

NEIS0708 Script Language Programming

SONGPOL RUENSUK
SONGPOL@MUT.AC.TH |

Contents

Network programming.....	7
Python	7
Python คืออะไร?	7
ข้อดี และข้อเสียของ Python	7
Setting UP Python.....	8
ติดตั้ง Python กัน (Windows).....	8
ติดตั้ง Python กัน (Mac OS).....	11
ทดสอบการใช้งาน Python (Windows)	14
ทดสอบการใช้งาน Python (Mac OS)	15
เริ่มต้นใช้งาน Python ด้วย IDLE.....	16
ลองเล่น Python Shell เบาๆ	17
ลองสร้างไฟล์ เพื่อรันใน Python Shell กัน.....	17
ติดตั้งเครื่องมือช่วยในการพัฒนา Python.....	18
ติดตั้ง PyCharm (Windows)	19
ติดตั้ง PyCharm (Mac OS).....	26
General Syntax	30
Creating a main script	30
Understanding whitespace in Python.....	30
Commenting code.....	31
Creating and using functions	31
Creating and using objects	32

การรับค่าจาก keyboard.....	33
Using numbers.....	34
Using strings	34
Conditionals.....	36
Selecting code with if and else conditional statements	36
รูปแบบการใช้งาน if	36
Setting multiple choices with elif.....	36
รูปแบบการใช้งาน if...elif...else	37
Loops	38
Creating loops with while	38
รูปแบบการใช้งาน while	38
Iterating with for	39
รูปแบบการใช้งาน for	39
range() function.....	39
Enumerating iterators.....	40
enumerate() function.....	40
Controlling loop flow with break, continue, and else	40
break	40
continue	40
else	40
pass	40
Operators.....	41
Performing simple arithmetic.....	41
Comparing values	42

Membership operation	42
Operating on Boolean values	43
Operating on parts of a container with the slice operator.....	43
Aggregating values with lists and tuples.....	44
Tuple	44
การประมวลผล tuple.....	44
List	44
การประมวลผล list.....	44
list.append(x).....	44
list.extend(L)	44
list.insert(i, x).....	44
list.remove(x).....	45
list.clear().....	45
list.index(x).....	45
list.count(x).....	45
list.sort()	45
list.reverse()	45
tuple <=> list.....	45
Creating associative lists with dictionaries.....	46
dictionary	46
การประมวลผล dictionary	46
Modules	47
การเรียกใช้ module.....	47
import module	47

from module import function	47
Number Type Conversion.....	49
int(x)	49
str(x).....	49
float(x)	49
String Methods.....	49
str.capitalize()	49
str.center(width [, fillchar])	49
str.find(sub).....	49
str.format()	50
str.isalnum()	50
str.isalpha().....	50
str.islower().....	50
str.isnumeric().....	50
str.isupper().....	50
str.ljust(width [, fillchar]).....	50
str.lower().....	50
str.join(iterable).....	50
str.lstrip().....	51
str.replace(old, new[, count]).....	51
str.rjust(width [, fillchar])	51
str.rstrip()	51
str.split(sep=None, maxsplit=-1).....	51
str.strip().....	51

str.upper()	51
str.zfill(width)	51
Error and Exceptions.....	53
Syntax Errors	53
Exceptions	53
Exception កំគុចទ្វាគ	56
Raising exceptions.....	56
Time	57
time.localtime([secs])	57
time.sleep(secs)	57
time.strftime(format[, t]).....	57
time.time()	58
Functions.....	60
Defining functions.....	60
Using lists of arguments.....	61
Using named function arguments	63
Returning values from functions	64
Classes.....	65
Class	65
Class variable	65
Data member.....	65
Instance variable	65
Instance	65
Method.....	65

Object.....	66
การสร้าง class	66
self.....	67
__init__.....	67
การนำ class ไปใช้งาน	67
File I/O.....	68
open(file [, mode]).....	68
f.read([size])	68
f.readline().....	69
f.seek(index).....	69
f.write(value).....	69
f.tell().....	69
f.close().....	69

Network programming

Programming คือ การเขียนโปรแกรม การเขียนเพื่อควบคุมการทำงานต่าง ๆ ที่เราต้องการ อาจมีการติดต่อฐานข้อมูล เพื่อบันทึกข้อมูล มีการตรวจสอบความถูกต้อง ก่อนที่จะทำงานได้ ๆ ในขั้นตอนต่อไป หรืออาจเขียนเว็บไซต์ เพื่อแสดงข้อมูล จัดเก็บข้อมูลสำคัญไว้บน database server

Network คือ ระบบเครือข่าย ที่มีการเชื่อมกันของคอมพิวเตอร์ที่มากกว่า 1 เครื่อง

Network programming คือการเขียนโปรแกรมบนระบบเครือข่าย เราสามารถที่จะรับ-ส่งไฟล์จากเครื่อง client many เครื่อง server ได้ หรือตรวจสอบข้อมูลเว็บไซต์ต่าง ๆ การตรวจสอบข้อมูลได้ ๆ ที่ต้องการได้

การเขียนโปรแกรม เราสามารถเลือกภาษาอะไรมาเขียนก็ได้ ไม่ว่าจะเป็น Java หรือ C# แต่ใน การศึกษาของเรานั้นจะใช้ภาษา Python

Python

Python คืออะไร?

Python คือภาษาที่ใช้ในการเขียนโปรแกรมภาษาหนึ่ง ที่ถูกพัฒนาขึ้นมาเพื่อให้สามารถใช้งานได้กับระบบปฏิบัติการต่าง ๆ ไม่ยึดติดกับระบบปฏิบัติการใด ๆ (Cross platform) คล้ายกับ PHP และมี license เป็น Python Software Foundation License ซึ่งทำให้การพัฒนาโปรแกรมด้วย Python นั้นไม่มีค่าใช้จ่ายได้ ๆ

Python เป็นภาษา Dynamic Object-Oriented Programming ที่ถูกพัฒนาขึ้นโดย Guido von Rossum ในปี ค.ศ. 1990 มีการแปลงคำสั่งแบบ interpreter (การแปลงคำสั่งเป็น machine code ทีละบรรทัดระหว่างที่โปรแกรมทำงาน) ไม่มีการกำหนดชนิดของตัวแปร พารามิเตอร์ พักรชั่น หรือเมธอดใด ๆ ทำให้การเขียนโปรแกรมมีความยืดหยุ่น โดย Python จะทำการติดตามตลอดเวลาว่า ตัวแปรนี้จะเป็นชนิดอะไร เอง

ข้อดี และข้อเสียของ Python

ข้อดีของ Python มีดังนี้

1. ทำงาน cross platform
2. เป็น open source
3. เป็น dynamic typing คือชนิดของข้อมูลสามารถเปลี่ยนแปลงได้ตลอดเวลา
4. จัดการหน่วยความจำให้อัตโนมัติ
5. มีโมดูลต่าง ๆ เช่น โมดูลเกี่ยวกับ Regular Expression מודูลด้านภาพกราฟฟิก

6. รองรับการเขียนโปรแกรมเชิงวัตถุ (OOP)
7. มีการทำงานแบบ Interpreter

ข้อเสียของ Python มีดังนี้

1. เพราะมีการทำงานแบบ interpreter ซึ่งคือการแปลงคำสั่งในแต่ละบรรทัดของโปรแกรมไปเป็นคำสั่งเครื่อง (machine code) ในระหว่างที่โปรแกรมทำงาน ทำให้ทำงานช้า
2. โอกาสที่จะเกิด runtime error มีมากกว่า เช่น การเรียกใช้ตัวแปรที่ไม่ได้ประกาศไว้

เมื่อเรารู้ข้อดี และข้อเสียของ Python ไปแล้ว เราจะไฝรอข้าอีกต่อไป เราจะเร่งรุดหน้า ในการศึกษาเกี่ยวกับ Python ให้ได้มากที่สุด แต่ก่อนที่เราจะไปศึกษา python เราต้องติดตั้ง Python กันก่อนซินะ

Setting UP Python

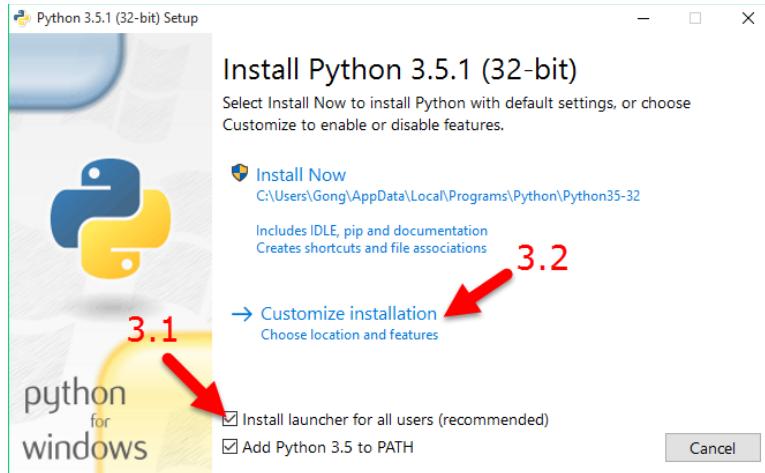
ติดตั้ง Python กัน (Windows)

1. ทำการ download Python จาก <https://www.python.org/downloads/> ในขณะที่ทำการดาวน์โหลด version ล่าสุดคือ Python 3.5.1 และ Python 2.7.11 ทั้งนี้ให้เลือก version 3.5.1

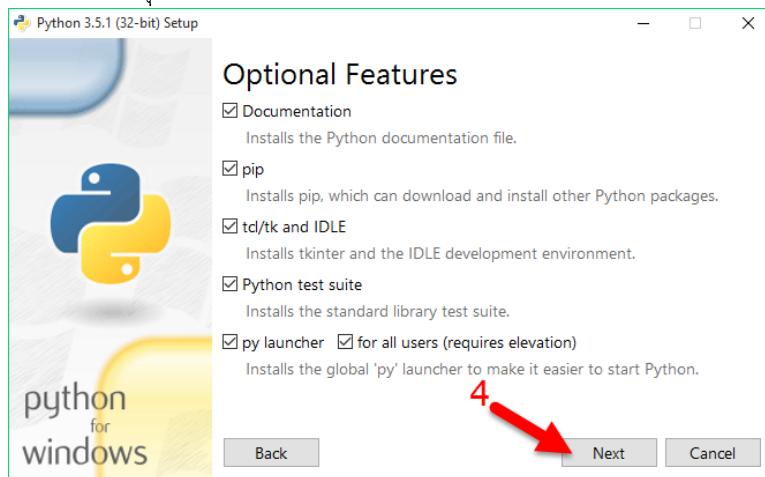


2. ให้ทำการ double click  python-3.5.1 เพื่อเริ่มการติดตั้ง

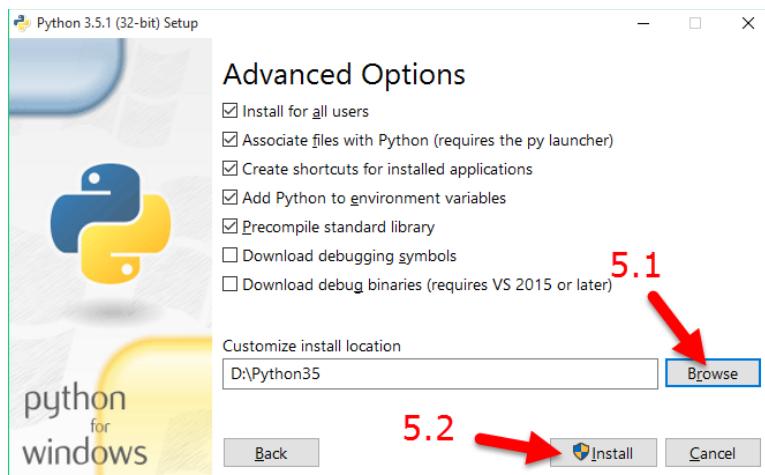
3. เริ่มต้นการติดตั้ง ทำการเลือก Add Python 3.5 to PATH เพื่อให้สามารถใช้งาน Python ได้จาก path ได ๆ บนเครื่องของเราได้ จากนั้นกด Customize installation



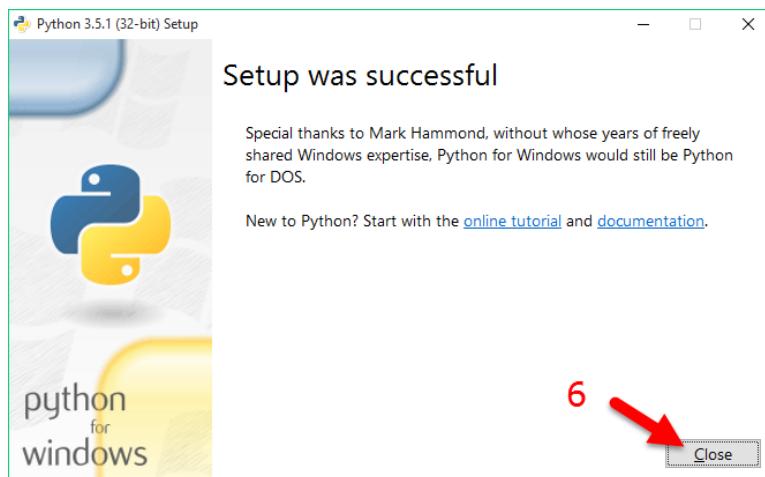
4. เลือกตามรูป จากนั้นกดปุ่ม Next



5. เลือกตามรูป จากนั้นเลือกสถานที่สำหรับติดตั้ง และกดปุ่ม Install



6. กดปุ่ม Close เพื่อเสร็จสิ้นการติดตั้ง



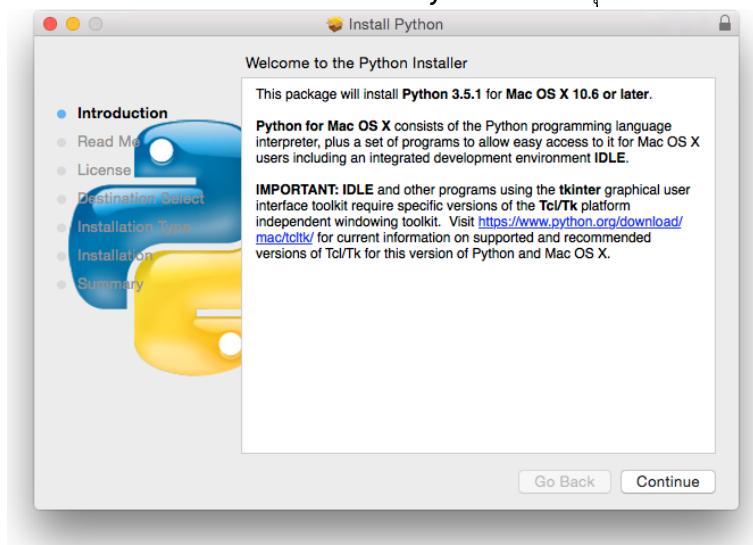
หรูหาร้าไปอีกขั้น กับการติดตั้งบน MacOS

ຕັດຕັ້ງ Python ກັບ (Mac OS)

1. ທຳການ download Python ຈາກ <https://www.python.org/downloads/> ໃນຂະໜາດທີ່ທໍາເອກສາຮັບເລືອດຕື່ມືຖຸດີຂອງ Python 3.5.1 ແລະ Python 2.7.11 ທີ່ຈຶ່ງໄດ້ເລືອດ version 3.5.1



2. ໄທ້ທຳການ double click python-3.5.1-macosx10.6.pkg ເພື່ອເຮີມຕິດຕັ້ງ
3. ເຮີມກາຣົດຕິດຕັ້ງ ເປັນໜ້າ introduction ເກີຍວັກນ Python ໃຫ້ກດປຸ່ມ continue



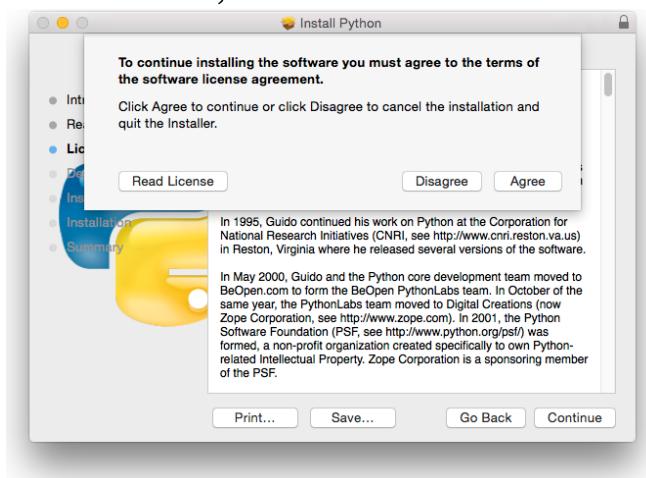
4. ດັດໄປເປັນໜ້າຂໍອມູລຄຳຄົມຕ່າງ ຖ້າໃຫ້ກດປຸ່ມ continue



5. ถัดไปเป็นข้อมูลเกี่ยวกับไลเซนส์ของ Python ให้กดปุ่ม continue



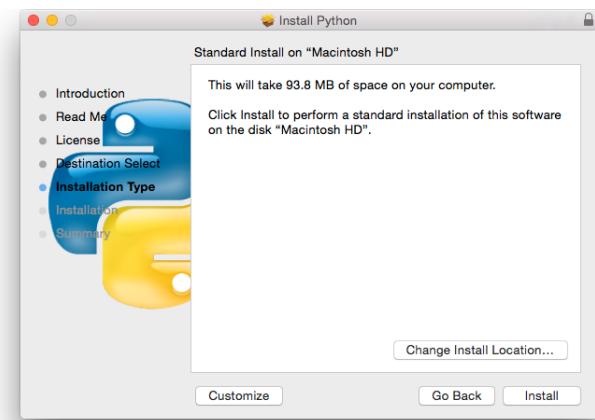
6. กด Agree เพื่อยอมรับเงื่อนไขต่อๆ



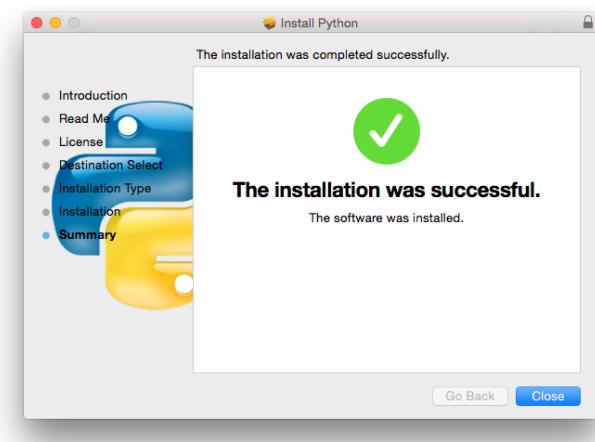
7. เลือกสถานที่สำหรับการติดตั้ง จากนั้นกดปุ่ม continue



8. สรุปข้อมูลต่าง ๆ ก่อนการติดตั้ง จากนั้นกด Install

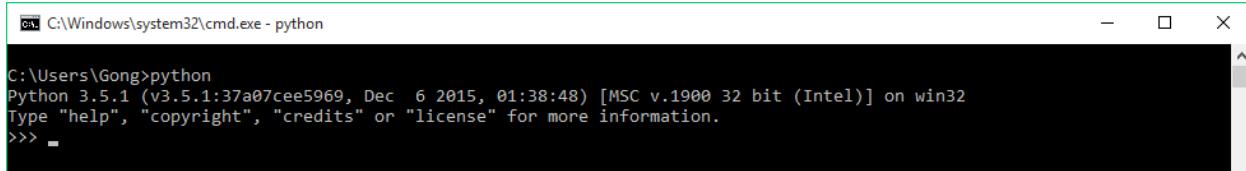


9. กดตั้งเรียบร้อยกด Close



ทดสอบการใช้งาน Python (Windows)

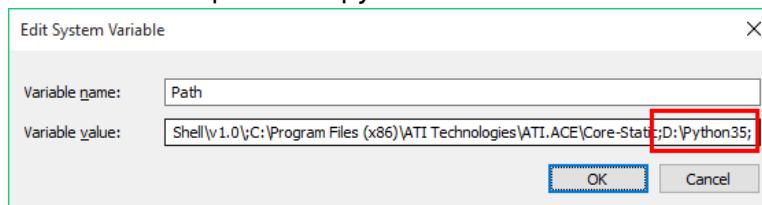
- ให้เปิด command prompt ขึ้นมา แล้วพิมพ์ python ลงไป หากสามารถใช้งานได้ จะแสดงดังรูป แต่หากไม่ได้ ให้ทำการขั้นตอนถัดไป



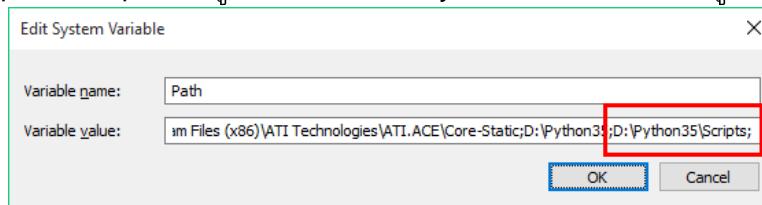
C:\Windows\system32\cmd.exe - python

```
C:\Users\Gong>python
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:38:48) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> -
```

- ให้ทำการกำหนด Path ให้กับ windows โดย click ขวา ที่ My Computers เลือก Properties เลือก Advanced system settings
- ที่แท็ป Advanced เลือก Environment Variables... ที่ System variables ให้เลือกที่ Path และ กด Edit...
- ที่ท้ายสุดของข้อความ ให้เพิ่ม path ของ python ที่เราได้ทำการติดตั้งไปเมื่อสักครู่นี้ ดังรูป



- เพียงเท่านี้ ก็สามารถใช้งาน Python ได้แล้ว
- เพิ่ม path เข้าไปอีกนิดนึง เพื่อให้สามารถใช้งาน Python ได้สะดวกขึ้น ในอนาคต
- ทำการเพิ่ม path Scripts ที่อยู่ภายใต้โฟลเดอร์ Python ที่เราสร้างไว้ ดังรูป



- ทดสอบการใช้งาน Python อีกครั้งดังข้อ 1

ทดสอบการใช้งาน Python (Mac OS)

1. เปิด terminal ขึ้นมาแล้วพิมพ์ python ลงไป จะเห็นว่าจริง ๆ แล้ว ใน Mac นั้นมี python ติดตั้งมาให้อยู่แล้ว แต่เป็น version 2.x ดังรูป

```
Last login: Mon Dec 28 21:23:28 on console  
IST-MAC-18s-Mac-mini:~ ist-mac-18$ python  
Python 2.7.6 (default, Sep  9 2014, 15:04:36)  
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 
```

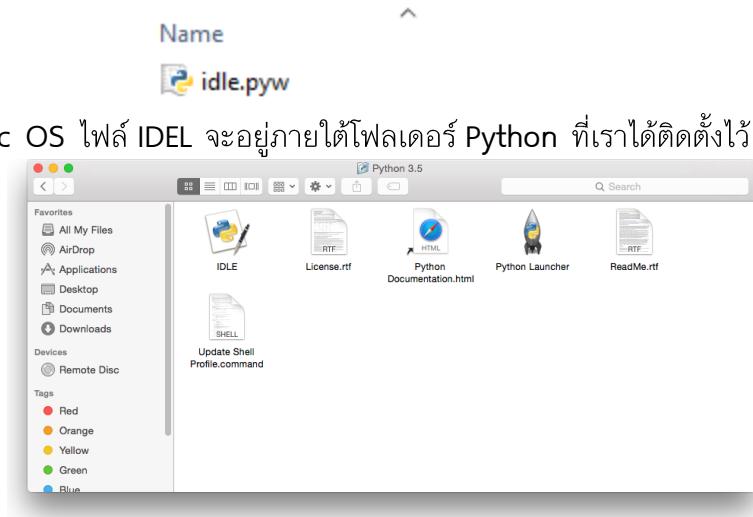
2. หากต้องการใช้งาน python version 3.x ให้พิมพ์ python3 ใน terminal ดังรูป

```
Last login: Mon Dec 28 21:38:48 on ttys000  
IST-MAC-18s-Mac-mini:~ ist-mac-18$ python3  
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  5 2015, 21:12:44)  
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 
```

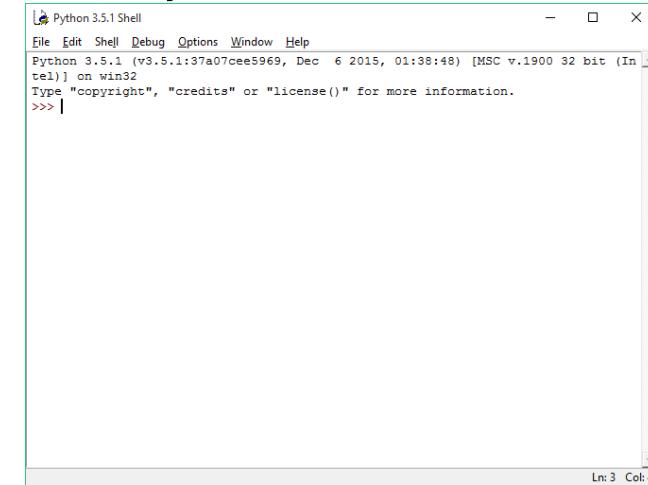
เริ่มต้นใช้งาน Python ด้วย IDLE

เมื่อเราติดตั้ง Python เรียบร้อยแล้ว เราสามารถที่จะเขียนโปรแกรมได้ทันที จะใช้ text editor หรือ โปรแกรม IDLE ที่ติดมากับ Python ก็ได้ โดยเรียก IDLE จาก start menu หรือ ตามรูปด้านล่าง ก็ได้ (windows)

New Volume (D:) > Python35 > Lib > idlelib



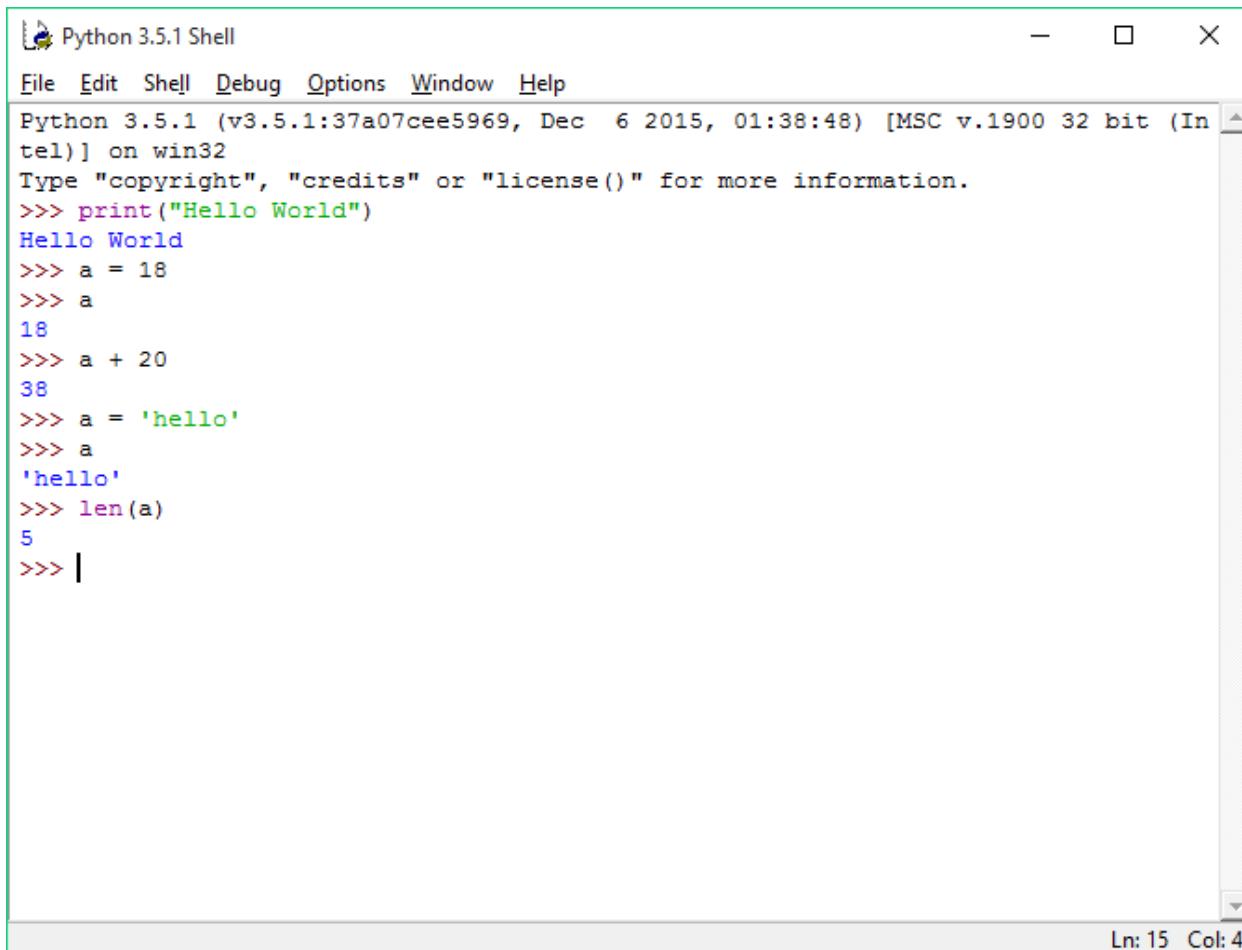
สำหรับ Mac OS ไฟล์ IDLE จะอยู่ภายใต้โฟลเดอร์ Python ที่เราได้ติดตั้งไว้



ซึ่งเราสามารถเขียน Python เล็ก ๆ น้อย ๆ ที่นี่ได้เลย แต่ถ้าต้องการโค้ดเยอะ ๆ ให้ไปสร้างไฟล์ .py และมารันที่ Python Shell จะสะดวกกว่ามาก

ลองเล่น Python Shell เบ้า

ที่ Python Shell หมายความว่าการทดลองคำสั่ง การใช้งานเบื้องต้น มากกว่าการเขียนโปรแกรมแบบจริงจัง เพราะบางครั้งอาจมีความผิดพลาดระหว่างการเขียนโค้ดก็เป็นได้ ลองพิมพ์โค้ดเล่นๆ ใน Python Shell กัน



```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:38:48) [MSC v.1900 32 bit (In tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello World")
Hello World
>>> a = 18
>>> a
18
>>> a + 20
38
>>> a = 'hello'
>>> a
'hello'
>>> len(a)
5
>>> |
```

Ln: 15 Col: 4

ลองสร้างไฟล์ เพื่อรันใน Python Shell กัน

ที่ python shell เลือก File > New File จะได้หน้าจอใหม่มา ซึ่งเราสามารถเขียนคำสั่ง Python ทั้งหมดที่นี่ได้ เมื่อพิมพ์โค้ดเสร็จเรียบร้อยแล้ว ให้ Save ไว้ที่ไหนก็ได้ และต้องบันทึกเป็นนามสกุล .py เมื่อต้องการรันไฟล์ ก็กดปุ่ม F5 ได้เลย

```
Python 3.5.1 Shell
File Edit Shell Debug Options Window Help
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:48) [MSC v.1900 32 bit (Int
el)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=====
RESTART: D:/Jobs/อบรม Basic Python/code/hello.py =====
Hello World
18
Hello
5
>>>
```

```
hello.py - D:/Jobs/อบรม Basic Python/code/hello.py (3.5.1)
File Edit Format Run Options Window Help
print("Hello World")
a = 18
print(a)
a + 20
a = 'Hello'
print(a)
print(len(a))
```

หลังจากที่ลองเล่น Python ไปได้นิดหน่อยแล้ว เราสามารถหา IDE เพื่อช่วยให้เราสามารถที่จะเขียน Python ได้สะดวกมากยิ่งขึ้นกันดีกว่า

ติดตั้งเครื่องมือช่วยในการพัฒนา Python

Python มีเครื่องมือช่วยพัฒนาจำนวนมาก (IDE) ไม่ว่าจะเป็น PyCharm, Komodo IDE, PyDev หรือ WingIDE ผู้ใช้งานสามารถเลือก IDE ที่ตัวเองชอบ และสนับได้เลย โดยในที่นี้จะเลือก PyCharm เป็น IDE ในการพัฒนา

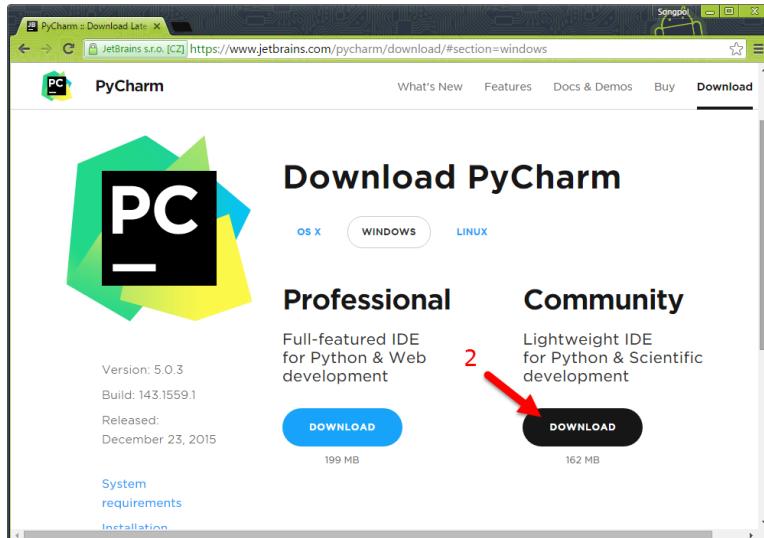
PyCharm เป็น 1 ในหลายๆ IDE ที่สามารถเขียน Python ได้ ซึ่งมีด้วยกัน 2 version คือ Professional และ Community โดย Professional นั้นมีคุณสมบัติต่าง ๆ มากกว่า Community แต่มีระยะเวลาใช้งานจำกัด หากอยากรีไซเคิลงาน ก็ต้องเสียเงินซื้อ เราเลยเลือกใช้ Community version

ติดตั้ง PyCharm (Windows)

1. เข้าเว็บ <https://www.jetbrains.com/pycharm/> ซึ่งเป็นเว็บหลักในการ download PyCharm ให้กดที่ DOWNLOAD NOW

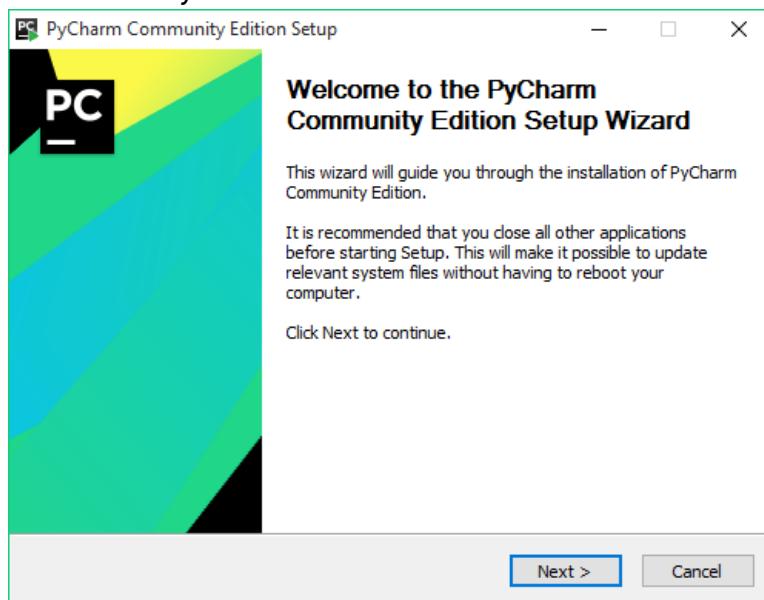


2. กด download ส่วนของ community

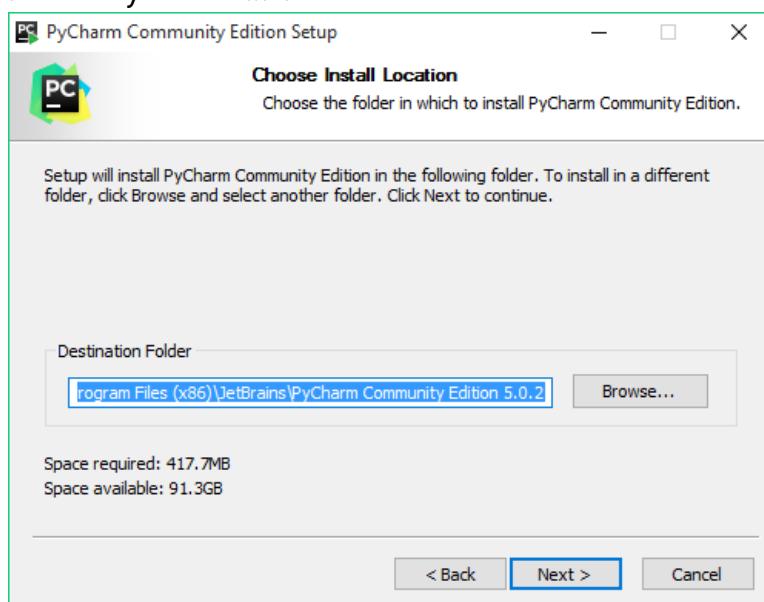


3. เมื่อกด download เว็บไซต์จะให้เราโหลดไฟล์ [pycharm-community-5.0.2.exe](#) ให้เราทำการติดตั้งได้เลย

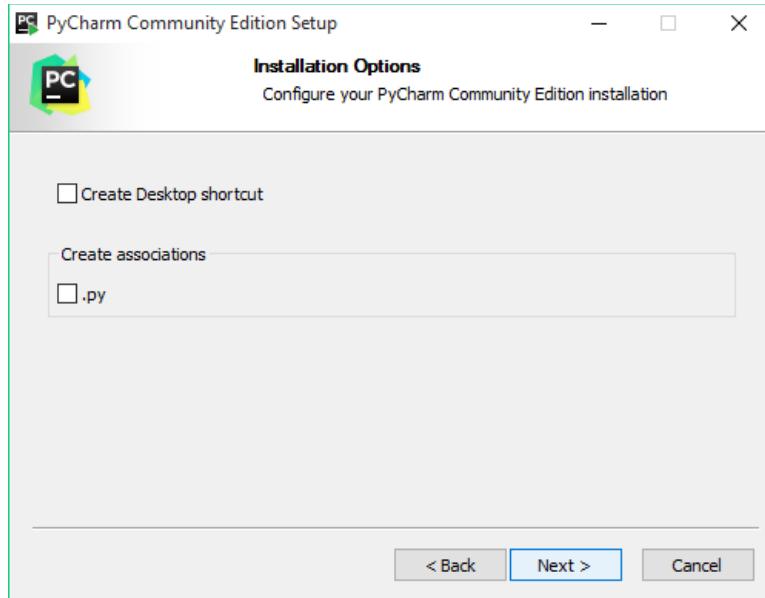
4. หน้าจอต้อนรับ การติดตั้ง PyCharm ให้กด Next



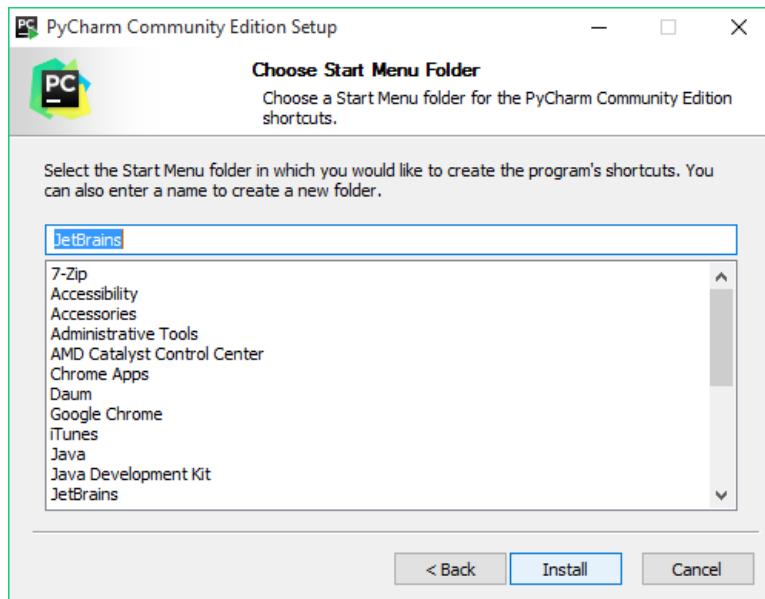
5. กำหนดสถานที่ติดตั้ง PyCharm และกด Next



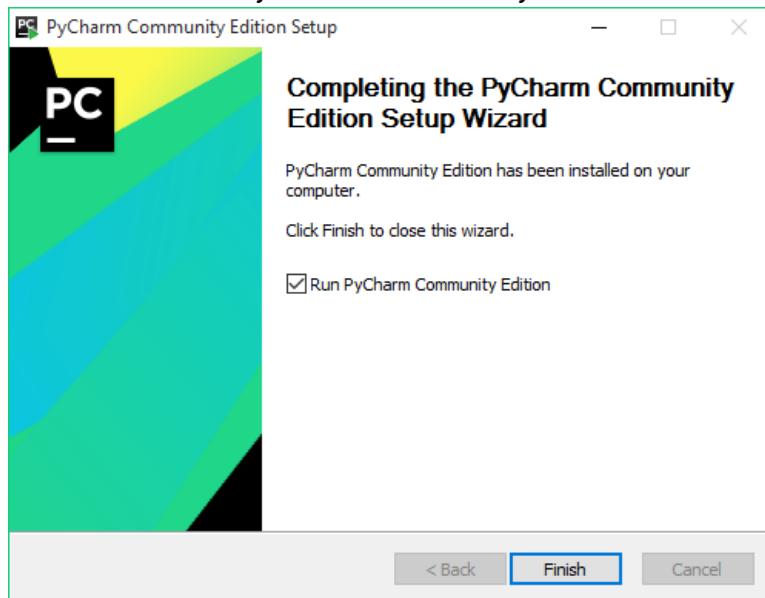
6. หากต้องการสร้าง shortcut ให้เลือกที่ Create Desktop shortcut หากต้องการให้ PyCharm เปิดไฟล์ .py ให้เลือก .py และกด Next



7. เลือก Folder ใน Start Menu และกด Install



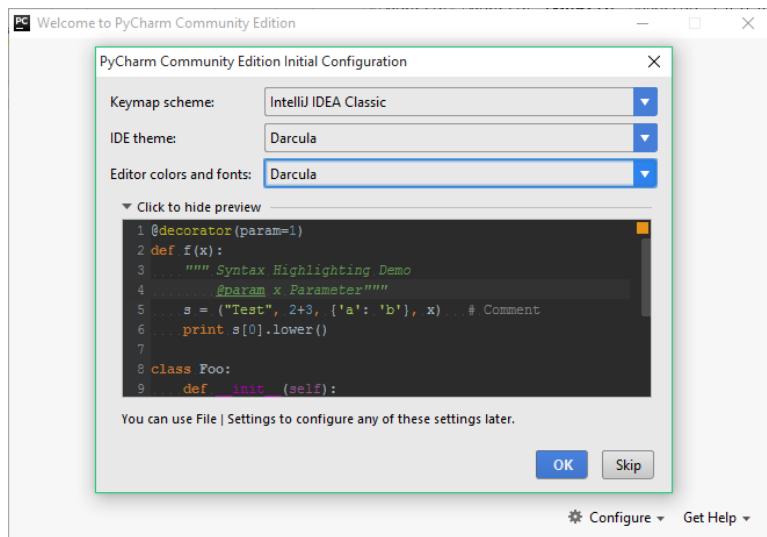
8. ติดตั้งเสร็จเรียบร้อย เลือก Run PyCharm Community Edition และกด Finish



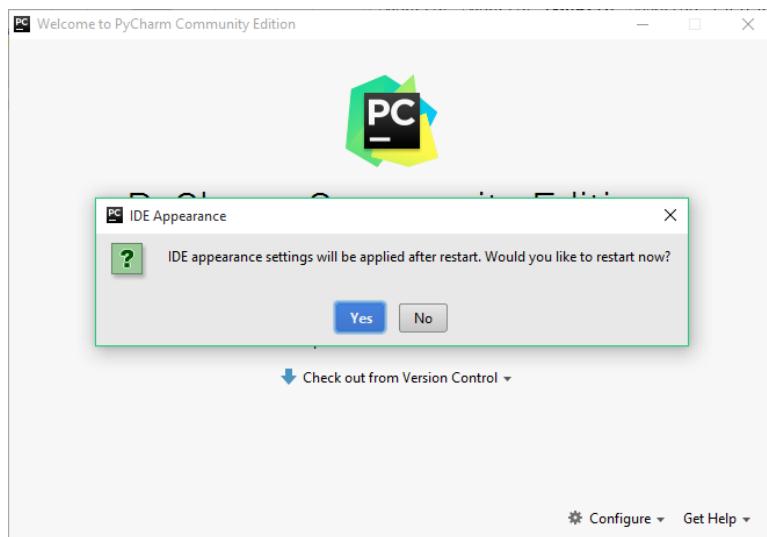
9. ทำการเปิด PyCharm



10. เข้ามาครั้งแรก PyCharm จะให้เรากำหนดค่าของ IDE ซึ่งสามารถกำหนดเองได้ภายหลัง กด OK

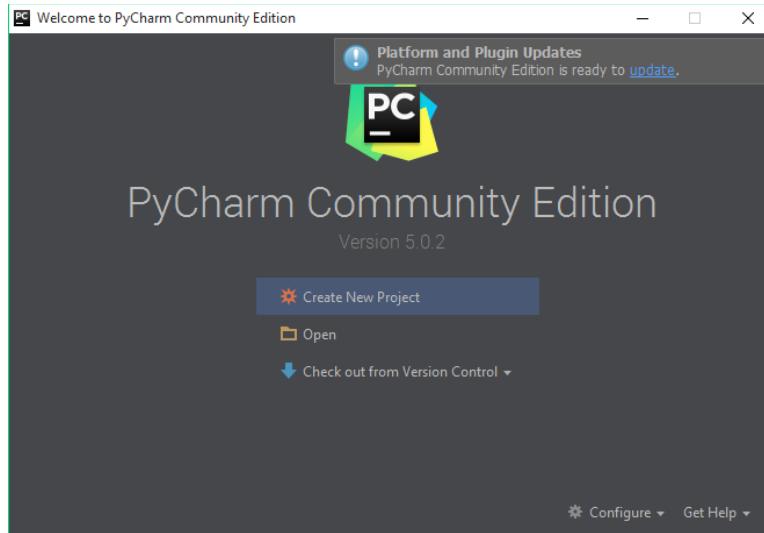


11. PyCharm จะทำการ restart ตัวเอง

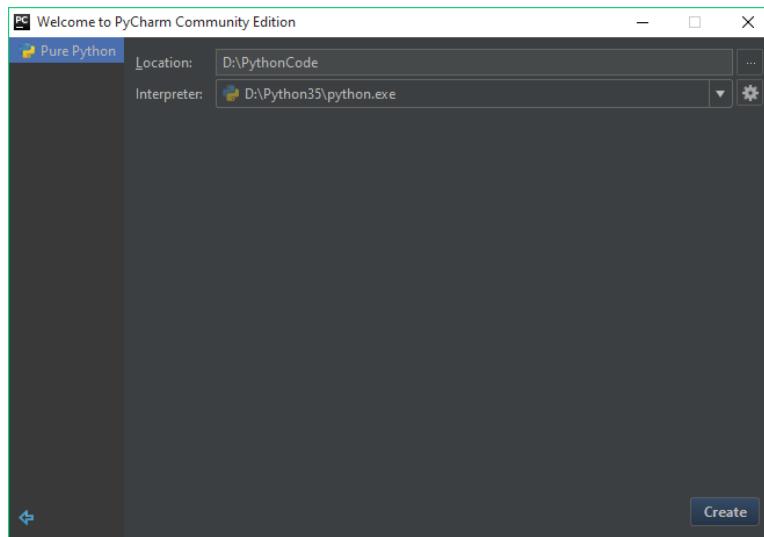


12. PyCharm จะทำการตรวจสอบว่ามี version ใหม่ หรือไม่ โดยจะแจ้งให้เราทราบที่มุมขวาบนของโปรแกรม หากต้องการ update ให้กดที่ update ได้เลย ตรงกลาง จะเป็นเมนูหน้าหลัก หากเรายังไม่

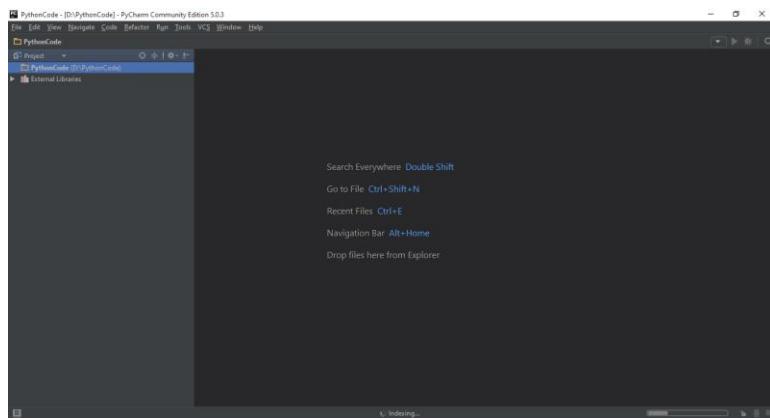
เคยสร้างโปรเจคไดๆ มาก่อน ให้เลือก Create New Project หากเราอยากรีเปิดโปรเจคเดิมที่เคยสร้างไว้ ให้กดที่ Open ในที่นี่เราจะกด Create New Project



13. Location คือสถานที่เก็บโปรเจคของเรา Interpreter คือสถานที่เก็บไฟล์ในการแปลงภาษาโปรแกรม เป็นภาษาเครื่อง ให้เลือกไปยังที่ๆ เราได้ติดตั้ง Python ไว้ จากนั้นกดปุ่ม Create (หน้าจอนี้ หากเป็น Professional Edition จะมีให้เลือกว่าโปรเจคที่เราจะสร้างนี้ จะใช้ framework ใดหรือไม่)

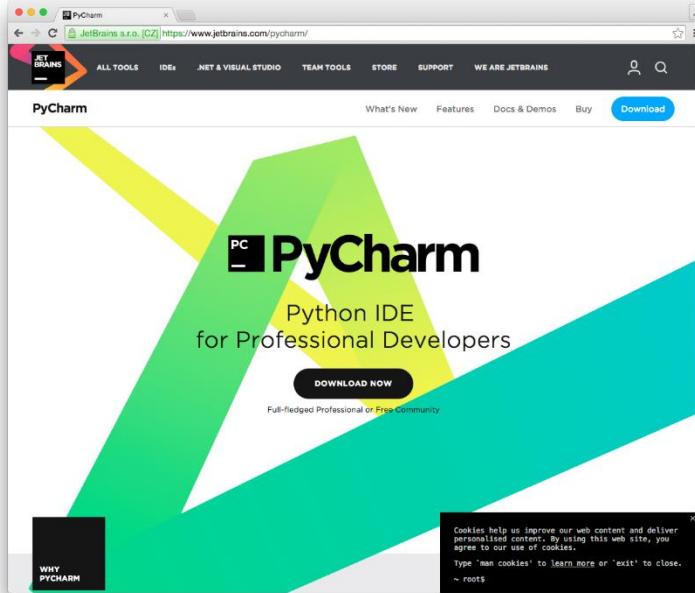


14. หน้าจอหลัก หลังจากที่สร้างโปรเจคเรียบร้อยแล้ว

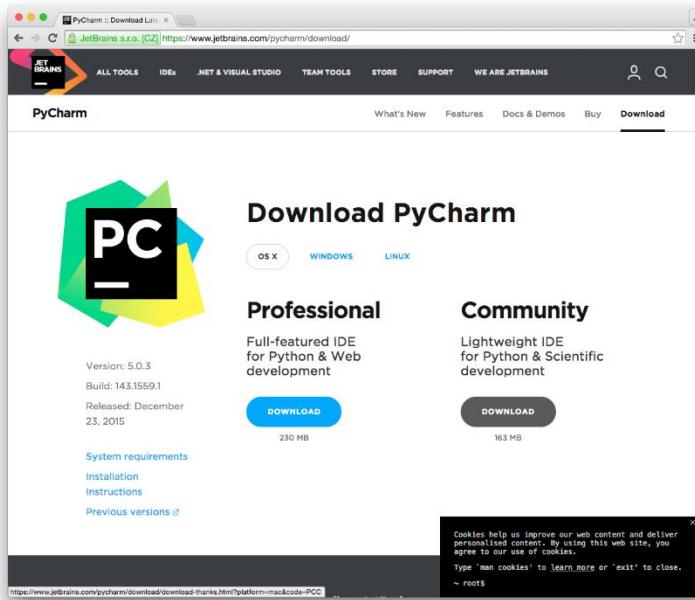


ติดตั้ง PyCharm (Mac OS)

1. เข้าเว็บ <https://www.jetbrains.com/pycharm/> ซึ่งเป็นเว็บหลักในการ download PyCharm ให้กดที่ DOWNLOAD NOW

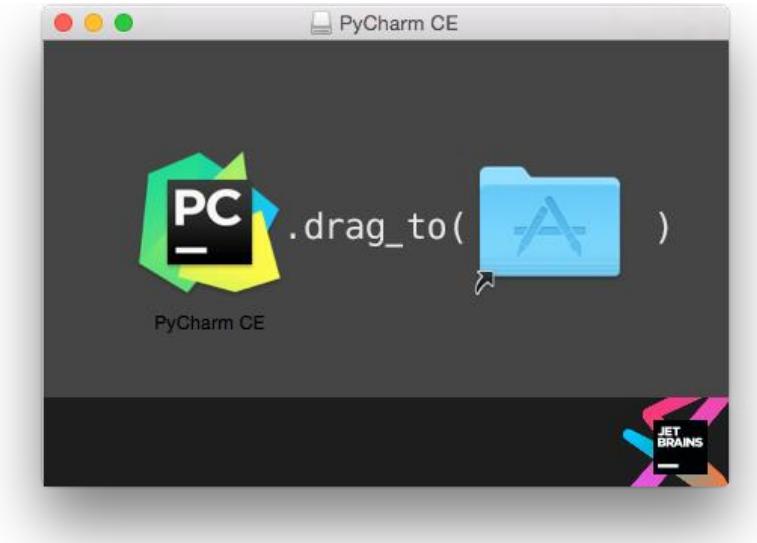


2. กด download ส่วนของ community



3. เมื่อกด download เว็บไซต์จะให้เราโหลดไฟล์ [pycharm-community-5.0.2-jdk-bundled.dmg](https://www.jetbrains.com/pycharm/download/thanks.html?platform=mac&code=PCC) ให้เราทำการติดตั้งได้เลย

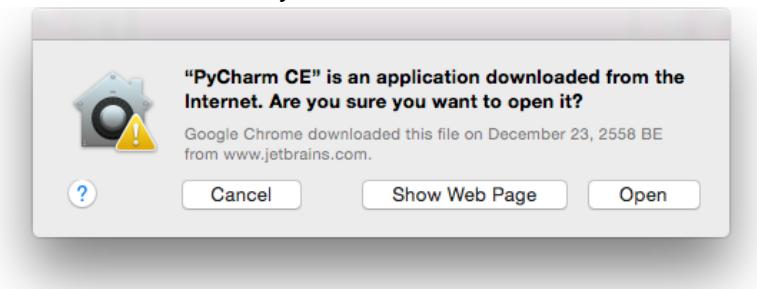
4. หน้าจอติดตั้ง PyCharm ให้ลาก icon จากซ้ายมือ ไปขวามือ



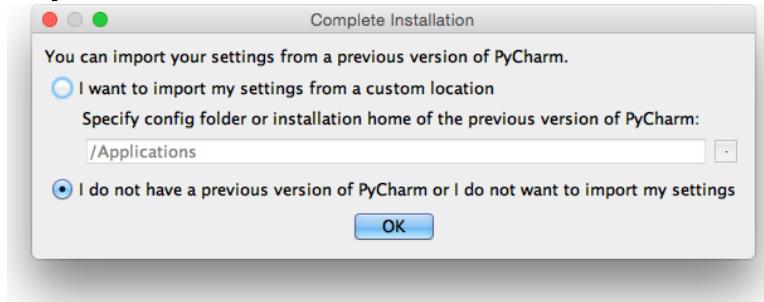
5. เป็นอันเสร็จสิ้นการติดตั้ง PyCharm ทำการเปิด PyCharm



6. ให้กดปุ่ม Open เพื่อเริ่มต้นใช้งาน PyCharm



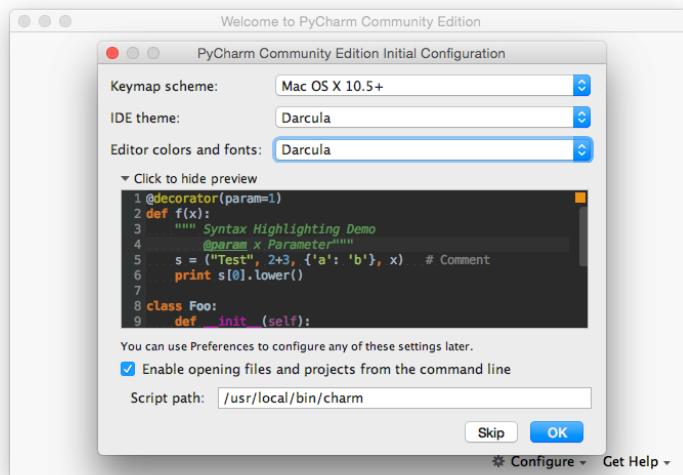
7. PyCharm สอบถามว่าเราต้องการดึงค่าต่างๆ จาก version ก่อนหน้าหรือไม่ หากไม่มี version ก่อนหน้า ให้เลือกดังรูป และกด OK



8. หน้าจอเริ่มต้น



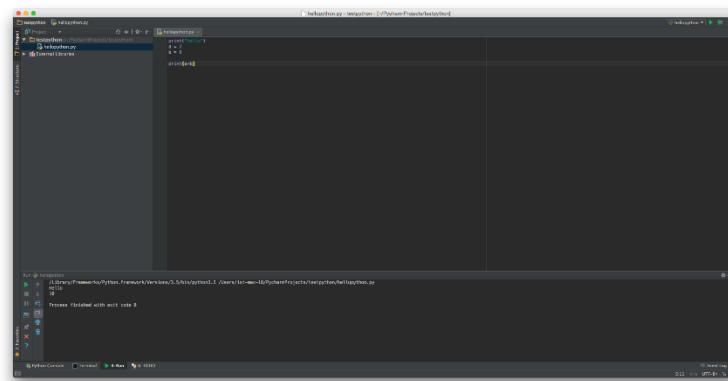
9. เข้ามาครั้งแรก PyCharm จะให้เรากำหนดค่าของ IDE ซึ่งสามารถกำหนดเองได้ภายหลัง กด OK



10. PyCharm จะทำการตรวจสอบว่ามี version ใหม่ หรือไม่ โดยจะแจ้งให้เราทราบที่มุมขวาบนของโปรแกรม หากต้องการ update ให้กดที่ update ได้เลย ตรงกลาง จะเป็นเมนูหน้าหลัก หากเรายังไม่เคยสร้างโปรเจคใดๆ มาก่อน ให้เลือก Create New Project หากเราอยากรีดโปรเจคเดิมที่เคยสร้างไว้ ให้กดที่ Open ในที่นี่เราจะกด Create New Project



11. Location คือสถานที่เก็บโปรเจคของเรา Interpreter คือสถานที่เก็บไฟล์ในการแปลงภาษาโปรแกรม เป็นภาษาเครื่อง ให้เลือกไปยังที่ๆ เราได้ติดตั้ง Python ไว้ จากนั้นกดปุ่ม Create (หน้าจอนี้ หากเป็น Professional Edition จะมีให้เลือกว่าโปรเจคที่เราจะสร้างนี้ จะใช้ framework ใดหรือไม่)
12. หน้าจอหลัก หลังจากที่สร้างโปรเจคเรียบร้อยแล้ว



General Syntax

ในส่วนนี้ เราจะมาวิจารณ์รูปแบบการเขียน Python กันสักนิดหน่อย เนื่องจากภาษา Python มีรูปแบบการเขียนที่ง่าย กระชับ และแตกต่างจากการภาษาโปรแกรมอื่น ๆ ที่คุ้นเคยกันมา

Creating a main script

ทุกครั้งที่เราทำการรันโปรแกรม เราจะมีตัวแปรตัวหนึ่งที่ชื่อว่า `__name__` เพื่อบอกว่า การรันไฟล์ .py ตัวนี้ เป็นการรันแบบไหน เป็นการรันจากการเรียกใช้โดยไฟล์อื่น หรือรันเป็นตัวหลัก

หากรันไฟล์ .py นี้ เป็นตัวหลัก ค่าใน `__name__` จะมีค่าเท่ากับ `__main__` ซึ่งทำให้เราสามารถตรวจสอบได้ว่า หากเป็น `__main__` จริง จะให้มีการทำงานแบบใดต่อไป

t01.py

```
1  def first():
2      print('Hello World')
3
4  if __name__ == '__main__':
5      first()
6
```

จาก t01.py บรรทัดที่ 4 มีการตรวจสอบตัวแปร `__name__` ว่า มีค่าเท่ากับ `__main__` จริง หรือไม่ หากจริง จะทำการเรียกใช้ `function first()` ที่สร้างไว้ในบรรทัดที่ 1 ซึ่งจะแสดงข้อความว่า Hello World ออกมานะ

```
D:\Python35\python.exe "D:/Jobs/อบรม Basic Python/code/training/t01.py"
Hello World
```

Understanding whitespace in Python

หากเคยเขียนโปรแกรมภาษาอื่น ๆ มา ก่อน ผสมเข้าว่าทุกท่าน คงมีปัญหาเกี่ยวกับการรูปแบบการเขียน โดยที่จะเป็นไปอย่างไร บางคนอาจจะย่อหน้าคำลั่งบ้าง บางคนไม่ย่อหน้า ซึ่งแต่ละคนก็มีวิธีการเขียนที่แตกต่างกันไป และบางครั้งก็สับสนว่า ปึกนี้ เป็นปึกปิดของใคร

Python จึงใช้วิธีการย่อหน้า แทนการใช้ {} ในการแสดง scope การทำงานต่าง ๆ ของ for, if, function และอื่น ๆ ซึ่งการใช้การย่อหน้านี้ อาจจะไม่คุ้นเคยไปสักนิดหน่อย แต่ก็ทำให้การเขียนโค้ดนั้น สวยงามมากขึ้น และช่วยให้การแก้ไขปัญหา ในกรณีที่เกิดการผิดพลาดทำได้ง่ายมากขึ้นด้วยเช่นเดียวกัน

Python ไม่แนะนำให้ใช้การกด tab เพราะในแต่ละ platform นั้นใช้จำนวนช่องว่างในการ tab ไม่เท่ากัน จึงแนะนำให้ใช้การกด space แทน หรือกำหนด editor ของเราให้ซองว่างแทน tab ซึ่ง Python แนะนำว่า 4 ช่องว่าง เท่ากับ 1 tab (แต่ Google ใช้แค่ 2 ช่องว่าง)

นอกจากนี้ Python ยังแนะนำให้เราใช้ บรรทัดว่าง ๆ ในการแบ่งแยกระหว่าง function, class, if หรืออื่น ๆ เพื่อให้ง่ายต่อการสังเกตเรื่องของขอบเขตการทำงาน

และอีกหนึ่งคำแนะนำจาก Python ก็คือ การใช้ช่องว่างระหว่าง operator ต่าง ๆ

Commenting code

ทุก ๆ ภาษาจำเป็นต้องมีการ comment เพื่อใช้จด บันทึก หรือเขียนอธิบายการทำงานต่าง ๆ ไว้สำหรับ Python ก็มี comment ด้วยเช่นกัน

Python ใช้ # แสดงการ comment ซึ่งเป็นการ comment ทั้งบรรทัด ไม่มีการ comment เป็นช่วง หรือมากกว่า 1 บรรทัด

PyCharm ใช้ Ctrl + / เพื่อทำการ comment และ uncomment

```
1  def first():
2      print('Hello World')
3
4  if __name__ == '__main__':
5      first()
6      # print('Python is Good')
7      # print('Python is Great')
```

Creating and using functions

หากเรามีโปรแกรมส่วนหนึ่งส่วนใด ที่จะถูกเรียกใช้บ่อยครั้ง เราสามารถสร้างเป็น function ขึ้นมาใช้งานได้ ซึ่งประโยชน์นี้ในการสร้าง function ก็คือเรียกใช้งานได้บ่อยครั้ง และหากมีการแก้ไข ก็แก้ไขที่จุดเดียว ไม่ต้องแก้ไขในหลาย ๆ จุด

การสร้าง function จะให้คำลั่ง def ตามด้วยชื่อ function ที่ต้องการ ปิดท้ายด้วยเครื่องหมาย : คำลั่งที่อยู่ใน function นี้ ต้องย่อหน้าเข้ามาด้วย

Function จะมีการรับค่า หรือไม่ก็ได้ ถ้ามีการรับค่า จะรับกี่ค่าก็ได้ ทั้งนี้ เราสามารถกำหนดค่า default ให้ได้ด้วย

```

1  def first():
2      second(2)
3      third(2, 8)
4
5
6  def second(a):
7      for i in range(a, 10):
8          print(i, end=' ')
9      print()
10
11
12 def third(a, b=10):
13     for i in range(a, b):
14         print(i, end=' ')
15     print()
16
17
18 if __name__ == '__main__':
19     first()

```

Creating and using objects

Python สนับสนุนการทำงานแบบ object และใน version 3 นี้ ทุก ๆ อย่างใน python ก็เป็น object ทั้งหมด

คลาส เปรียบเสมือนพิมพ์เขียว ที่บอกว่า โครงสร้างมีอะไรบ้าง มีการเก็บข้อมูลใดบ้าง เราสามารถสร้าง object จากพิมพ์เขียนนี้ได้ ซึ่งจะมีคุณสมบัติเหมือนคลาสทุกประการ

ในการสร้างคลาส เราใช้คำสั่ง `class` ตามด้วยชื่อคลาสที่ต้องการ ชื่อคลาส โดยปกติเราตั้งเป็นลักษณะ `CamelCase` คือ ตัวอักษรตัวแรกของคำเป็นตัวอักษรตัวใหญ่ จนนั้นปิดท้ายด้วย :

ถึงแรกที่ทำหลังจากสร้างคลาส คือการสร้าง `constructor` ซึ่งใน Python จะใช้ชื่อ `method` ว่า `__init__` พารามิเตอร์ที่ต้องมีในทุก ๆ `method` ในคลาส ก็คือ `self` โดย `self` ใช้สำหรับอ้างถึงตัว `object` ที่ใช้งาน ณ เวลาปัจจุบัน จากนั้นตามด้วยพารามิเตอร์ตัวอื่น ๆ ที่คลาสจำเป็นต้องมี ทั้งนี้ พารามิเตอร์นี้อาจมีหรือไม่มีค่า `default` ก็ได้

นอกจาก `constructor` แล้ว เราสามารถสร้าง `method` อื่น ๆ ได้อีก ตามที่คลาสควรจะมี

เมื่อสร้างคลาสเสร็จแล้ว เราต้องเรียกใช้งาน โดยทุกครั้งที่มีการสร้าง `object` ขึ้นมา ก็จะมีการเรียกใช้งาน `constructor` โดยอัตโนมัติ

ทั้งนี้ เราจะมาลงรายละเอียดเกี่ยวกับคลาสอีกครั้ง

```

1  class Movie:
2      def __init__(self, category='action'):
3          self.category = category
4
5      def what_category(self):
6          return self.category
7
8
9  def first():
10     iron = Movie()
11     harry = Movie('fantasy')
12
13     print(iron.what_category())
14     print(harry.what_category())
15
16
17 if __name__ == '__main__':
18     first()

```

การรับค่าจาก keyboard

ความสามารถรับค่าจาก keyboard ได้ด้วย function input('...') โดยค่าที่ได้มาจะเป็น string

```

1  def main():
2      name = input('Please enter your name: ')
3
4      print('Hello, ' + name + '.')
5      print('Hello, ' + name + '.')
6
7  if __name__ == '__main__':
8      main()

```

ผลลัพธ์

```

Please enter your name: Gong
Hello, Gong .
Hello, Gong.

```

Using numbers

ใน Python นั้น มีชนิดตัวแปรที่เป็นตัวเลขอยู่ 2 ชนิดคือ int และ float

t05.py

```
1  def main():
2      num = 18
3      print(type(num), num)  # <class 'int'> 18
4
5      num = 18.0  # <class 'float'> 18.0
6
7      num = 18 / 7  # <class 'float'> 2.5714285714285716
8
9      num = 18 // 7  # <class 'int'> 2
10
11     num = round(18 / 7)  # <class 'int'> 3
12
13     num = round(18 / 7, 5)  # <class 'float'> 2.57143
14
15     num = 18 % 7  # <class 'int'> 4
16
17     num = int(18.18)  # <class 'int'> 18
18
19     num = float(18)  # <class 'float'> 18.0
20
21 if __name__ == "__main__":
22     main()
```

บรรทัดที่ 2 กำหนดค่าตัวแปร num ให้มีการอ้างอิงไปยัง object int ที่มีค่า เท่ากับ 18

บรรทัดที่ 5 ให้ตัวแปร num อ้างอิงไปยัง 18.0

บรรทัดที่ 7 คือการหาร

บรรทัดที่ 9 คือการหารเอาส่วน โดยใช้ //

บรรทัดที่ 11 การใช้ function round ในการปัดค่าขึ้น ผลลัพธ์ได้เป็น int

บรรทัดที่ 13 การใช้ function round โดยมีการกำหนดจำนวนจุดทศนิยม ผลลัพธ์ที่ได้เป็น float

บรรทัดที่ 15 คือการหารเอาเศษ โดยใช้ % หรือที่เรียกวันว่า การ modulo หรือการ mod

บรรทัดที่ 17 คือการสร้าง object int จากค่าตัวเลขที่เป็น float

บรรทัดที่ 19 คือการสร้าง object float จากค่าตัวเลขที่เป็น int

Using strings

ข้อความใน Python มีความยืดหยุ่นมาก เราสามารถใช้ลักษณะในการแทนข้อความได้มากกว่า 1 แบบ ดังนี้ ใช้ single quote, double quote หรือ triple quote ก็ได้

เราสามารถใช้สัญลักษณ์พิเศษ เช่น `\n` เพื่อขึ้นบรรทัดใหม่ หรือ `\t` เพื่อ tab กับ string ได้ หากเราใส่ `r` ไว้ข้างหน้าสุดของ string จะทำให้สัญลักษณ์ต่าง ๆ หมดความหมายไป เพราะ `r` หมายถึง raw string

การแทรกตัวแปรต่าง ๆ เข้าไปใน string นั้น มีด้วยกันหลายวิธี ไม่ว่าจะเป็น การใช้ + การใช้ , หรือ การใช้ `.format()` (เราจะกล่าวถึง `.format()` ภายหลัง)

ผลลัพธ์ที่ได้จากการใช้ triple quote นั้น จะมีลักษณะการจัดรูปแบบเหมือนกับตอนที่เราเขียนโค้ด ทั้งนี้ การใช้ triple quote นี้ ถูกใช้อย่างมากในการเขียน document string เพื่ออธิบายการทำงานของ function ต่าง ๆ ของ python (จะกล่าวถึงอีกทีภายหลัง)

```
1  def main():
2      s = 'Hello world' # Hello world
3
4      s = "Hello world" # Hello world
5
6      s = """Hello
7          world"""
8      # Hello
9      #     world
10
11     s = '''Hello
12         world'''
13     # Hello
14     #     world
15
16     s = 'Hello \nworld'
17     # Hello
18     # world
19
20     s = r'Hello \n world' # Hello \n world
21
22     print(s)
23
24     a = 18
25
26     print("I'm", a, "years old") # I'm 18 years old
27
28     print("I'm " + str(a) + " years old") # I'm 18 years old
29
30
31 if __name__ == "__main__":
32     main()
```

Conditionals

คำสั่ง if คือคำสั่งในการเลือกปฏิบัติคำสั่งที่เราได้กำหนดไว้ โดยที่ if จะมีเงื่อนไข และ if นั้นจะทำงาน ก็ต่อเมื่อ เงื่อนไขในคำสั่งนั้น เป็นจริง

Selecting code with if and else conditional statements

การทำงาน if...else เป็นการทำงานแบบ 2 ทางเลือก โดยจะทำการตรวจสอบเงื่อนไขใน if ก่อน หาก เป็นจริง ก็จะทำงานคำสั่งที่อยู่ภายใต้ if แต่หากไม่เป็นจริง จะทำงานคำสั่งที่อยู่ภายใต้ else

รูปแบบการใช้งาน if

```
if <condition>:  
    statements  
else:  
    statements
```

```
1  def main():  
2      height = 180  
3  
4      if height < 160:  
5          print('short')  
6      else:  
7          print('tall')  
8  
9      a = 'short' if height < 160 else 'tall'  
10  
11     print(a)  
12  
13     if __name__ == "__main__":  
14         main()
```

Setting multiple choices with elif

หากการทำงานของเรามีทางเลือกมากกว่า 2 ทางเลือก เราสามารถใช้คำสั่ง if...elif...else ได้ ทั้งนี้ใน ภาษา Python ไม่มีการทำงานแบบ switch case เหมือนกับภาษาอื่น ๆ

សម្រាប់ការវិភាគ if...elif...else

If <condition 1>:

 statements

elif <condition 2>:

 statements

...

else:

 statements

```
1  def main():
2      height = 175
3
4      if height < 160:
5          print('short')
6      elif height >= 160 and height < 180:
7          # elif 160 <= height < 180:
8          print('normal')
9      else:
10         print('tall')
11
12     if __name__ == "__main__":
13         main()
```

Loops

การวนรอบการทำงาน ช่วยให้การเขียนโปรแกรมยืดหยุ่นมากยิ่งขึ้น ซึ่ง Python เตรียมการวนรอบ การทำงานให้แล้ว คือ while และ for

Creating loops with while

ในการทำงานที่บางครั้งเราไม่ทราบจำนวนรอบที่แน่ชัด เราจะใช้คำสั่ง while เข้ามาช่วยการทำงาน การใช้งาน while นี้ จำเป็นที่ต้องมีการตรวจสอบเงื่อนไขก่อนเสมอ ซึ่งเงื่อนไขต้องเป็นจริงเท่านั้น จึงจะวนรอบการทำงาน

รูปแบบการใช้งาน while

```
while <condition>:
```

```
    statements
```

```
1  def main():
2      i = 0
3
4      while i <= 18:
5          print(i)
6          i += 1
7
8
9  if __name__ == "__main__":
10     main()
```

Iterating with for

For คือคำสั่งในการวนรอบเช่นเดียวกับ while แต่เป็นการวนรอบที่เราทราบจำนวนແນซัดว่า ต้องวนรอบทั้งหมดกี่รอบ

รูปแบบการใช้งาน for

```
for iterating_var in sequence:  
    statements
```

```
1  def main():  
2      i = 5  
3  
4      for i in range(8):  
5          print(i)  
6  
7      print()  
8  
9      for i in range(2, 8, 3):  
10         print(i)  
11  
12     s = 'Hello world'  
13     print()  
14  
15     for i in s:  
16         print(i, end=' ')  
17  
18     print()  
19     l = ['a', 'b', 'c']  
20     for i in l:  
21         print(i)  
22  
23  
24     if __name__ == "__main__":  
25         main()
```

range() function

range() เป็น function ที่สร้างตัวเลขขึ้นมา 1 ชุด ที่มีค่าเริ่มต้นกันไปเรื่อยๆ จากตัวอย่างก่อนหน้า เราใช้คำสั่ง range(8) ผลลัพธ์ที่ได้จะเป็น 0 1 2 3 4 5 6 7

range(start, stop, step) เป็นรูปแบบการกำหนดค่าของ range โดยสามารถกำหนดค่าเริ่มต้น ค่าสิ้นสุด และค่าที่ใช้เพิ่มในแต่ละรอบการทำงาน เช่น range(2, 8, 3) คือกำหนดให้เริ่มต้นที่ 2 เพิ่มค่าไปทีละ 3 และจบที่ค่า 8

Enumerating iterators

enumerate() function

enumerate() เป็น function ที่ใช้ร่วมกับข้อมูลที่สามารถวนค่าภายในได้ เช่น ข้อมูลที่เป็น string หรือ list เป็นต้น โดย function นี้จะให้ค่าคืนกลับมา 2 ค่า เป็น tuple ประกอบด้วย index และ value ของข้อมูล

```
1 def main():
2     s = 'Hello world'
3     print()
4
5     for index, value in enumerate(s):
6         print(index, value)
7
8     print()
9     l = ['a', 'b', 'c']
10    for i, v in enumerate(l):
11        print(i, v)
12
13
14 if __name__ == "__main__":
15     main()
```

Controlling loop flow with break, continue, and else

ในการเขียนโปรแกรม บ้างครั้ง เราต้องการที่จะขัดจังหวะการทำงานของ่วนรอบ หรือต้องการให้หลุดจากการวนรอบ เราสามารถใช้ break และ continue ได้

break

การใช้งาน break เพื่อหลุดรอบการทำงานจากการทำงานในสุด ของ for หรือ while
continue

เป็นคำสั่งในการข้ามรอบการทำงานปั๊บๆ

else

else เป็นคำสั่งที่จะทำงานก็ต่อเมื่อ หลุดจาก loop ด้วยวิธีปกติ ไม่ได้หลุดด้วยการ break
pass

python ไม่อนุญาตให้เราสร้าง function ว่าง ๆ ทิ้งไว้ หรือ เขียน if โดยไม่มี statement ใด ๆ ซึ่งหากเรายังไม่รู้ว่า ณ เวลานั้น จะเขียนคำสั่งอะไร เราสามารถใช้คำสั่ง pass ได้

`pass` เป็นคำสั่งที่ไม่ว่าเพื่อให้ไม่ทำการใด ๆ เช่น เรายุ่ง เราต้องมีเงื่อนไขนี้ แต่ตอนนี้ยังไม่รู้ว่าจะทำอะไร ก็ให้ใส่ `pass` ไว้ก่อน แล้วเมื่อรู้ว่าต้องเขียนอะไร ก็ค่อยมาแก้ไขอีกที แล้วให้ลบ `pass` ออกไป

```

1  def main():
2      for i in range(10):
3          if i == 8:
4              break
5          # continue
6          # pass
7          print(i, end=' ')
8      else:
9          print('end loop')
10
11
12 if __name__ == "__main__":
13     main()

```

Operators

หัวข้อนี้เราจะดูเกี่ยวกับตัวกระทำ (operators) ที่มีใน Python

Performing simple arithmetic

ตัวกระทำทางคณิตศาสตร์ที่ python เตรียมไว้ให้มีดังนี้ (สมมติให้ `num = 8`)

Operator	คำอธิบาย	ตัวอย่าง	ผลลัพธ์
+	บวก	$5 + 3$	8
-	ลบ	$5 - 3$	2
*	คูณ	$5 * 3$	15
/	หาร (ผลลัพธ์เป็น float)	$5 / 3$	1.6666666666666667
//	หารเอาส่วน (ผลลัพธ์เป็น int)	$5 // 3$	1
%	หารเอาเศษ (ผลลัพธ์เป็น int)	$5 \% 3$	2
divmod()	Function เพื่อหา ผลลัพธ์ที่ได้จากคำสั่ง // และ % (ผลลัพธ์เป็น tuple)	<code>divmod(5, 3)</code>	(1, 2)

Operator	คำอธิบาย	ตัวอย่าง	ผลลัพธ์
<code>+=</code>	บวกค่าให้กับตัวแปร	<code>num += 2</code>	10
<code>-=</code>	ลบค่าให้กับตัวแปร	<code>num -= 2</code>	6
<code>*=</code>	คูณค่าให้กับตัวแปร	<code>num *= 2</code>	16
<code>/=</code>	หารค่าให้กับตัวแปร	<code>num /= 2</code>	4
<code>//=</code>	หารเอาส่วน	<code>num //= 2</code>	4
<code>**</code>	ยกกำลัง	<code>2 ** 3</code>	8

Comparing values

Operator ที่ใช้สำหรับเปรียบเทียบค่ามีดังนี้

Operator	คำอธิบาย	ตัวอย่าง	ผลลัพธ์
<code><</code>	น้อยกว่า	<code>8 < 18</code>	True
<code>></code>	มากกว่า	<code>8 > 18</code>	False
<code><=</code>	น้อยกว่า หรือเท่ากับ	<code>8 <= 18</code>	True
		<code>8 <= 8</code>	True
		<code>10 <= 8</code>	False
<code>>=</code>	มากกว่า หรือเท่ากับ	<code>8 >= 18</code>	False
		<code>10 >= 8</code>	True
<code>==</code>	เท่ากัน	<code>8 == 8</code>	True
		<code>8 == 18</code>	False
<code>!=</code>	ไม่เท่ากัน	<code>8 != 8</code>	True
		<code>8 != 8</code>	False

Membership operation

Operator ที่ใช้สำหรับตัวแปรที่มีค่าเรียงต่อกัน เช่น string, lists หรือ tuples

สมมติ `a = 8` และ `b = [1, 3, 5, 7, 18]`

Operator	คำอธิบาย	ตัวอย่าง	ผลลัพธ์
<code>in</code>	ตรวจสอบว่ามีค่าอยู่ในอีกตัวแปรจริงหรือไม่	<code>a in b</code>	False
<code>not in</code>	ตรวจสอบว่าไม่มีค่าอยู่ในอีกตัวแปรจริงหรือไม่	<code>a not in b</code>	True

Operating on Boolean values

Python มีการเขียน Boolean เช่นเดียวกับภาษาอื่น โดยค่าที่ได้จะเป็น True และ False จากคลาส bool

โดยส่วนมาก เราใช้ Boolean ใน if หรือ while หากเงื่อนไขของเรามีมากกว่า 1 เงื่อนไข เราสามารถใช้ and และ or ในการเชื่อมแต่ละเงื่อนไขเข้าด้วยกันได้

หากเราใช้ and เป็นตัวเชื่อมเงื่อนไขเข้าด้วยกัน ผลลัพธ์จะเป็นจริง ก็ต่อเมื่อ เงื่อนไขทั้งหมดเป็นจริง

หากเราใช้ or เป็นตัวเชื่อมเงื่อนไขเข้าด้วยกัน ผลลัพธ์จะเป็นเท็จ ก็ต่อเมื่อ เงื่อนไขทั้งหมดเป็นเท็จ

Operating on parts of a container with the slice operator

Slice คือตัด การหั่นข้อมูลออกเป็นท่อน ๆ โดยใช้ได้กับตัวแปร list หรือ string เป็นต้น

```
1 def main():
2     a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
3     print(a) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
4     print(a[2]) # 2
5     print(a[0:5]) # [0, 1, 2, 3, 4]
6     print(a[:]) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
7     print(a[:5]) # [0, 1, 2, 3, 4]
8     print(a[2:]) # [2, 3, 4, 5, 6, 7, 8, 9]
9     print(a[2:9:3]) # [2, 5, 8]
10    # [start:stop:step]
11
12 if __name__ == "__main__":
13     main()
```

บรรทัดที่	คำอธิบาย
2	ประกาศตัวแปร a เป็นชนิด list
3	ทดสอบ print ตัวแปร a
4	สามารถอ้างถึงข้อมูลต่าง ๆ ด้วย index ได้ โดย index เริ่มต้นที่ 0
5	[0:5] หมายถึง ให้ทำการ slice ข้อมูลตั้งแต่ ตำแหน่งที่ 0 จนถึงตำแหน่งที่ 4
6	[] หมายถึง ให้แสดงข้อมูลทั้งหมด (ไม่มีการ slice)
7	[5] หมายถึง ให้ slice ตั้งแต่ตำแหน่งที่ 0 จนถึงตำแหน่งที่ 4
8	[2:] หมายถึง ให้ slice ตั้งแต่ตำแหน่งที่ 2 จนถึงตำแหน่งสุดท้าย
9	[2:9:3] หมายถึง ให้ slice ตั้งแต่ตำแหน่งที่ 2 จนถึงตำแหน่งที่ 8 โดยข้อมูลถัดไปที่ต้องการคือ ตำแหน่งที่ 3

Aggregating values with lists and tuples

Tuple

Tuple เป็นตัวแปรชนิดหนึ่ง สามารถเก็บข้อมูลได้จำนวนมาก เข้าถึงข้อมูลโดยการระบุเลข index (คล้ายกับ array) แต่สิ่งที่ tuple แตกต่างจาก array ก็คือ tuple จะเก็บข้อมูลแบบคงที่ ไม่สามารถเปลี่ยนแปลงข้อมูลภายใน tuple ได้ ไม่สามารถเพิ่มลบ เรียงข้อมูลที่อยู่ใน tuple ได้โดยตรง แต่ข้อดีคือ ทำให้การเข้าถึงข้อมูลทำได้เร็วกว่า

การประกาศตัวแปร tuple

ใช้เครื่องหมาย วงเล็บ (ข้อมูล1, ข้อมูล2, ...) หรือการใส่เครื่องหมาย comma คั่นข้อมูล โดยไม่มีการใส่วงเล็บ

List

List เป็นตัวแปรอีกชนิดหนึ่ง สามารถเก็บข้อมูลได้มากกว่า 1 ค่า เช่นเดียวกับ tuple เราสามารถอ้างถึงข้อมูลต่าง ๆ ได้ โดยใช้ index เช่นเดียวกัน แต่สิ่งที่แตกต่างจาก tuple ก็คือ list สามารถที่จะเพิ่มข้อมูลเข้าไปได้เรื่อย ๆ แทรกข้อมูล เรียงข้อมูลได้

การประกาศตัวแปร list

ใช้เครื่องหมาย brackets [ข้อมูล1, ข้อมูล2,...]

method ที่สามารถใช้กับ list ได้ เช่น

list.append(x)

เป็นการเพิ่ม x ที่ท้ายของ list

list.extend(L)

เป็นการเพิ่ม L ซึ่งเป็น list อีกตัวหนึ่ง มาต่อท้ายจาก list เดิมที่มีอยู่

list.insert(i, x)

เป็นการเพิ่ม x ไปยังตำแหน่ง i ที่ต้องการ

list.remove(x)

เป็นการลบ item ตัวแรกที่เจอใน list ที่มีค่าเท่ากับ x หากไม่มีค่า x ใน list ก็จะแจ้ง error ออกมาก

list.clear()

เป็นการลบ item ทั้งหมด ออกจาก list

list.index(x)

เป็นการหาตำแหน่งของค่า x ที่เจอตัวแรก หากไม่มีค่า x ใน list ก็จะแจ้ง error ออกมาก

list.count(x)

เป็นการนับค่าที่ซ้ำของ x ใน list

list.sort()

เป็นการเรียงข้อมูลใน list จากน้อยไปมาก

list.reverse()

เป็นการสลับลำดับการเรียงข้อมูลจากหลังมาหน้า

tuple <=> list

การเปลี่ยน tuple ไปเป็น list สามารถทำได้โดยการใช้ function list() และเช่นเดียวกัน การแปลง list ไปเป็น tuple สามารถทำได้โดยการใช้ function tuple()

```
1  def main():
2      t = (1, 2, 3)
3      print(t)
4      print(type(t))
5
6      l = [1, 2, 3]
7      print(l)
8      print(type(l))
9
10     print(t[0])
11     print(l[2])
12
13     for i in t:
14         print(i)
15
16     s = 'Hello'
17
18     print(s[3])
19     print(s[2:])
20     print(s[1:4])
21     for i in s:
22         print(i)
23
24
25 if __name__ == "__main__":
26     main()
```

Creating associative lists with dictionaries dictionary

ตัวแปร dictionary ลักษณะคล้าย list แต่ index ของ dictionary เราจะเรียกว่า keys โดย key นี้จะเป็นข้อความ หรือตัวเลขก็ได้ ทั้งนี้ dictionary จะเก็บค่าเป็นคู่ระหว่าง key : value โดยที่ key นั้นจะต้องไม่ซ้ำกัน (จริง ๆ ซ้ำกันได้ แต่ค่าเดิมของ key นั้น จะถูกแทนที่ด้วยค่าใหม่)

การประกาศตัวแปร dictionary

ตัวแปร dictionary สร้างได้โดยใช้ {...} แยกแต่ละ key กับ value ด้วยเครื่องหมาย comma (,) คั่นระหว่าง key กับ value ด้วย colon (:)

Modules

Module คือไฟล์ python ที่มีการประกาศ function และคำสั่งต่าง ๆ อยู่ภายใน module ถูกสร้างมาเพื่อให้เก็บคำสั่งที่มีลักษณะคล้าย ๆ กัน ให้เก็บไว้ในที่เดียวกัน เพื่อให้สามารถเรียกใช้ได้ง่าย และเป็นระเบียบ

การเรียกใช้ module

การเรียกใช้งาน module สามารถทำได้ 2 แบบ

1. import module
2. from module import function

import module

มีรูปแบบการเรียกใช้งานดังนี้

```
import module1[, module2[,..., moduleN]]
```

การเรียกใช้งาน module แบบนี้ จะเป็นการเรียกใช้ทุก function ที่อยู่ใน module นั้นมาใช้งาน

```
1  import math
2
3
4  def main():
5      radius = input('Enter radius: ')
6
7      sphere = (4 / 3) * math.pi * math.pow(float(radius), 3)
8
9      print(sphere)
10
11 if __name__ == '__main__':
12     main()
```

ผลลัพธ์

```
Enter radius: 20
33510.32163829113
```

from module import function

มีรูปแบบการเรียกใช้งานดังนี้

```
from module import function1[, function2[... function]][*]
```

การเรียกใช้งาน module แบบนี้ จะเป็นการเรียกใช้เฉพาะ function ที่ต้องการที่อยู่ใน module นั้นมาใช้งานเท่านั้น ทำให้ลีนเบลีองหน่วยความจำน้อยกว่า แต่หากต้องการเรียกใช้ทุก function สามารถใช้ * ได้เลย

```
1  from math import pi, pow
2
3
4  def main():
5      radius = input('Enter radius: ')
6
7      sphere = (4 / 3) * pi * pow(float(radius), 3)
8
9      print(sphere)
10
11 if __name__ == '__main__':
12     main()
```

ผลลัพธ์

```
Enter radius: 20
33510.32163829113
```

Number Type Conversion

โดยปกติแล้ว ค่าที่ได้จากการทำงานต่าง ๆ จะเป็น **string** แต่บางครั้ง เราต้องเอาค่าเหล่านั้นไปทำการคำนวณ เราจำเป็นต้องแปลงค่า ก่อนที่จะทำการคำนวณได้ ซึ่ง Python ก็มี **method** สำหรับการแปลงค่ามาให้แล้ว ดังนี้

int(x)

เป็นการแปลงค่า x ให้เป็น **integer**

str(x)

เป็นการแปลงค่า x ให้เป็น **string**

float(x)

เป็นการแปลงค่า x ให้เป็น **float**

String Methods

Python มี **method** สำหรับการจัดการต่าง ๆ เกี่ยวกับ **string** ไว้ให้แล้ว โดยมีตัวอย่าง ดังนี้

str.capitalize()

เป็นการทำให้ตัวอักษรตัวแรกของคำเป็นตัวอักษรตัวใหญ่ ที่เหลือเป็นตัวอักษรตัวเล็ก

str.center(width [, fillchar])

เป็นการจัด **string** กึ่งกลาง ตาม **width** ที่กำหนด และจะเติม **fillchar** ลงไปให้ครบตามจำนวน **width** ที่เรากำหนด หาก **string** ที่แสดงมีค่าน้อยกว่า **width**

str.find(sub)

เป็นการหาตำแหน่งของข้อความใน **str** ด้วย **sub** โดยจะทำการหาตำแหน่งแรกสุดที่เจอ การหาว่ามีข้อความ **sub** ใน **str** หรือไม่นั้น สามารถใช้ **in** ได้เลย แต่หากต้องการรู้ตำแหน่ง ให้ใช้ **find()**

str.format()

เป็นการจัดรูปแบบข้อความ โดยทำการแทนข้อความที่ต้องการ ใน {} ที่อ้างอิงตาม index ของตำแหน่งที่ระบุ

str.isalnum()

เป็นการตรวจสอบว่า str เป็นข้อความที่ผลิตด้วยตัวอักษร และตัวเลข และอย่างน้อยต้องมี 1 ตัวอักษรจริงหรือไม่

str.isalpha()

เป็นการตรวจสอบว่า str เป็นข้อความที่มีแต่ตัวอักษรจริงหรือไม่

str.islower()

เป็นการตรวจสอบว่า str นี้ เป็นตัวอักษรตัวเล็กทั้งหมดจริงหรือไม่

str.isnumeric()

เป็นการตรวจสอบว่า str นี้ เป็นตัวเลขจริงหรือไม่

str.isupper()

เป็นการตรวจสอบว่า str นี้ เป็นตัวอักษรตัวใหญ่ทั้งหมดจริงหรือไม่

str.ljust(width [, fillchar])

เป็นการจัด string ชิดซ้าย ตาม width ที่กำหนด และจะเติม fillchar ลงไปให้ครบตามจำนวน width ที่เรากำหนด หาก string ที่แสดงมีค่าน้อยกว่า width

str.lower()

เป็นการทำให้ตัวอักษรทั้งหมดใน str เป็นตัวอักษรตัวเล็ก

str.join(iterable)

เป็นการเชื่อมต่อข้อมูลที่อยู่ใน iterable ด้วย str ทั้งนี้ ใน iterable ต้องเป็นข้อมูลชนิด string เท่านั้น

str.lstrip()

เป็นการลบซ่องว่างที่อยู่ทางด้านหน้าของ str ทิ้ง ทั้งหมด

str.replace(old, new[, count])

เป็นการค้นหาข้อความ old ใน str และแทนที่ด้วย new หากใส่ count มาด้วย จะทำการแทนที่ตามจำนวนที่กำหนดใน count

str.rjust(width [, fillchar])

เป็นการจัด string ชิดขวา ตาม width ที่กำหนด และจะเติม fillchar ลงไปให้ครบตามจำนวน width ที่เรากำหนด หาก string ที่แสดงมีค่าน้อยกว่า width

str.rstrip()

เป็นการลบซ่องว่างที่อยู่ทางด้านหลังของ str ทิ้ง ทั้งหมด

str.split(sep=None, maxsplit=-1)

เป็นการแบ่ง str ออกจากกันด้วย sep ผลลัพธ์ที่ได้จะเป็น list โดยค่าเริ่มต้นของ maxsplit มีค่าเป็น -1 หรือไม่กำหนดก็ได้ หมายถึง ให้ทำการแบ่ง str แบบไม่จำกัดจำนวนครั้งในการแบ่ง แต่หากกำหนด maxsplit หมายถึงให้ทำการแบ่ง str ออกจากกัน ตามจำนวนครั้งที่กำหนดใน maxsplit

str.strip()

เป็นการลบซ่องว่างที่อยู่ทางด้านหน้า และด้านหลังของ str ทิ้ง ทั้งหมด

str.upper()

เป็นการทำให้ตัวอักษรทั้งหมดใน str เป็นตัวอักษรตัวใหญ่

str.zfill(width)

เป็นการเติมเลข 0 ที่ด้านซ้ายของ str ให้ครบตามจำนวน width โดยจะนับเครื่องหมาย + - ด้วย

```

1  def main():
2      a = 'Hello'
3      b = 'gong'
4      pi = 22 / 7
5      c = '123456'
6
7      print('{}, my name is {}'.format(a, b))
8      print("{}{}, my name is {}".format(a, b))
9      print('{:.8f}'.format(pi).rjust(20, '*'))
10     print(a.ljust(10, '_'))
11     print(a.center(10, '*'))
12     print('*' * 20)
13     print(a.isdecimal())
14     print(str(pi).isnumeric())
15     print(c.isdecimal())
16     print(c.isdigit())
17     print(c.isnumeric())
18     print(b.islower())
19     print(a.upper().isupper())
20     print(a.replace('l', 'e', 1))
21
22     print('*' * 20)
23
24     s = 'Hello my name is Gong'
25     a = s.split()
26     print(a)
27
28     b = '::'.join(a)
29     print(b)
30
31     if __name__ == '__main__':
32         main()

```

ผลลัพธ์

```

Hello, my name is gong
gong, my name is Hello
*****3.14285714
Hello_____
**Hello***_
*****
False
False
True
True
True
True
True
Heelo
*****
['Hello', 'my', 'name', 'is', 'Gong']
Hello::my::name::is::Gong

```

Error and Exceptions

ความผิดพลาดในการเขียนโปรแกรม โดยส่วนใหญ่ เราสามารถแบ่งกลุ่มของความผิดพลาดออกเป็นได้ 2 กลุ่ม คือ **syntax error** และ **exception**

Syntax Errors

Syntax Errors คือความผิดพลาดในการพิมพ์ของเราวา ที่ผิดไปจากรูปแบบที่ Python กำหนดไว้ เช่น ลีบ : ไว้หลัง while หรือหลัง for หรือลืมย่อหน้าเข้ามา

```
1 def main():
2     while True~  
3         print('Hello')  
4  
5 if __name__ == "__main__":
6     main()
```

ผลลัพธ์

```
while True
^
SyntaxError: invalid syntax
```

ซึ่งความผิดพลาดแบบนี้ อาจเกิดจากความไม่เคยชิน ถ้าฝึกเขียนไปเรื่อย ๆ ความผิดพลาดแบบนี้ก็จะน้อยลง แต่ก็ยังมีความผิดพลาดอีกอย่างนึง ที่ถึงแม้จะไม่เกิด **syntax error** แต่ก็เกิด **error** ขณะที่รันโปรแกรม เรา เรียกว่า **exception**

Exceptions

Exception เกิดขึ้นได้ เมื่อเราจะเขียนคำสั่งต่าง ๆ ได้อย่างถูกต้องตาม **syntax** ของ Python และถ้า ตาม ซึ่ง ความผิดพลาด **exception** นี้จะเกิดขึ้นในระหว่างที่เรากำลังทำงาน หรือรันโปรแกรมอยู่

```
1     def main():
2         while True:
3             a = int(input('dividend: '))
4             b = int(input('denominator: '))
5
6             print(a / b)
7
8         if __name__ == "__main__":
9             main()
```

ผลลัพธ์

```
D:\Python35\python.exe "D:/Jobs/อบรม Basic Python/controlloop.py"
dividend: 18
denominator: 0
Traceback (most recent call last):
  File "D:/Jobs/อบรม Basic Python/controlloop.py", line 8, in <module>
    main()
  File "D:/Jobs/อบรม Basic Python/controlloop.py", line 6, in main
    print(a / b)
ZeroDivisionError: division by zero
```

ถ้าในกรณีนี้ เราสามารถแก้ไขคำสั่งได้ ด้วยการตรวจสอบค่า b ก่อน เช่น

```
1     def main():
2         while True:
3             a = int(input('dividend: '))
4             b = int(input('denominator: '))
5
6             if b == 0:
7                 print('denominator can not be zero, try again')
8             else:
9                 break
10
11             print(a / b)
12
13         if __name__ == "__main__":
14             main()
```

ผลลัพธ์

```
D:\Python35\python.exe "D:/Jobs/อนرم Ba
dividend: 18
denominator:
demominator can not be zero, try again
dividend: 18
denominator: 2
9.0
```

การเขียนคำสั่งแบบด้านบน สามารถตัดกได้ว่า ตัวแปร b มีค่าเป็น 0 หรือไม่ แต่ในความเป็นจริง ผู้ใช้สามารถกรอกตัวอักษรผสมกับตัวเลขได้ เช่น กัน ซึ่งเราอาจจะเขียนคำสั่งตัดไว้ได้อีก เช่น กัน แต่นั่นก็หมายความว่า เราต้องเขียนคำสั่งมากขึ้น ดังนั้นหลาย ๆ ภาษาจึงมีการสร้าง exception ขึ้นมา เพื่อค่อยติดตามว่า มีความผิดพลาดใด ๆ เกิดขึ้นบ้าง หรือไม่

ความสามารถนำ exception มาเขียนเพื่อตรวจสอบความผิดพลาดจากตัวอย่างข้างต้นได้ ดังนี้

```
1 def main():
2     while True:
3         try:
4             a = int(input('dividend: '))
5             b = int(input('denominator: '))
6             print('division is: ', a / b)
7
8         except ZeroDivisionError:
9             print('denominator can not be zero, try again')
10        except ValueError:
11            print('enter a number, try again')
12        else:
13            print('success')
14            break
15        finally:
16            print('*' * 20)
17
18
19 if __name__ == "__main__":
20     main()
```

```

dividend: 18
denominator: 0
denominator can not be zero, try again
*****
dividend: a
enter a number, try again
*****
dividend: 18
denominator: 2
division is: 9.0
success
*****

```

บรรทัดที่	คำอธิบาย
3	ใช้คำสั่ง try
8	ตักจับ error หากมี error ด้วยตัวหารเป็น 0
10	ตักจับ error หากมี error ด้วยการที่ไม่สามารถแปลงค่าเป็น int ได้จากบรรทัดที่ 4 และ 5
12	else จะทำงานก็ต่อเมื่อ ไม่มีการเกิด exception ใด ๆ
15	finally จะทำงานตลอดทุกครั้งไม่ว่าจะเกิด exception หรือไม่

Exception ที่ควรรู้จัก

Exception: ความผิดพลาดที่เกิดขึ้นทั้งหมด จะเป็น Exception หากเราไม่รู้ว่าจะตักจับเหตุการณ์ใด ก็ใช้ Exception นี้ได้

NameError: ตัวแปรที่เรียกใช้ไม่มีอยู่จริง

ValueError: ค่าของตัวแปร ไม่สามารถนำมาใช้ดำเนินการได้ ๆ หรือคำนวนได้

TypeError: ชนิดของตัวแปร ไม่สามารถนำมาใช้ดำเนินการได้ ๆ หรือคำนวนได้

Raising exceptions

Raising exceptions คือการบังคับให้เกิด exception

```

1 def main():
2     raise NameError('Name error test')
3
4
5 if __name__ == "__main__":
6     main()

```

ผลลัพธ์

```

Traceback (most recent call last):
  File "D:/Jobs/อบรม Basic Python/code/main.py", line 1, in <module>
    main()
  File "D:/Jobs/อบรม Basic Python/code/main.py", line 5, in main
    raise NameError('Name error test')
NameError: Name error test

```

Time

เราสามารถใช้งาน time ซึ่งเป็นการใช้งานเกี่ยวกับวัน เวลาได้ แต่ก่อนที่จะใช้งาน ต้องทำการ import time มา ก่อน จึงจะสามารถใช้งานได้
time มี function ต่าง ๆ ที่น่าสนใจ ดังนี้

time.localtime([secs])

ทำการแปลงเวลา ให้อยู่ในรูปแบบของ struct_time ซึ่งประกอบไปด้วย tm_year, tm_mon, tm_mday, tm_hour, tm_min, tm_sec, tm_wday, tm_yday, tm_isdst หากเรามีได้ใส่ sec ก็จะนำเอกสารเวลา ปัจจุบันมาใช้งาน

time.sleep(secs)

ห่วงเวลา ตามจำนวน secs ที่ใส่มา

time.strftime(format[, t])

ทำการแปลง struct_time ให้อยู่ในรูปแบบของข้อความ ตาม format ที่เราต้องการ ถ้าเราไม่ได้ใส่ t ก็จะใช้เวลาปัจจุบันในการทำงาน ทั้งนี้ format ต้องเป็นข้อความเท่านั้น

format	ความหมาย
%a	วัน 3 ตัวอักษร

<code>format</code>	ความหมาย
<code>%A</code>	วัน
<code>%b</code>	เดือน 3 ตัวอักษร
<code>%B</code>	เดือน
<code>%c</code>	วันที่ เดือน ปี เวลา
<code>%d</code>	วันที่ [01,31]
<code>%H</code>	ชั่วโมง ในรูปแบบ 24 ชม. [00,23]
<code>%I</code>	ชั่วโมง ในรูปแบบ 12 ชม. [01,12]
<code>%j</code>	วันที่เท่าไหร่ ในปีปัจจุบัน [001,366]
<code>%m</code>	เดือน [01,12]
<code>%M</code>	นาที [00,59]
<code>%S</code>	วินาที [00,61]
<code>%w</code>	วันในสัปดาห์ [0 (Sunday), 6]
<code>%x</code>	เดือน/วัน/ปี
<code>%X</code>	เวลาปัจจุบัน
<code>%y</code>	ปี 2 หลัก
<code>%Y</code>	ปี 4 หลัก

time.time()

ได้ค่าเวลา หน่วยเป็นวินาที ตั้งแต่วันที่ 1 มกราคม 1970

```
1 import time
2
3
4 def main():
5     my_time = time.time()
6     print(my_time)
7
8     my_local1 = time.localtime()
9     my_local2 = time.localtime(my_time)
10    print(my_local1)
11    print(my_local2)
12
13    time.sleep(1)
14
15    print(my_local1[0])
16
17    temp = 'Today is %A %d %B %Y <=> %d/%m/%y <=> %x'
18
19    print(time.strftime(temp))
20    print(time.ctime(time.time()))
21
22
23 if __name__ == '__main__':
24     main()
```

ผลลัพธ์

```
1456765537.3466508
time.struct_time(tm_year=2016, tm_mon=3, tm_mday=1, tm_hour=0, tm_min=5, tm_sec=37, tm_wday=1, tm_yday=61, tm_isdst=0)
time.struct_time(tm_year=2016, tm_mon=3, tm_mday=1, tm_hour=0, tm_min=5, tm_sec=37, tm_wday=1, tm_yday=61, tm_isdst=0)
2016
Today is Tuesday 01 March 2016 <=> 01/03/16 <=> 03/01/16
Tue Mar  1 00:05:38 2016
```

Functions

Function ถูกสร้างขึ้นมา เพื่อให้เกิดการ reuse คำสั่งที่ใช้บ่อย ๆ ทำให้เราเขียนโปรแกรมที่ซ้ำซ้อนน้อยลง และหากมีการแก้ไขก็สามารถที่จะแก้ไขได้ง่าย

Defining functions

ใช้ def ในการสร้าง function ต่อด้วยชื่อ function ที่เราต้องการ ภายใต้ function จะเป็นต้องมีคำสั่งเสมอ แต่หากเราไม่รู้ว่าจะเขียนคำสั่งใด ในช่วงแรก ให้ใช้ pass ไปก่อน

Function จะมีการรับ argument หรือไม่มีก็ได้ ถ้ามีการรับ argument ตอนที่เรียกใช้ function เรา ก็ต้องส่ง parameter ไปตามจำนวน argument ที่ระบุไว้ แต่ Python อนุญาตให้เรากำหนดค่า default ให้กับ argument ได้ หากไม่มีการส่งค่า parameter มา

```
1  def main():
2      test_function()
3      test_function_2(18, 81)
4      test_function_3(18)
5      test_function_3(18, 181)
6
7
8  def test_function():
9      # pass
10     print("Hi, I'm a function")
11
12
13 def test_function_2(number1, number2):
14     print("Hi, I'm a function number: ", number1, number2)
15
16
17 def test_function_3(number1, number2=81):
18     print("Hi, I'm a function number: ", number1, number2)
19
20
21 if __name__ == "__main__":
22     main()
```

บรรทัดที่	คำอธิบาย
8	ประกาศ function test_function() โดยไม่มีการรับค่าใด ๆ
13	ประกาศ function test_function_2() โดยมีการรับค่า จำนวน 2 ค่าเสมอ
17	ประกาศ function test_function_3() function นี้ มีการประกาศค่า default ไว้ที่ number2=81 ซึ่งหากเราเรียกใช้ function นี้ เราสามารถส่ง argument มาได้ทั้ง 1 ค่า และ 2 ค่า

บรรทัดที่	คำอธิบาย
	ทั้งนี้ หากส่ง argument มาแค่ ค่าเดียว ค่าที่เหลือ จะใช้ค่าจาก default ที่กำหนดไว้

ผลลัพธ์

```
Hi, I'm a function
Hi, I'm a function number: 18 81
Hi, I'm a function number: 18 81
Hi, I'm a function number: 18 181
```

Using lists of arguments

จากหัวข้อก่อนหน้า เราจะเห็นว่า function ใน Python นั้น สามารถที่จะกำหนด arguments ที่จะส่งค่ามาด้วย หรือไม่ก็ได้ (optional) ทั้งนี้ Python ยังสามารถรับค่า argument ที่มีลักษณะแบบนี้ ได้แบบไม่จำกัดจำนวนได้อีกด้วย

Argument ที่เป็น optional แบบนี้ ต้องมีลักษณะ * ทั้งนี้ argument ที่มี * นำหน้า จะรับค่ามา กี่ค่าก็ได้ แต่ค่าที่รับมาจะเป็นชนิด tuple

```
1  def main():
2      test_function(1, 2, 3, 4)
3
4
5  def test_function(num1, num2, *nums):
6      print(num1, num2)
7      for i in nums:
8          print(i)
9
10
11 if __name__ == "__main__":
12     main()
```

บรรทัดที่	คำอธิบาย
5	ประกาศ function test_function() จะต้องมีการรับค่า num1, num2 เสมอ ส่วน *num จะมีข้อมูลหรือไม่ก็ได้ ถ้าจะมีข้อมูล จะมีกี่ตัวก็ได้

ผลลัพธ์

1 2

3

4

Using named function arguments

บางครั้งเราต้องการส่งชื่อตัวแปร และค่าไปยัง function อื่นด้วย ซึ่ง Python ก็ได้เตรียมคุณสมบัตินี้ไว้ให้แล้ว โดยใน argument ใน function จะมีสัญลักษณ์ เป็น ** นำหน้า โดยค่าที่รับมาจะเป็น dictionary

ทั้งนี้ หาก function นั้นมีการรับค่าทั้งแบบ argument ที่จำเป็นต้องใส่ argument ที่เป็น optional และปิดท้ายด้วย argument ที่รับค่าพร้อมทั้งชื่อตัวแปร และค่าของตัวแปร

```
1  def main():
2      test_function(one=1, two=2, three=3)
3      test_function2(1, 2, 3, 4, 5, one=1, two=2, three=3)
4      test_function2(1, 2, one=8)
5
6
7  def test_function(**nums):
8      print(nums)
9      print(nums['one'])
10     for k, v in nums.items():
11         print(k, v)
12     print('*' * 20)
13
14
15 def test_function2(num1, num2, *num, **kwargs):
16     print(num1, num2)
17     print(num)
18     print(kwargs)
19     print('*' * 20)
20
21 if __name__ == "__main__":
22     main()
```

```
{'one': 1, 'two': 2, 'three': 3}
1
one 1
two 2
three 3
*****
1 2
(3, 4, 5)
{'one': 1, 'two': 2, 'three': 3}
*****
1 2
()
{'one': 8}
*****
```

Returning values from functions

ในการสร้าง function เราสามารถที่จะส่งค่ากลับไปให้กับผู้ที่เรียกใช้ function ได้ โดยใช้ return ทั้งนี้ return สามารถส่งกลับไป ด้วยชนิดตัวแปรได้ก็ได้ หรือจะ return เป็น object ก็ได้

```
1  def main():
2      print(test_function())
3      print(test_function2())
4      for i in test_function2():
5          print(i, end=' ')
6
7
8  def test_function():
9      return 'Hello World'
10
11
12 def test_function2():
13     return range(18)
14
15 if __name__ == "__main__":
16     main()
```

ผลลัพธ์

```
Hello World
range(0, 18)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
```

Classes

Python นั้นถูกสร้างมาเป็น object-oriented ตั้งแต่แรกเริ่มแล้ว เพราะมีการใช้งาน class และ object ต่าง ๆ

Class

Class คือการกำหนดรูปร่างโครงสร้าง ของ object ที่เราต้องการ class เปรียบเสมือนพิมพ์เขียว ที่บอกเราได้ว่า มีการเก็บข้อมูลอะไรบ้าง มีการทำงานอะไรได้บ้าง ซึ่งเราเรียกข้อมูลที่ต้องการเก็บว่า attribute และการทำงานต่าง ๆ ที่คลาสสามารถทำได้ เราเรียกว่า method ทั้งนี้ เราสามารถเข้าถึง attribute และ method ได้ผ่านทางการใช้สัญลักษณ์ .

Class variable

Class variable เป็นตัวแปรชนิดเด็กได้ แต่ตัวแปรนี้ จะใช้ร่วมกับทุก ๆ instance ที่ถูกสร้างจากคลาสเดียวกัน class variable นี้ถูกสร้างภายใต้ class แต่ไม่อยู่ภายใต้ method ใด ๆ

Data member

Data member คือ class variable หรือ instance variable ที่เก็บข้อมูลต่าง ๆ ที่สัมพันธ์กับ class และ object ของมัน

Instance variable

Instance variable เป็นตัวแปรที่ถูกประกาศไว้ภายใต้ method และจะทำการเก็บข้อมูลสำหรับ instance ปัจจุบันเท่านั้น

Instance

Instance คือ object ที่ถูกสร้างมาจาก class โดยจะเป็น object ที่มีคุณสมบัติเช่นเดียวกับ class

Method

Method คือ function ที่ถูกสร้างไว้ภายใต้ class

Object

Object คือ โครงสร้างของข้อมูลที่ได้จาก class โดย object นี้ประกอบด้วย data member (ทั้ง class variable และ instance variable) และ method

การสร้าง class

เราสามารถสร้าง class ขึ้นมาได้โดยง่าย ซึ่งการสร้าง class นี้ เมื่อเทียบกับการสร้าง function ทั้งนี้ ภายใน class จะประกอบไปด้วย attribute และ method

ยกตัวอย่างการสร้าง class ชื่อ Customer สำหรับลูกค้าธนาคาร Alpaca โดย จะเก็บข้อมูล name คือชื่อของลูกค้า และ balance คือจำนวนเงินของลูกค้า

Customer.py

```
1  class Customer:
2      """A customer of Alpaca Bank
3
4      Attributes:
5          name: customer's name
6          balance: current balance of customer
7      """
8
9      # class variable
10     customer_count = 0
11
12     # constructor
13     def __init__(self, name, balance=0.0):
14
15         # instance variable
16         self.name = name
17         self.balance = balance
18         Customer.customer_count += 1
19
20     # method
21     def withdraw(self, amount):
22         self.balance = self.balance - amount
23         return self.balance
24
25     def deposit(self, amount):
26         self.balance = self.balance + amount
27         return self.balance
28
29     def check_balance(self):
30         print("Your current balance is", self.balance)
```

self

จากรูปเราจะเห็นว่า `self` อยู่ในทุก ๆ method นั้นก็ เพราะว่า `self` คือ ตัวแทนของ `instance` ปัจจุบันที่เรียกใช้นั่นเอง

`__init__`

`__init__` เปรียบเสมือนกับ constructor ของ class โดย `__init__` จะถูกเรียกใช้ครั้งแรกที่มีการสร้าง `instance`

การนำ class ไปใช้งาน

เราสามารถนำ class ไปใช้งานได้ โดยการ ใช้คำสั่ง `from`

```
1  from Customer import *
2
3
4  def main():
5      a = Customer("A")
6      a.check_balance()
7      a.deposit(180)
8      a.check_balance()
9      a.withdraw(100)
10     a.check_balance()
11
12     print("-" * 20)
13
14     b = Customer("B", 250)
15     b.check_balance()
16     b.deposit(80)
17     b.check_balance()
18     b.withdraw(18)
19     b.check_balance()
20
21 if __name__ == '__main__':
22     main()
```

```
Your current balance is 0.0
Your current balance is 180.0
Your current balance is 80.0
-----
Your current balance is 250
Your current balance is 330
Your current balance is 312
```

File I/O

Python ได้เตรียมคำสั่งสำหรับใช้ในการ เปิด อ่าน และเขียนไฟล์ไว้ให้แล้ว โดยการทำงาน ก็ยังคงยึดหลักของ Python ไว้ ก็คือ เขียนคำสั่งง่ายๆ

Method ที่ใช้กับ file ได้ เช่น

open(file [, mode])

ใช้สำหรับเปิดไฟล์ที่ต้องการ

file คือ path ของไฟล์ที่เราต้องการเปิด ทั้งนี้จะอ้างอิงจากตำแหน่งของไฟล์ .py ที่กำลังทำงาน หรือ path ที่แท้จริงก็ได้

mode คือ โหมดของการเปิดไฟล์ โดยมีค่า default เป็น โหมด r ทั้งนี้โหมดสำหรับการเปิดไฟล์ มีดังนี้

Mode	ความหมาย
r	เปิดไฟล์เพื่ออ่านเท่านั้น (เป็นค่า default)
w	เปิดไฟล์เพื่อเขียน โดยลบข้อมูลเดิมทิ้งก่อน
a	เปิดไฟล์เพื่อเขียน โดยเขียนข้อมูลใหม่ ต่อท้ายข้อมูลเดิม
b	โหมด binary
+	เปิดไฟล์ เพื่อทำการอ่าน และเขียน

โหมด + นี้ สามารถใช้ร่วมกับ r w และ a ได้ โดยจะทำให้มีคุณสมบัติการอ่าน หรือการเขียนเพิ่มขึ้น เช่น r+ ทำให้เปิดไฟล์เพื่ออ่าน และเขียนได้ เป็นต้น

เมื่อเรียกใช้ open ควรมีตัวแปรรับด้วย เพื่อใช้สำหรับจัดการไฟล์ที่เปิดมา

f.read([size])

สมมติว่า f คือ ตัวแปรที่เก็บค่าจากการ open ไฟล์

`read()` เป็น function ที่ใช้สำหรับอ่านข้อมูลจากไฟล์ โดย `read()` นี้ จะอ่านข้อมูลทั้งหมดของมาจากการไฟล์ และเก็บไว้ในหน่วยความจำของเครื่อง

`size` เป็นจำนวนตัวอักษรที่เราต้องการอ่าน เราสามารถใส่ตัวเลขลงไปเพื่อรับว่า ต้องการอ่านข้อมูลมากกี่ตัวอักษร แต่หากไม่กำหนด จะเป็นการอ่านข้อมูลมากทั้งหมด

f.readline()

`readline()` เป็น function สำหรับการอ่านข้อมูลของมาจากการไฟล์ที่ละacco ซึ่งวิธีนี้ไม่ทำให้สิ้นเปลืองหน่วยความจำของเครื่องมากนัก

f.seek(index)

`seek()` เป็น function ในการเลื่อนตำแหน่งในการอ่านข้อมูลจากไฟล์

`index` คือ ตำแหน่งที่เราต้องการอ่านข้อมูล หากกำหนดให้มีค่าเป็น 0 หมายถึง เลื่อนไปยังจุดเริ่มต้นของไฟล์

f.write(value)

`write()` เป็น function สำหรับเขียนข้อมูลลงไฟล์ที่เราต้องการ โดย function นี้จะทำการคืนค่าเป็นจำนวนตัวอักษรที่เขียนลงไฟล์ กลับคืนมาให้

`value` คือ ข้อมูลที่เราต้องการเขียนลงไฟล์

f.tell()

`tell()` เป็น function ที่ใช้สำหรับบอกว่า ตอนนี้ตำแหน่งปัจจุบันที่ใช้อ่านข้อมูลนั้นอยู่ที่ตำแหน่งเท่าไหร่

f.close()

`close()` เป็น function สำหรับปิดไฟล์ เมื่อเรากระทำการกับไฟล์เรียบร้อยแล้ว

```
1  def main():
2      f = open('testfile.txt', 'r+')
3
4      print(f.tell())
5      print(f.read())
6
7      print('*' * 10 + "1" + '*' * 10)
8
9      f.seek(3)
10
11     print(f.tell())
12     print(f.read())
13
14     print('*' * 10 + "2" + '*' * 10)
15
16     f.seek(0)
17
18     print(f.readline())
19     print(f.readline())
20     print(f.readline())
21
```

```
22     print('*' * 10 + "3" + '*' * 10)
23
24     f.seek(0)
25
26     for line in f:
27         print(line, end='')
28
29     print('*' * 10 + "4" + '*' * 10)
30
31     with open('testfile.txt', 'r') as ff:
32         for l in ff:
33             print(l, end='')
34
35     msg = "\nBye"
36
37     f.write(msg)
38
39     print('*' * 10 + "5" + '*' * 10)
40
41     f.seek(0)
42
43     for line in f:
44         print(line, end='')
45
46     f.close()
47
48 if __name__ == '__main__':
49     main()
```

ผลลัพธ์

```
0
Good
morning
Bye
Bye
Bye
*****1*****
3
d
morning
Bye
Bye
Bye
*****2*****
Good

morning

Bye

*****3*****
Good
morning
Bye
Bye
Bye*****4*****
Good
morning
Bye
Bye
Bye*****5*****
Good
morning
Bye
Bye
Bye
Bye
```