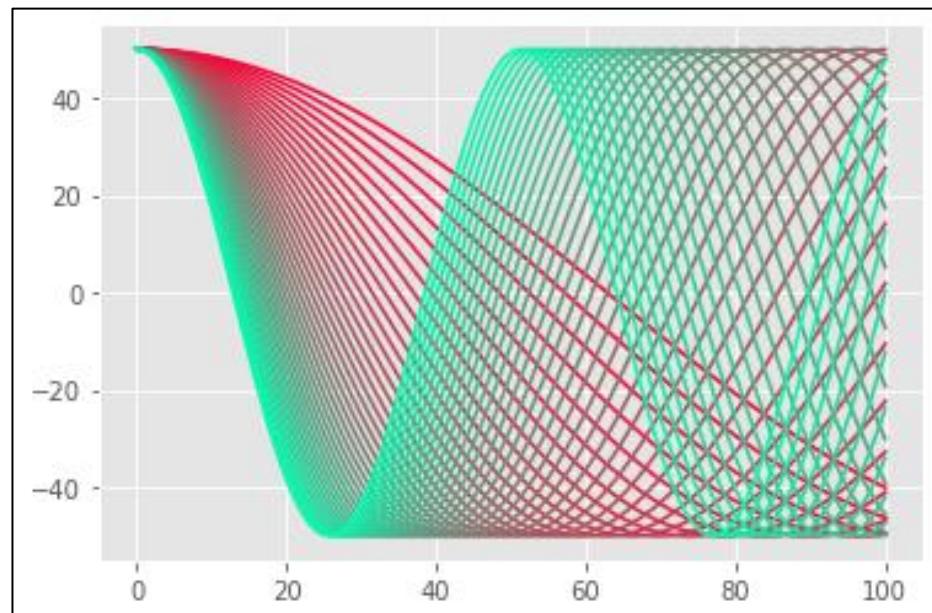
8.3. จงเขียนโปรแกรมเพื่อหาผลลัพธ์ Matplotlib Line ใส่เป็นตัวเลขส่วนผสมของแมสีสามสี (แดง,เขียว,น้ำเงิน) โดยมีค่าตั้งแต่ 0 ถึง 1 ดังภาพ



8.4. จงเขียนโปรแกรมเพื่อหาผลลัพธ์ Matplotlib Pie Charts ดังภาพ



8.5 จงเขียนโปรแกรมเพื่อหาผลลัพธ์ กราฟหลายๆ อันที่มีการไล่สีได้ง่าย โดยใช้ for แล้วให้เลขค่า สีเปลี่ยนไปในแต่ละรอบ ดังภาพ



บรรณานุกรม

- กิตติภณ พลการ, กิตติภพ พลการ, สมชาย ประสิทธิ์จุตระกูล, สุกรี สินธุภิญโญ. (2561). Python 101. ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย : โรงพิมพ์แห่งจุฬาลงกรณ์มหาวิทยาลัย.
- บัญชา ปะสีลักษณะ. (2562). การเขียนโปรแกรม Python สำหรับผู้เริ่มต้น. กรุงเทพฯ : ซีเอ็ด ยูเคชั่น.
- ณัฐรัตน์ คำภักดี. (2562). คู่มือเขียนโปรแกรม Python ฉบับปรับปรุง. กรุงเทพฯ : โพรไวชั่น.
- สุดา เอียร์มนตรี. (2563). คู่มือเรียนโปรแกรม Python ฉบับสมบูรณ์. นนทบุรี : ไอเดีย.
- สุชาติ คุ้มมะณี. (2561). เชี่ยวชาญการเขียนโปรแกรมด้วยไพธอน. สืบคันเมื่อวันที่ 12 เมษายน 2562 , จาก<https://www.theconfig.me/p/programming-expert-with-python.html>
- Allen B. Downey. (2016). THINK PYTHON: HOW TO THINK LIKE A COMPUTER SCIENTIST. Retrieved January 22 2020, <https://pythonbooks.org/think-python-how-to-think-like-a-computer-scientist/>
- Al Sweigart . (2020). Automate the Boring Stuff with Python Retrieved January 22 2020, <https://pythonbooks.org/free-books/>
- Anaconda (2021). Anaconda Individual Edition (Version Anaconda3-2021.05-Windows-x86_64) [software]. Retrieved from <https://www.anaconda.com/products/individual>
- Al Sweigart . (2018). Cracking Codes with Python Retrieved November 2020, from <http://inventwithpython.com/cracking/>
- Bradley N. Miller. (2012). PROBLEM SOLVING WITH ALGORITHMS AND DATA STRUCTURES USING PYTHON. Retrieved January 30 2020, <https://pythonbooks.org/problem-solving-with-algorithms-and-data-structures-using-python-second-edition/>
- Charles Russell Severance. (2017). PYTHON FOR EVERYBODY: EXPLORING DATA IN PYTHON 3 . Retrieved January 22 2020,<https://www.py4e.com/book>
- Google . (2021). Google Colab [application software]. Retrieved September 2021, from <https://colab.research.google.com/notebooks/intro.ipynb>
- JetBrains s.r.o. (2021). PyCharm (Version: 2021.2.2). [application software]. Retrieved 1 March 2021, form <https://www.jetbrains.com/pycharm/>
- Jupyter Notebook. (2021). Installing the Jupyter Software . [application software]. Retrieved October 06 2021, form <https://jupyter.org/install>
- Learn Python. (2020). Retrieved January 1 2020, form <https://www.learnpython.org/>
- Nigel George. (2017). MASTERING DJANGO: THE COMPLETE GUIDE TO DJANGO 1.8 LTS. Retrieved October 06 2020, form <https://djngobook.com/the-django-book/>
- Python. (2020). Retrieved January 1 2020, form <https://www.python.org/>

- PythonBooks. (2020). Retrieved January 1 2020, from
<https://wiki.python.org/moin/PythonBooks>
- Python Software Foundation. (2020). Python (Version 3.9.1). [application software].
Retrieved January 01 2021, from <https://www.jetbrains.com/pycharm/>
- Visual Studio Code. (2020). (Version VSCodeUserSetup-x64-1.64.2). [application software]. Retrieved January 15 2021, from
<https://code.visualstudio.com/docs/?dv=win64user>
- W3School. (2020). Python Tutorial. Retrieved November 2020, from
<https://www.w3schools.com/python/default.asp>
- Zed A. Shaw. LEARN MORE PYTHON 3 THE HARD WAY: THE NEXT STEP FOR NEW PYTHON PROGRAMMERS. Retrieved January 22 2020,
<https://learncodethehardway.org/more-python-book/>

ดัชนี

A	
Access List Items	93
Access Set Items	103
Add List Items	95
Add Set Items	104
Add Sets	104
Alpha	207
Anaconda	14, 15, 20, 21, 22, 23, 26, 174
Arguments	153
Arithmetic Operators	75
B	
Bitwise Operators	82
break Statement	137
Built-in Data Types	51
Built-in Math Functions	84
C	
Change List Items	94
Comment	43, 44
Community	7
Compare Plots	202
Comparison Operators	76
Compound statement	42
Compound statement	42
Conda	14, 21, 22, 23, 24
continue Statement	140
Copy Dictionaries	117
count()	93
Creating Scatter Plots	202
D	
def	152
Default Parameter Value	156
Default X-Points	178
Dictionarie	124
E	
Dictionaries	108
Dictionary Items	109
Display Multiple Plots	196
F	
Edu	7, 12
Else in for Loop	141
else Statement	138
escape	67
Escape Character	67
extend()	93
G	
Format Strings fmt	181
H	
global Keyword	50
Google Colab	31
I	
IDLE	2, 3, 5
immediate mode	2
Indentation	45
index()	93
input()	43, 143, 151
insert()	93
J	
Join Sets	107
Join Tuples	101
Jupyter Notebook	14, 26, 29, 30, 31, 41
K	
Keyword Arguments	155
L	
Lambda	160
len()	93

Lists.....	124
Logical Operators	77
Loop Dictionaries	115
Loop Sets.....	107
Loop Tuples	100
<i>M</i>	
Marker Color.....	182
Marker Reference	180
Marker Size	181
Math Module.....	85
Matplotlib Bars	209
Matplotlib Codebase.....	173
Matplotlib Grid.....	193
Matplotlib Histograms.....	213
Matplotlib Labels and Title	190
Matplotlib Line	185
Matplotlib Markers.....	179
Matplotlib Pie Charts	215
Matplotlib Plotting.....	176
Matplotlib Pyplot	175
Matplotlib Scatter	202
Matplotlib Subplots.....	196
Membership Operators.....	81
Multiple Points	178
<i>N</i>	
Nested Dictionaries.....	118
Nested Loops.....	142
<i>O</i>	
Operators	74
<i>P</i>	
Passing a List as an Argument.....	157
Plotting Without Line.....	177
pop().....	93
Pre-defined function	151
print().....	42, 59, 73, 76, 125, 151
PyCharm.....	6, 12, 41
Pyplot	175
Python14, 20, 31, 41, 42, 43, 44, 45, 46,	
47, 48, 49, 50, 51, 53, 54, 57, 59, 60,	
63, 67, 70, 73, 74, 75, 76, 77, 78, 79,	
81, 82, 83, 84, 85, 93, 124, 174, 223	
<i>R</i>	
range() Function	140
Recursion	159
Remove Set Items	105
remove().....	93
reverse()	93
<i>S</i>	
script mode	2
Set124	
Set Methods.....	108
Sets	102
Single statement.....	42
Slicing Strings	62
sort().....	93
Statement.....	42, 129, 130, 136, 137
String Methods	67
<i>T</i>	
Tuple	124
Tuple Items.....	98
Tuples.....	97
<i>U</i>	
Update Tuples	99
User-defined function.....	151
<i>V</i>	
value-returning function	151
Visual Studio Code.....	36
void function	151
VS Code	36

ก	
กราฟแท่ง.....	209
กราฟแบบกระจาย.....	202
กราฟแบบไม่มีเส้น	177
กราฟย่ออย	196
กราฟวงกลม	215
กราฟเส้น	185
การกำหนดขอบเขตของคำสั่ง.....	45
การกำหนดค่าให้ชนิดข้อมูล.....	53
การกำหนดฟอนต์และรีม.....	13
การกำหนดสตริงหลายบรรทัด.....	60
การแก้ไขสตริง	63
การขยาย List ด้วยเมธอด extend.....	95
การเข้าถึงค่าในทูเพลิ	99
การเข้าถึงเซต.....	103
การเข้าถึงตำแหน่งข้อมูล Lists.....	93
การเข้าถึงสมาชิกของดิกชันนารี	110
การเขียนคำอธิบายโค้ด.....	43
การเขียนคำอธิบายแบบบรรทัดเดียว	43
การเขียนคำอธิบายแบบหลายบรรทัด	44
การคัดลอกดิกชันนารี.....	117
การคุณทูเพลิ	101
การจัดการ conda	21
การจัดการ Python.....	23
การจัดการแพ็คเกจ	23
การจัดลำดับของอาร์กิวเมนต์	152
การใช้งาน Google Colab	31
การใช้งาน IDLE เป็นต้น	5
การใช้อักษรพิเศษ.....	67
การตรวจสอบเวอร์ชัน Matplotlib.....	174
การต่อสตริง.....	64
การติดตั้ง Matplotlib.....	173
การติดตั้งโปรแกรม Visual Studio Code	36
การติดตั้งและใช้งาน Jupyter Notebook	14
การติดตั้งและใช้งาน Pycharm.....	6
การแทรก Lists ด้วยเมธอด insert	95
การนับความยาวของสตริง.....	61
การนำเข้า Matplotlib.....	174
การนำเข้าโมดูลด้วยคำสั่ง from...import	164
การนำเข้าโมดูลด้วยคำสั่ง import	163
การประกาศและใช้งาน List	92
การเปลี่ยนค่าของ List	94
การเปลี่ยนช่วงค่าของ Lists	94
การเปลี่ยนรายการดิกชันนารี.....	111
การแปลงชนิดข้อมูล	57
การแปลงประเภทข้อมูล (Type Conversion)	56
การเพิ่ม Lists	95
การเพิ่มรายการเซต.....	104
การเพิ่มรายการดิกชันนารี	112
การรวมทูเพลิ	101
การระบุชนิดข้อมูลพร้อมข้อมูล	53
การรับข้อมูลทางคีย์บอร์ดด้วยฟังก์ชัน input()	43
การรับชนิดข้อมูล	52
การเริ่มต้น conda	21
การเรียกใช้ฟังก์ชัน	152
การเรียกใช้ฟังก์ชันแบบ Recursion	159
การลบ Lists	96
การลบด้วยคำหลัก Del.....	96
การลบรายการดิกชันนารี	112
การล้าง Lists	97
การวนซ้ำดิกชันนารี	115
การสร้างกราฟกระจาย	202
การสร้างกราฟด้วย Matplotlib	173, 174
การสร้างกราฟแท่ง.....	209
การสร้างเซต.....	103
การสร้างป้ายกำกับสำหรับกราฟ.....	190
การสร้างฟังก์ชันแบบไม่ส่งค่ากลับ	151
การสร้างโมดูลในภาษาไฟทอน	162
การสร้างและใช้โมดูล	162
การแสดงผลข้อมูลด้วยฟังก์ชัน print()	42
การอัปเดททูเพลิ.....	99
กำหนดค่าให้กับหลายตัวแปร.....	48
กิจกรรม.....	23, 86, 124, 143, 166

ก	ตัวดำเนินการเอกลักษณ์.....	79
ขนาดมาร์กเกอร์	181	
ข้อมูลชนิดตัวเลข	54	
ข้อมูลแบบลำดับรายการ	92	
ค		
ความยาวทูเพิล	97	
คำสั่ง if...elif...else	133	
คำสั่งควบคุมแบบทำซ้ำ	136	
คำสั่งควบคุมแบบมีเงื่อนไข	130	
คำสั่งควบคุมแบบลำดับ	129	
คำสั่งที่เขียนในโปรแกรมไพทอน	42	
คำสั่งวนซ้ำด้วย for loop.....	138	
คำสั่งวนซ้ำด้วย while loop.....	136	
คูณทูเพิล.....	101	
เครื่องหมายอ้างอิง.....	180	
จ		
จำนวนเชิงซ้อน (Complex).....	56	
จำนวนเต็ม (int)	54	
ช		
ชนิดข้อมูลแบบ Built-in.....	51	
ชนิดข้อมูลสตริง	59	
ชื่อตัวแปร	47	
ด		
ดิกชันนารีที่ซ้อนกัน	118	
ต		
ตรวจสอบสตริง	61	
ตัวจัดการสภาพแวดล้อม	22	
ตัวดำเนินการ	74	
ตัวดำเนินการกำหนดค่า.....	74	
ตัวดำเนินการคณิตศาสตร์.....	75	
ตัวดำเนินการตรรกศาสตร์.....	77	
ตัวดำเนินการตรวจสอบสมาชิก	81	
ตัวดำเนินการเบรียบเทียบ	76	
ตัวดำเนินการระดับบิต.....	82	
ท		
แทนที่สตริง	64	
บ		
บลีน.....	52, 70, 73, 98, 99, 103	
ป		
ประเมินค่าและตัวแปร	70	
เปรียบเทียบกราฟ	202	
พ		
พารามิเตอร์ 140, 141, 151, 152, 153, 155,		
156, 158, 176, 177, 192, 194, 215,		
216, 218		
เพิ่มรายการที่ทำซ้ำได้	105	
ฟ		
ฟังก์ชัน 30, 42, 43, 50, 51, 52, 53, 54, 57,		
59, 61, 65, 70, 73, 74, 84, 85, 92, 93,		
97, 100, 101, 117, 127, 140, 143,		
151, 152, 153, 155, 156, 157, 160,		
167, 193, 194, 196, 201, 202, 210		
ฟังก์ชันเกี่ยวกับตัวเลข.....	84	
ฟังก์ชันของกราฟย่ออย	197	
ฟังก์ชันแบบไม่ส่งค่ากลับ.....	151	
ฟังก์ชันแบบส่งค่ากลับ.....	151, 158	
ฟังก์ชันและโมดูล.....	151	
ฟังก์ชันสามารถคืนค่าบลีน	73	
ม		
เมรอดของดิกชันนารี.....	119	
เมรอดและฟังก์ชันที่ใช้กับ List	92	

เมอรอดและฟังก์ชันที่ใช้งานกับชนิดข้อมูลทุกเพิล	101	วนลูปผ่านสตริง 61
ไมดูลนามແຟັງ 165		ສ
ຍ		ສຕຣິງ.... 44, 47, 49, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 71, 98, 99, 101, 103, 174
ແຍກສຕຣິງ..... 64		ສ້າງຂໍ້ເຮືອງສໍາຫັບກາຟ 191
ຮ		ສ້າງຖຸເພີລດ້ວຍຮາຍກາຣເດີຍວ 98
ຮາຍກາຣຂໍ້ອມຸລແບບ Lists 92		ສັນລັກຂໍ້ນກາຣແສດງຮູປແບບชนິດຂໍ້ອມຸລ 65
ຮາຍກາຣຂໍ້ອມຸລແບບເຊດ 102		ສືມາർກເກໂວ່ງ 182
ຮາຍກາຣຂໍ້ອມຸລແບບດົກໜ້າຮີ 108		ເສັ້ນກົດຂອງກາຟ 193
ຮາຍກາຣຂໍ້ອມຸລແບບທຸເພີລ 97		ທ
ຮູປແບບສຕຣິງ..... 65		ໄໂທມດສຄຣິປີຕ 2
ຮູປແບບສຕຣິງ fmt 181		ໄໂທມດອິມມີເດີຍທ 2
ລ		ອ
ລບດັ່ນນີ້ທີ່ຮະບຸດ້ວຍເມຮອດ pop() 96		ອນຸໝາຕຮາຍກາຣທີ່ໜ້າກັນ 97
ລບຮາຍກາຣທີ່ຮະບຸດ້ວຍເມຮອດ remove().... 96		ອາກົວມີເມນີຕ 42, 66, 111, 151, 152, 153, 155, 156, 157, 160, 182, 185, 186, 188, 197, 203, 205, 207, 209
ລັກຂໍ້ນະຂອງຟັງກຳ 151		ອາເຮຍື່ນສຕຣິງ 60
ລຳດັບກາຣປະມວລຜລອງຕົວດຳເນີນກາຣ 84		ອ
ເລັກທຸນິຍມ (Float)..... 55		ອີສໂໂຕແກຣມ 213
ຈ		
ວນໜ້າໜາຍເລີດໜີ 100		
ວນລູປ (While)..... 100		
ວນລູປຜ່ານຖຸເພີລ 100		

ประวัติผู้เขียน

ชื่อ-สกุล

ดร. ลาภณ์ ดุลยชาติ

ประวัติการศึกษา



ปริญญาเอก	ปรัชญาดุษฎีบัณฑิต (ปร.ด.) สาขาวิชาคอมพิวเตอร์ศึกษา
ปริญญาโท	ศิลปศาสตรมหาบัณฑิต (ศศ.ม.) สาขาวิชาสารสนเทศศาสตร์
ปริญญาตรี	มหาวิทยาลัยสุโขทัยธรรมาธิราช วิทยาศาสตรบัณฑิต (วท.บ.) สาขาวิชาเทคโนโลยีสารสนเทศ

ประวัติการทำงาน

ปัจจุบัน	อาจารย์ประจำสาขาวิชาคอมพิวเตอร์ คณะศึกษาศาสตร์และนวัตกรรมการศึกษา มหาวิทยาลัยกาฬสินธุ์
----------	---

งานวิจัย

พ.ศ. 2563	การส่งเสริมทักษะศตวรรษที่ 21 (4Cs) ด้านการสร้างสรรค์และนวัตกรรม ด้วยการจัดการเรียนรู้แบบสร้างสรรค์เป็นฐาน ในรายวิชาโครงการพิเศษด้านคอมพิวเตอร์ศึกษา สาขาวิชาคอมพิวเตอร์ มหาวิทยาลัยกาฬสินธุ์
พ.ศ. 2558	แอ��พพลิเคชันเพื่อการเรียนรู้เครื่องดนตรีโปงลางบันอุปกรณ์แท็บเล็ต
พ.ศ. 2554	วิจัยในชั้นเรียน : การจัดการเรียนการสอน เรื่อง การวิเคราะห์และออกแบบ อัลกอริทึม ด้วยเทคนิคกระบวนการกลุ่ม
พ.ศ. 2552	ระบบสารสนเทศแหล่งท่องเที่ยวทางวัฒนธรรม กรณีศึกษา : วัดพุทธโนนุมิต(ภูค่าว)
พ.ศ. 2551	การพัฒนาศักยภาพบุคลากรชุมชนเพื่อสร้างภูมิคุ้มกันด้านเศรษฐกิจชุมชนอย่างยั่งยืนของจังหวัดกาฬสินธุ์: กรณีศึกษากลุ่มศตรีท่อผ้าบ้านกุดหว้า อำเภอภูนิหารายณ์ จังหวัดกาฬสินธุ์
พ.ศ. 2549	การจัดการที่พักทางวัฒนธรรม กรณีศึกษา : ชุมชนผู้ไทยบ้านโน彭
พ.ศ. 2547	การพัฒนาระบบสารสนเทศผ้าไหมแพรฯว้าบ้านโน彭