บทที่ 3 ภาษาไพธอนสำหรับวิทยาศาสตร์ข้อมูล

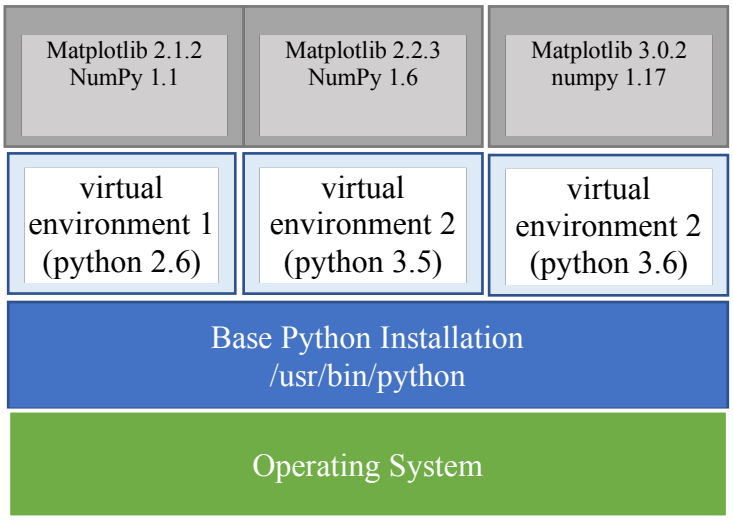
หัวข้อหลัก

- ทบทวนภาษาไพธอนเวอร์ชั่น 3 สำหรับงานทางวิทยาศาสตร์ข้อมูล

1. Virtual Environments

ไลบารี่ที่ใช้สำหรับงานทางวิทยาศาสตร์ข้อมูล เช่น matplotlib, numpy, pandas ไม่ได้เป็นส่วนหนึ่งของไลบารี่มาตรฐานในภาษาไพธอน (Python Standard Library) เมื่อนักวิทยาศาสตร์ข้อมูลจำเป็นต้องใช้ไลบารี่ภายนอก (external libraries) เหล่านี้ในการวิเคราะห์ข้อมูล จึงจำเป็นต้องติดตั้งเพิ่มเติมเข้าไปในแพลตฟอร์มของภาษาไพธอน เนื่องจากไลบารี่แต่ละตัวมีการปรับปรุงเวอร์ชั่นใหม่อยู่เสมอ เมื่อนักวิทยาศาสตร์ข้อมูลมีโปรเจ็กต์หลายโปรเจ็กต์ในระบบ ก็อาจทำให้เกิดความขัดแย้ง (conflicts) ระหว่างไลบารี่ต่างเวอร์ชั่นกันที่ถูกใช้ในโปรเจ็กต์เหล่านั้นได้

วิธีป้องกันปัญหาที่เกิดจากการใช้งานไลบารี่ต่างเวอร์ชั่นร่วมกันบนแพลตฟอร์มภาษาไพธอน สามารถทำได้โดยการใช้สภาพแวดล้อมเสมือน หรือ virtual environments เพื่อสร้างสภาพแวดล้อมสำหรับพัฒนาโปรแกรมภาษาไพธอน ที่เป็นอิสระจากแพลตฟอร์มภาษาไพธอน และสภาพแวดล้อมอื่นในระบบ ดังแสดงในรูปที่ 1



รูปที่ 1 Python virtual environments

เครื่องมือสำหรับสร้าง Virtual environments ในภาษาไพธอน มีให้เลือกหลายตัว เช่น venv, virtualenv, Anaconda Python ในรายวิชานี้เราจะเลือกใช้ Anaconda Python เป็นเครื่องมือสำหรับสร้างสภาพแวดล้อมเสมือนและจัดการแพกเกจในโปรเจ็กต์ของเรา

การติดตั้ง Anaconda Python ลงบนระบบปฏิบัติการวินโดวส์

1. หากเครื่องของนักศึกษายังไม่ได้ติดตั้ง Python 3 ให้ทำการติดตั้ง Python 3 ลงในเครื่องก่อน
    a. ดาวน์โหลดโปรแกรมติดตั้งจาก https://www.python.org/ftp/python/3.7.2/python-3.7.2.exe
    b. จากนั้น รันโปรแกรมและปฏิบัติตาม installation wizard เพื่อติดตั้ง Python 3.7.2

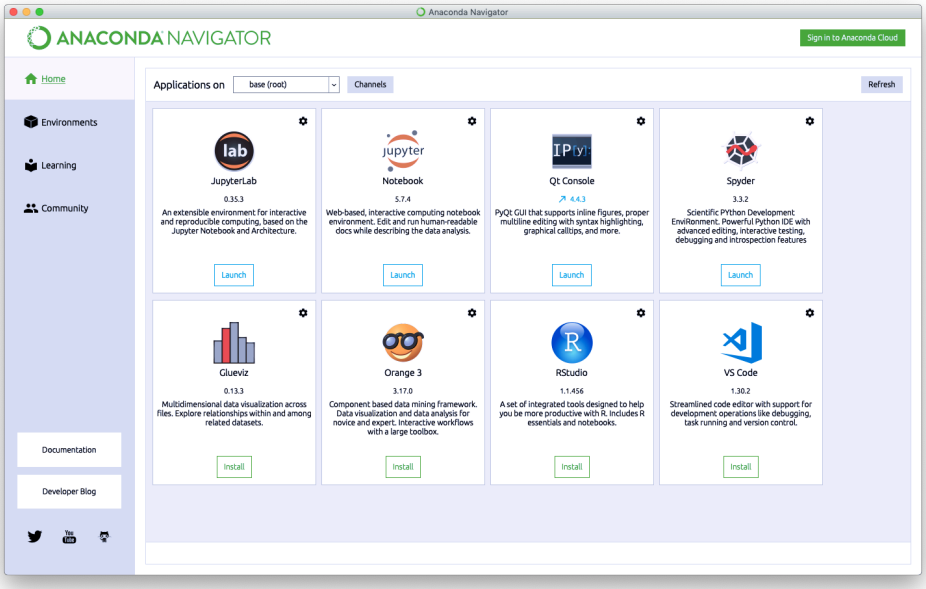
2. ดาวน์โหลดและรันโปรแกรม Anaconda Windows Installer (64-Bit Graphical Installer) สำหรับ Python 3.7 (ลิงก์ดาวน์โหลด: https://repo.anaconda.com/archive/Anaconda3-2018.12-Windows-x86_64.exe)

หมายเหตุ: รายละเอียดของวิธีการติดตั้ง Anaconda Python บนวินโดวส์ สามารถศึกษาเพิ่มเติมได้จาก

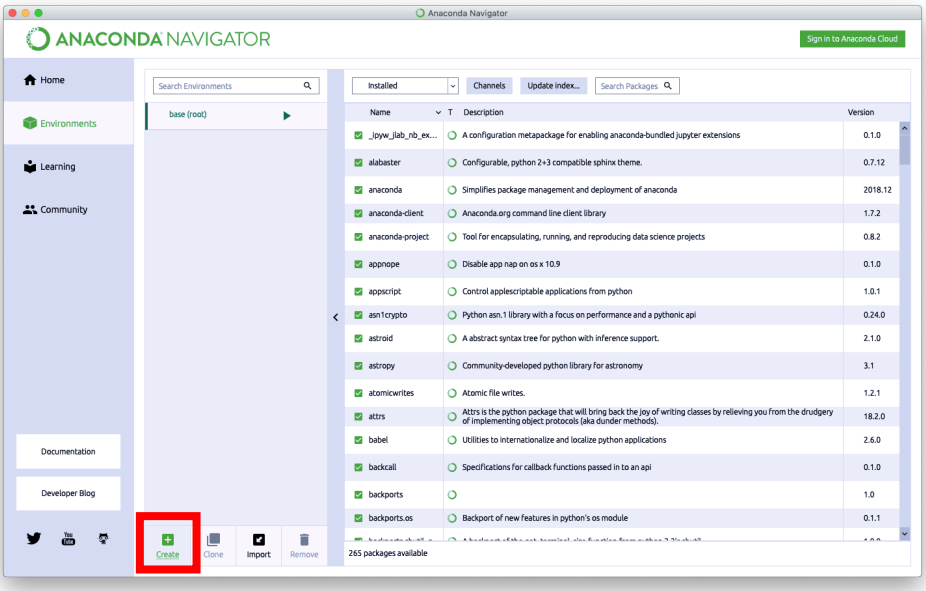
- https://www.datacamp.com/community/tutorials/installing-anaconda-windows
- https://docs.anaconda.com/anaconda/install/windows/

การสร้าง virtual environment ด้วย Anaconda

1. เปิดโปรแกรม Anaconda Navigator



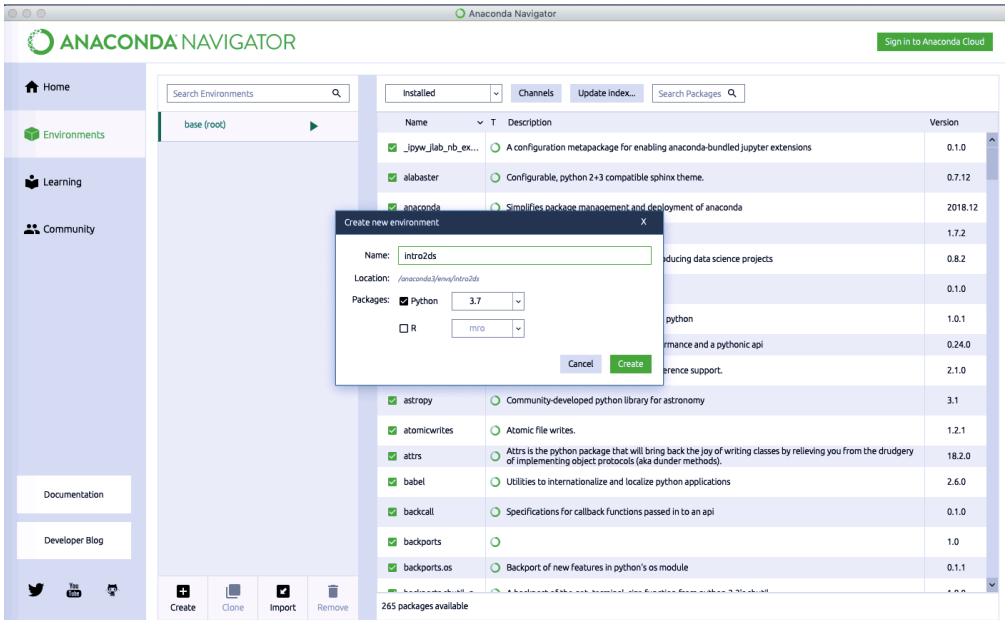
2. เลือกแท็บ Environments แล้วคลิกปุ่ม Create



3. สร้าง virtual environment ใหม่ ด้วยค่าพารามิเตอร์

   a. Name: intro2ds

   b. Python: 3.7



กดปุ่ม Create จากนั้น รอให้ระบบสร้าง virtual environment ใหม่ ให้เสร็จสิ้น

การใช้งานและการติดตั้งโปรแกรมภายใน virtual environment

1. เปิด Anaconda Prompt และเข้าเรียกใช้ virtual environment “intro2ds” โดยรันคำสั่ง:

   > activate intro2ds

2. ติดตั้งแพกเกจ ipython โดยรันคำสั่งต่อไปนี้

   > pip install ipython

3. เมื่อใช้งาน virtual environment เสร็จ สามารถปิด virtual environment ได้ โดยรันคำสั่ง:

   > deactivate

2. ทบทวนภาษาไพธอน

2.1 Modules

ฟีเจอร์ส่วนใหญ่ในภาษาไพธอนไม่ได้ถูกโหลดอัตโนมัติ โปรแกรมเมอร์จะต้องโหลดโมดูลโดยใช้คำสั่ง import เมื่อต้องการใช้ฟีเจอร์เหล่านั้น

ตัวอย่าง

- **รูปแบบคำสั่ง:** import *module_name*

```
In [19]: import math            # โหลดโมดูล math

In [20]: y = math.log(10.0)     # เรียกใช้ฟังก์ชัน log จากโมดูล math
```

- **รูปแบบคำสั่ง:** import *module_name* as *alias*

```
In [4]: import math as m        # โหลดโมดูล math และให้มีชื่อเรียกว่า m

In [5]: y = m.log(10.0)         # เรียกใช้ฟังก์ชัน log จากโมดูล math
```

- **รูปแบบคำสั่ง:** from *module_name* import *class_or_other_values_inside_module*

```
In [10]: from collections import Counter      # โหลดคลาส Counter จากโมดูล collections

In [11]: my_counter = Counter()               # สร้างอ็อบเจ็กต์ my_counter
```

ข้อควรระวัง ไม่ควรโหลดโมดูลทั้งโมดูล เข้ามาในโปรแกรม เนื่องจากอาจทำให้เกิดปัญหาจากความซ้ำซ้อนของชื่อที่ใช้ในโปรแกรมได้ เช่น

```
In [16]: match = 10

In [17]: print(match)
10

In [18]: from re import *               # โมดูล re มีฟังก์ชัน match

In [19]: print(match)                   # ค่า match เปลี่ยนจาก 10 เป็นฟังก์ชัน match!
<function match at 0x102200a60>
```

2.2 Functions

เราสามารถสร้างฟังก์ชันในภาษาไพธอน โดยใช้คีย์เวิร์ด def เช่น

```
In [20]: def cube(x):
    ...:     """
    ...:     ฟังก์ชัน cube รีเทิร์นค่า x ยกกำลัง 3
    ...:     """
    ...:     return x * x * x
    ...:

In [21]: cube( 3 )
Out[21]: 27

In [22]: cube( 3.3 )
Out[22]: 35.937
```

ฟังก์ชันสามารถถูกส่งเป็นค่าอินพุทพารามิเตอร์ของฟังก์ชันได้ เช่น

```
In [37]: def apply_to_two(f):
    ...:     """เรียกฟังก์ชัน f โดยใช้ 2 เป็นอินพุทพารามิเตอร์"""
    ...:     return f(2)
    ...:

In [38]: apply_to_two(lambda x: x * 3)          # สร้างฟังก์ชันโดยใช้คีย์เวิร์ด lambda
Out[38]: 6
```

เราสามารถกำหนดค่า default ของอินพุทพารามิเตอร์ของฟังก์ชันได้ ตัวอย่างเช่น

```
In [39]: def hello(name = "World!"):
    ...:     print("Hello ", name)
    ...:

In [40]: hello()
Hello  World!

In [41]: hello("Python")
Hello  Python

In [42]: hello(name = "Data Science")
Hello  Data Science
```

2.3 Strings

ในภาษาไพธอน เราใช้เครื่องหมาย single quotation หรือ double quotation ในการกำหนดค่าสตริง

```
In [43]: single_quote = 'random forest'

In [44]: double_quote = "random forest"
```

ตัวอักขระพิเศษ เช่น newline, tab จะขึ้นต้นด้วยเครื่องหมาย backslash

```
In [47]: new_line = "\n"

In [48]: len(new_line)
Out[48]: 1
```

หากต้องการให้เครื่องหมาย backslash มีค่าเป็นตัวอักขระ '\' (แทนที่จะเป็นสัญลักษณ์เริ่มต้นของอักขระพิเศษ) สามารถทำได้โดยสร้าง *raw string* โดยใช้ **r" "** เช่น

```
In [49]: slash_not_tab = r"\t"

In [50]: slash_not_tab
Out[50]: '\\t'

In [51]: len(slash_not_tab)
Out[51]: 2

In [52]: len("\t")
Out[52]: 1
```

การสร้าง multi-line strings สามารถทำได้โดยใช้ triple-[double]-quote

```
In [1]: multiline_string = """นี่คือสตริง
   ...: ที่มีหลายบรรทัด
   ...: ต้องใช้เครื่องหมายคำ พูด  "
   ...: สามตัว ในการนิยาม
   ...: """

In [2]: multiline_string
Out[2]: 'นี่คือสตริง \nที่มีหลายบรรทัด\nต้องใช้เครื่องหมายคำ พูด  "  \nสามตัว ในการนิยาม\n'
```

Python 3.6 ได้เพิ่มฟีเจอร์ **f-string** มาให้ ซึ่งทำให้เราสามารถแทนค่าเข้าไปในสตริงได้ง่ายขึ้น

```
In [21]: student_id = "39014033"

In [22]: student_name = "กุลวดี สมบูรณ์วิวัฒน์"

In [23]: stdrec1 = student_id + " " + student_name                # ใช้ string addition

In [24]: stdrec2 = "{0} {1}".format(student_id, student_name)     # ใช้ string.format

In [25]: stdrec3 = f"{student_id} {student_name}"                  # ใช้ f-string

In [26]: stdrec1
Out[26]: '39014033 กุลวดี สมบูรณ์วิวัฒน์'

In [27]: stdrec2
Out[27]: '39014033 กุลวดี สมบูรณ์วิวัฒน์'

In [28]: stdrec3
Out[28]: '39014033 กุลวดี สมบูรณ์วิวัฒน์'
```

2.4 Exceptions

เมื่อมีความผิดพลาดเกิดขึ้นในโปรแกรม Python runtime จะสร้าง exception object ขึ้น หากไม่มีการเตรียมจัดการ exceptions ไว้ภายในโปรแกรม โปรแกรมของเราก็จะ crash ในที่สุด วิธีการจัดการ exceptions ทำได้โดยการดักจับ exceptions ด้วยบล็อค **try .. .except**

```
In [33]: try:
   ...:     L = []
   ...:     print( L[0] )
   ...: except IndexError:
   ...:     print("ไม่สามารถเข้าถึง L[0] ได้")
   ...:
ไม่สามารถเข้าถึง L[0] ได้
```

2.5 Data structures

โครงสร้างข้อมูลในภาษาไพธอน ที่ใช้บ่อยในงานทางวิทยาศาสตร์ข้อมูล ได้แก่

- Lists
- Tuples
- Dictionaries
- Counter
- Sets

2.5.1 Lists

List ในภาษาไพธอนเป็นโครงสร้างข้อมูลสำหรับจับกลุ่ม items หลายๆ ตัว ไว้ด้วยกันแบบมีลำดับ (ordered collections) ลิสต์ในภาษาไพธอนเทียบได้กับอาร์เรย์ในภาษาอื่น ๆ แต่ลิสต์ในภาษาไพธอนจะมีฟังก์ชันให้ใช้งานมากกว่า เราสามารถสร้างลิสต์ได้โดยเขียนรายการของ items ลงในวงเล็บเหลี่ยม ([...]) ตามลำดับคั่นด้วยเครื่องหมายจุลภาค (,)

```
In [47]: my_integers = [1, 2, 3]          # สร้าง list ของจำนวนเต็ม

In [48]: various_things = ["hello", 0.9, math.pi, True]    # items ในลิสต์ มีได้หลายประเภท ไม่จำเป็นต้องมีชนิดเดียวกันทุกตัว

In [49]: nested_list = [my_integers, [], various_things]   # list สามารถเป็นสมาชิกของ list อื่นได้

In [50]: len( my_integers )               # จำนวนสมาชิก ใน ลิสต์
Out[50]: 3

In [51]: sum( my_integers )               # ผลรวมของจำนวนเต็มทุกตัวใน ลิสต์  my_intgers
Out[51]: 6
```

การเข้าถึงสมาชิกตัวที่ *n* ของลิสต์ ทำได้โดยใช้เครื่องหมายวงเล็บเหลี่ยม

```
In [52]: x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [53]: first_element = x[0]

In [54]: second_element = x[1]

In [55]: last_element = x[-1]

In [56]: next_to_last_element = x[-2]

In [57]: first_element, second_element, last_element, next_to_last_element
Out[57]: (0, 1, 9, 8)

In [58]: x[0] = -9

In [59]: x
Out[59]: [-9, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

เราสามารถตัดเฉพาะบางส่วนของลิสต์ออกมาได้ (เรียกว่า การ slice list) โดยการใช้เครื่องหมายวงเล็บเหลี่ยม และการกำหนด slice ตามรูปแบบ *beg_index_inclusive : end_index_non_inclusive : stride*

```
In [62]: x
Out[62]: [-9, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [63]: first_four = x[0:4]

In [64]: first_four
Out[64]: [-9, 1, 2, 3]

In [65]: x[:4]
Out[65]: [-9, 1, 2, 3]

In [66]: fourth_to_end = x[3:]

In [67]: fourth_to_end
Out[67]: [3, 4, 5, 6, 7, 8, 9]

In [68]: second_to_five = x[1:5]

In [69]: second_to_five
Out[69]: [1, 2, 3, 4]

In [70]: remove_first_and_last_elements = x[1:-1]

In [71]: remove_first_and_last_elements
Out[71]: [1, 2, 3, 4, 5, 6, 7, 8]
```

```
In [74]: copy_of_x = x[:]

In [75]: copy_of_x
Out[75]: [-9, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [76]: every_second = x[::2]

In [77]: every_second
Out[77]: [-9, 2, 4, 6, 8]

In [78]: six_to_three = x[6:2:-1]

In [79]: six_to_three
Out[79]: [6, 5, 4, 3]
```

การตรวจสอบว่า item ใด ๆ เป็นสมาชิกของลิสต์หรือไม่ ทำได้โดยใช้โอเปอเรเตอร์ **in**

```
In [80]: 1 in six_to_three
Out[80]: False

In [81]: 4 in six_to_three
Out[81]: True
```

การต่อลิสต์สองอันเข้าด้วยกันทำได้โดยใช้เมธอด **extend** หรือใช้โอเปอเรชั่น list addition

```
In [82]: x = [1, 2, 3]

In [83]: x.extend([4, 5, 6])

In [84]: x
Out[84]: [1, 2, 3, 4, 5, 6]

In [85]: x = [1, 2, 3]

In [86]: y = x + [4, 5, 6]

In [87]: x
Out[87]: [1, 2, 3]

In [88]: y
Out[88]: [1, 2, 3, 4, 5, 6]
```

การเพิ่มสมาชิกเข้าใหม่ที่ท้ายลิสต์ทำได้โดยใช้เมธอด **append**

```
In [89]: x = [1, 2, 3]

In [90]: x.append(4)

In [91]: x
Out[91]: [1, 2, 3, 4]
```

การ unpack lists คือ การดึงค่าของสมาชิกในลิสต์มาใส่ลงในตัวแปร

```
In [102]: x, y = [True, 1]

In [103]: x
Out[103]: True

In [104]: y
Out[104]: 1
```

2.5.2 Tuples

Tuples คือลิสต์ที่ไม่สามารถแก้ไขค่าได้ (immutable) โอเปอเรชั่นทุกอย่างที่ใช้ได้กับ lists ก็จะสามารถใช้กับ tuples ได้เช่นเดียวกัน แต่การเขียนแจกแจง tuples จะใช้เครื่องหมายวงเล็บ แทนที่จะเป็นเครื่องหมายวงเล็บเหลี่ยมดังเช่นที่ใช้เขียนแจกแจง lists

```
In [105]: aList = [1, 2]

In [106]: aTuple = (1, 2)

In [107]: another_tuple = 3, 4

In [108]: aList[0] = 4

In [109]: aTuple[0] = 4
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-109-fb800bf48292> in <module>
----> 1 aTuple[0] = 4

TypeError: 'tuple' object does not support item assignment

In [110]: try:
     ...:     aTuple[0] = 4
     ...: except TypeError:
     ...:     print("ไม่สามารถแก้ไข tuple ได้")
     ...:
ไม่สามารถแก้ไข tuple ได้
```

2.5.3 Dictionaries

Dictionaries เป็นโครงสร้างข้อมูลพื้นฐานอีกตัวหนึ่งในภาษาไพธอน ซึ่งใช้สำหรับผูกค่า values กับค่า keys เข้าด้วยกัน และทำให้เราสามารถดึงค่า value ที่ผูกไว้กับค่า key ได้อย่างรวดเร็ว

```
In [120]: # dictionaries

In [121]: empty_dict = {}

In [122]: scores = {"Liverpool": 61, "Tottenham": 57, "Man. City": 56, "Chelsea": 50, "Arsenal": 47}

In [123]: chelsea_score = scores["Chelsea"]

In [124]: chelsea_score
Out[124]: 50

In [125]: scores["Fulham"]
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-125-fd64911e1d79> in <module>
----> 1 scores["Fulham"]

KeyError: 'Fulham'

In [126]: # ไม่มี Fulham ใน keys ของ scores ทำให้เกิด Key Error exception

In [127]: try:
     ...:     fulham_score = scores["Fulham"]
     ...: except KeyError:
     ...:     print("no score for Fulham!")
     ...:
no score for Fulham!
```

การตรวจสอบว่ามีค่าคีย์อยู่ใน dictionaries หรือไม่ ทำได้โดยใช้โอเปอเรเตอร์ **in**

```
In [128]: "Fulham" in scores
Out[128]: False

In [129]: "Liverpool" in scores
Out[129]: True
```

อีกวิธีหนึ่งในการดึงค่าที่ผูกกับคีย์ที่กำหนด ก็คือ การใช้เมธอด **get** ซึ่งมีข้อดีคือ ในกรณีที่ค่าคีย์ที่กำหนดไม่มีอยู่ใน dictionaries จะไม่มีเกิด KeyError exception แต่ค่าส่งกลับจากเมธอดจะมีค่าเป็น default value (สามารถกำหนดค่าที่ต้องการได้)

```
In [130]: liverpool_score = scores.get("Liverpool", 0)

In [131]: fulham_score = scores.get("Fulham", 0)

In [132]: everton_score = scores.get("Everton")          # ค่า default = None

In [133]: liverpool_score
Out[133]: 61

In [134]: fulham_score
Out[134]: 0

In [135]: everton_score

In [136]: everton_score is None
Out[136]: True
```

การเพิ่มคู่ของ key-value เข้าไปใน dictionaries ทำได้โดยใช้เครื่องหมายวงเล็บเหลี่ยม

```
In [137]: scores
Out[137]:
{'Liverpool': 61,
 'Tottenham': 57,
 'Man. City': 56,
 'Chelsea': 50,
 'Arsenal': 47}

In [138]: scores["Man United"] = 45

In [139]: scores["Wolves"] = 38

In [140]: scores
Out[140]:
{'Liverpool': 61,
 'Tottenham': 57,
 'Man. City': 56,
 'Chelsea': 50,
 'Arsenal': 47,
 'Man United': 45,
 'Wolves': 38}
```

การดึงค่า keys ทุกตัวจาก dictionaries ทำได้โดยใช้ เมธอด keys()

```
In [141]: scores.keys()
Out[141]: dict_keys(['Liverpool', 'Tottenham', 'Man. City', 'Chelsea', 'Arsenal', 'Man United', 'Wolves'])

In [142]: for team in scores.keys():
     ...:     print(team)
     ...:
Liverpool
Tottenham
Man. City
Chelsea
Arsenal
Man United
Wolves
```

การดึงค่า values ทุกตัวจาก dictionaries ทำได้โดยใช้ เมธอด values()

```
In [143]: scores.values()
Out[143]: dict_values([61, 57, 56, 50, 47, 45, 38])

In [144]: for score in scores.values():
     ...:     print(score)
     ...:
61
57
56
50
47
45
38
```

การดึงค่า items ทั้งหมดจาก dictionaries ทำได้โดยใช้ เมธอด items()

```
In [155]: scores.items()

Out[155]: dict_items([('Liverpool', 61), ('Tottenham', 57), ('Man. City', 56), ('Chelsea', 50), ('Arsenal', 47),
 ('Man United', 45), ('Wolves', 38)])

In [156]: for team, score in scores.items():
     ...:     print(f"{team:12s}  {score}")
     ...:

Liverpool     61
Tottenham     57
Man. City     56
Chelsea       50
Arsenal       47
Man United    45
Wolves        38
```

2.5.4 defaultdict

สมมุติว่า เราได้ทำการวัดส่วนสูงของประชากรกลุ่มหนึ่งจำนวน 100 คน และเราได้เก็บค่าส่วนสูงที่วัดได้ไว้ในลิสต์ ชื่อ heights หากเราต้องการทราบว่ามีประชากรกี่คนสำหรับแต่ละค่าส่วนสูง เราสามารถเขียนโปรแกรมเพื่อนับความถี่ของแต่ละค่าส่วนสูงได้ดังนี้

```
In [163]: height_freq = { }

In [164]: for height in heights:
     ...:     if height in height_freq:
     ...:         height_freq[height] += 1
     ...:     else:
     ...:         height_freq[height] = 1      # height ถูกเพิ่มเข้าในดิกชั่นนารี่เป็นครั้งแรก
```

พิจารณาตัวอย่างโปรแกรมข้างบน จะพบว่า สาเหตุที่ต้องใช้ if-else ข้างใน for-loop นั้น ก็เพื่อป้องกันการเกิด KeyError exception ในกรณีที่เราไม่ใช้ if-else เราสามารถใช้โครงสร้าง try-except แทนได้ ดังโปรแกรมต่อไป

```
In [165]: height_freq = { }

In [166]: for height in heights:
     ...:     try:
     ...:         height_freq[height] += 1
     ...:     except KeyError:
     ...:         height_freq[height] = 1
```

วิธีที่สามของการเขียนโปรแกรมนับความถี่ของค่าส่วนสูงที่เก็บไว้ในลิสต์ คือการใช้เมธอด **get**

```
In [167]: height_freq = { }

In [168]: for height in heights:
     ...:     prev_freq = height_freq.get(height, 0)
     ...:     height_freq[height] = prev_freq + 1
```

วิธีที่สี่ คือการใช้ defaultdict แทน dictionaries

```
In [172]: from collections import defaultdict

In [173]: height_freq = defaultdict(int)            # int() = 0; จะถูกเรียกใช้เมื่อค่าคีย์ที่ใช้ดึงข้อมูลไม่มีอยู่ใน dictionary

In [174]: for height in heights:
     ...:     height_freq[height] += 1
```

ตัวอย่างการใช้งาน defaultdict เพิ่มเติม

ตัวอย่างที่ 1: default factory = list

```
In [179]: dd1 = defaultdict(list)              # list() = []

In [180]: dd1[0].append(5)

In [181]: dd1
Out[181]: defaultdict(list, {0: [5]})

In [182]: dd1[2].append(2)

In [183]: dd1
Out[183]: defaultdict(list, {0: [5], 2: [2]})

In [184]: dd1[2].append(3)

In [185]: dd1
Out[185]: defaultdict(list, {0: [5], 2: [2, 3]})
```

ตัวอย่างที่ 2: default factory = dict

```
In [192]: dd2
Out[192]: defaultdict(dict, {'Liverpool': {'MP': 24, 'W': 19}})

In [193]: dd2 = defaultdict( dict )          # dict() = {}

In [194]: dd2["Liverpool"]["W"] = 19

In [195]: dd2["Liverpool"]["MP"] = 24

In [196]: dd2["Man. City"]
Out[196]: {}

In [197]: dd2
Out[197]: defaultdict(dict, {'Liverpool': {'W': 19, 'MP': 24}, 'Man. City': {}})

In [198]: dd2["Man. City"]["W"] = 18

In [199]: dd2["Man. City"]["MP"] = 24

In [200]: dd2
Out[200]:
defaultdict(dict,
            {'Liverpool': {'W': 19, 'MP': 24},
             'Man. City': {'W': 18, 'MP': 24}})
```

ตัวอย่างที่ 3: default factory คือ lambda function

```
In [207]: dd3 = defaultdict(lambda: [0, 0, 0])

In [208]: dd3[0][2] = 1

In [209]: dd3
Out[209]: defaultdict(<function __main__.<lambda>()>, {0: [0, 0, 1]})

In [210]: dd3[0]
Out[210]: [0, 0, 1]
```

2.5.5 Counter

คลาส Counter ในโมดูล collections ใช้สำหรับนับความถี่ของ items ในลิสต์ โดยเอาท์พุทที่ได้จะเป็นอ็อบเจ็กต์ defaultdict(int) ซึ่งมีแมปจากค่าคีย์ item ไปเป็นความถี่ของ item นั้น ในลิสต์

```
In [222]: fruits = ["banana", "apple", "banana", "grape", "jackfruit", "apple", "apple", "jackfruit", "mango"]

In [223]: fruit_counts = Counter(fruits)

In [224]: # most popular fruit is...

In [225]: fruit_counts.most_common(1)
Out[225]: [('apple', 3)]

In [226]: # top-3

In [227]: fruit_counts.most_common(3)
Out[227]: [('apple', 3), ('banana', 2), ('jackfruit', 2)]

In [228]: # list all fruits and their counts

In [229]: for fruit, count in fruit_counts.items():
     ...:     print(fruit, count)
     ...:
banana 2
apple 3
grape 1
jackfruit 2
mango 1
```

2.5.6 Sets

โครงสร้างข้อมูล **set** ใช้เก็บกลุ่มของ item ที่ไม่ซ้ำกัน การสร้างเซตว่างทำได้โดยการเรียกใช้ฟังก์ชัน set() การสร้างเซตโดยการแจกแจงสมาชิกทำได้โดยการลิสต์รายการสมาชิกคั่นด้วยเครื่องหมายจุลภาคระหว่างเครื่องหมายวงเล็บปีกกา การเพิ่มสมาชิกเข้าไปในเซตทำได้โดยใช้เมธอด add การหาจำนวนสมาชิกในเซตทำได้โดยใช้ฟังก์ชัน in และการตรวจสอบว่า item ที่กำหนดเป็นสมาชิกของเซตหรือไม่ทำได้โดยใช้โอเปอเรเตอร์ in

เรามักใช้ set เมื่อต้องการตรวจสอบว่า item ที่กำหนด เป็นสมาชิกของเซตของข้อมูลหรือไม่ (membership test) เนื่องจากเซตจะสามารถตรวจสอบความเป็นสมาชิกดังกล่าว (ด้วยโอเปอเรเตอร์ in) ได้รวดเร็วกว่าโครงสร้างข้อมูล list มาก การประยุกต์ใช้งาน set ที่พบบ่อย อีกอย่าง ก็คือ การหา unique item ในชุดข้อมูล

ตัวอย่างการสร้างเซต และการเพิ่มสมาชิกเข้าไปในเซต

```
In [243]: even_numbers = {2, 4, 6, 8, 10}

In [244]: empty_set = set()

In [245]: even_numbers = {2, 4, 6, 8, 10}     # create a new set with five elements

In [246]: empty_set = set()                   # create an empty set

In [247]: s = { }                             # Be careful! s is an empty dictionary! not a set!

In [248]: type(s)
Out[248]: dict

In [249]: s = set()

In [250]: s.add(1)

In [251]: s.add(3)

In [252]: s.add(5)

In [253]: len(s)
Out[253]: 3

In [254]: s
Out[254]: {1, 3, 5}

In [255]: 3 in s
Out[255]: True

In [256]: 9 in s
Out[256]: False
```

ตัวอย่างการใช้เซตเพื่อทดสอบความเป็นสมาชิก (membership test) และสร้างรายการ unique items

```
In [274]: bohemian_rhasody = """
     ...: Is this the real life?
     ...: Is this just fantasy?
     ...: Caught in a landslide
     ...: No escape from reality
     ...: Open your eyes
     ...: Look up to the skies and see"""

In [275]: bohemian_wordlist = bohemian_rhasody.split()

In [276]: bohemian_wordset = set( bohemian_wordlist )

In [277]: # 'crazy' มีในเนื้อร้องหรือไม่

In [278]: %timeit 'crazy' in bohemian_wordlist
413 ns ± 9.12 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

In [279]: %timeit 'crazy' in bohemian_wordset
51 ns ± 0.245 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)

In [287]: # จ ำนวนค ำศัพท์   ที่ใช้ในเนื้อร้อง

In [288]: len(bohemian_wordset)
Out[288]: 24

In [289]: # จ ำนวนค ำ   ในเนื้อร้อง

In [290]: len(bohemian_wordlist)
Out[290]: 27
```

```
In [287]: # จำนวนคำศัพท์ที่ใช้ในเนื้อร้อง

In [288]: len(bohemian_wordset)
Out[288]: 24

In [289]: # จำนวนคำ ในเนื้อร้อง

In [290]: len(bohemian_wordlist)
Out[290]: 27
```

2.6 Control Flow

ในภาษาไพธอน เราสามารถเลือกดำเนินการคำสั่งตามเงื่อนไขใด ๆ ได้ โดยใช้ **if:**

```
In [3]: errcode = 3

In [4]: if errcode == 0:
   ...:     message = "everythings look okay."
   ...: elif errcode == 1:
   ...:     message = "something wrong."
   ...: elif errcode >= 2:
   ...:     message = "certainly, there is something wrong."
   ...: else:
   ...:     message = "negative error code!"
   ...:

In [5]: print(message)
certainly, there is something wrong.
```

เราสามารถใช้เขียนโครงสร้าง if-then-else ได้ในบรรทัดเดียว โดยใช้ ternary-if-then-else เช่น

```
In [11]: x = 33

In [12]: is_odd = True if x % 2 == 1 else False

In [13]: is_odd
Out[13]: True
```

การทำซ้ำในภาษาไพธอน ใช้ while-loop และ for-loop ดังตัวอย่างต่อไปนี้

```
In [14]: x = 5

In [15]: while x < 10:
    ...:     print(f"{x} is less than 10")
    ...:     x += 1
    ...:
5 is less than 10
6 is less than 10
7 is less than 10
8 is less than 10
9 is less than 10

In [16]: for x in range(5, 10):
    ...:     print(f"{x} is less than 10")
    ...:
5 is less than 10
6 is less than 10
7 is less than 10
8 is less than 10
9 is less than 10
```

การควบคุมการทำงานภายในลูปและการหยุดการทำงานของลูป สามารถทำได้โดยใช้คำสั่ง continue และคำสั่ง break

```
In [22]: x = 5

In [23]: while x < 10:
    ...:     if x == 7:
    ...:         x += 1
    ...:         continue
    ...:     if x == 9:
    ...:         x += 1
    ...:         break
    ...:     print(x)
    ...:     x += 1
    ...:
5
6
8

In [24]: for x in range(5, 10):
    ...:     if x == 7:
    ...:         continue
    ...:     if x == 9:
    ...:         break
    ...:     print(x)
    ...:
5
6
8
```

2.7 ค่าความจริง (Boolean)

ค่าความจริงในภาษาไพธอน เขียนแทนด้วย True และ False และค่าที่ไม่มีอยู่ (nonexistent values) แทนด้วย None:

```
In [25]: 1 < 2
Out[25]: True

In [26]: True == False
Out[26]: False

In [27]: x = None
```

คีย์เวิร์ด **assert** ใช้ทดสอบว่าเงื่อนไขที่กำหนดมีค่าความจริงเป็นจริงหรือไม่ ถ้าเงื่อนไขเป็นจริง assert จะรีเทิร์นค่า True แต่หากเงื่อนไขที่กำหนดมีค่าความจริงเป็นเท็จก็จะเกิด AssertionError ขึ้นในโปรแกรม

```
In [27]: x = None

In [28]: assert x is None, "to check if x is None, use `is`"

In [29]: assert x is not None, "this will cause an AssertionError exception"
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-29-d43398dcc5ef> in <module>
----> 1 assert x is not None, "this will cause an AssertionError exception"

AssertionError: this will cause an AssertionError exception
```

ฟังก์ชั่น all รับอินพุทเป็นค่า iterable[1] และรีเทิร์น True เมื่อสมาชิกทุกตัวใน iterable มีค่าความจริงเป็นจริง
ฟังก์ชั่น any รับอินพุทเป็นค่า iterable และรีเทิร์น True เมื่อสมาชิกตัวใดตัวหนึ่งใน iterable มีค่าความจริงเป็นจริง

```
In [31]: all([True, 1, -1])
Out[31]: True

In [32]: all([True, 1, -1])
Out[32]: True

In [33]: any([True, 1, -1, {}])
Out[33]: True

In [34]: all([True, 1, -1, {}])
Out[34]: False

In [35]: all([])
Out[35]: True

In [36]: any([])
Out[36]: False
```

2.8 การเรียงลำดับ

Python list มีเมธอด sort ไว้สำหรับจัดเรียงข้อมูลภายในลิสต์แบบ in-place หากต้องการเรียงลำดับข้อมูลในลิสต์โดยไม่เปลี่ยนแปลงค่าของลิสต์เดิมให้ใช้ฟังก์ชัน sorted

```
In [37]: x = [7, 9, -1, 0, 2, 8]

In [38]: y = sorted(x)

In [39]: x
Out[39]: [7, 9, -1, 0, 2, 8]

In [40]: y
Out[40]: [-1, 0, 2, 7, 8, 9]

In [41]: x.sort()

In [42]: x
Out[42]: [-1, 0, 2, 7, 8, 9]
```

ถ้าต้องการเรียงลำดับข้อมูลจากมากไปน้อย ให้กำหนดพารามิเตอร์ reverse=True และหากต้องการกำหนดฟังก์ชั่นสำหรับเปรียบเทียบค่าให้ส่งฟังก์ชั่นที่ต้องการใช้ผ่านพารามิเตอร์ key

[1] iterable คืออ็อบเจ็กต์ที่ implement iterable protocol ซึ่งประกอบด้วย เมธอด __iter__() และ เมธอด __next__() เช่น list และ dict เป็นต้น

```
In [37]: x = [7, 9, -1, 0, 2, 8]

In [38]: y = sorted(x)

In [39]: x
Out[39]: [7, 9, -1, 0, 2, 8]

In [40]: y
Out[40]: [-1, 0, 2, 7, 8, 9]

In [41]: x.sort()

In [42]: x
Out[42]: [-1, 0, 2, 7, 8, 9]

In [43]: x = sorted([7, 9, -1, 0, 2, 8], key=abs, reverse=True)

In [44]: x
Out[44]: [9, 8, 7, 2, -1, 0]

In [45]: x = [7, 9, -1, 0, 2, 8]

In [46]: x.sort(key=abs, reverse=True)

In [47]: x
Out[47]: [9, 8, 7, 2, -1, 0]
```

2.9 List Comprehensions

List comprehensions คือการสร้างลิสต์ใหม่ขึ้นจากลิสต์ตั้งต้น โดยการเลือกเฉพาะสมาชิกบางตัว และ/หรือโดยการแปลงค่าของสมาชิก

ตัวอย่าง เช่น กำหนดลิสต์ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] เราสามารถสร้างลิสต์ของจำนวนคี่จากลิสต์ตั้งต้นนี้ ได้ โดย:

```
odd_numbers = [x for x in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] if x % 2 == 1]
```

ตัวอย่างเพิ่มเติมของ list comprehensions

```
In [48]: even_numbers = [x for x in range(10) if x % 2 == 0]

In [49]: even_numbers
Out[49]: [0, 2, 4, 6, 8]

In [50]: squares = [x * x for x in range(5)]

In [51]: squares
Out[51]: [0, 1, 4, 9, 16]

In [52]: square_dict = {x: x * x for x in range(5)}

In [53]: square_dict
Out[53]: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

In [54]: square_set = {x * x for x in [1, -1]}

In [55]: square_set
Out[55]: {1}

In [56]: square_set = {x * x for x in [1, -1, 1, -1]}

In [57]: square_set
Out[57]: {1}
```

2.10 Object-Oriented Programming

ในภาษาไพธอน เราสามารถสร้างคลาสเพื่อ encapsulate ข้อมูลและฟังก์ชันสำหรับจัดการข้อมูล ได้ด้วยคีย์เวิร์ด class ดังต่อไปนี้

```
In [74]: class Pixel:
    ...:     """Pixel on a screen."""
    ...:     def __init__(self, color=(0,0,0)):
    ...:         """a constructor receiving one input parameter: color."""
    ...:         self.color = color
    ...:     def __repr__(self):
    ...:         """the string representation of a Pixel class instance."""
    ...:         return f"Pixel(color={self.color})"
    ...:
    ...:

In [75]: black = Pixel()

In [76]: white = Pixel((255, 255, 255))

In [77]: red = Pixel((255, 0, 0))

In [78]: print(f"RGB-value of black is {black.color}.")
RGB-value of black is (0, 0, 0).

In [79]: print(f"RGB-value of white is {white.color}.")
RGB-value of white is (255, 255, 255).

In [80]: print(f"RGB-value of red is {red.color}.")
RGB-value of red is (255, 0, 0).
```

2.11 Iterables and Generators

ข้อดีอย่างหนึ่งของการใช้ลิสต์เก็บข้อมูลในโปรแกรมก็คือ เราสามารถดึงค่าของข้อมูลในลิสต์ออกมาได้อย่างรวดเร็วโดยใช้ค่าอินเด็กซ์ อย่างไรก็ตามในกรณีที่ข้อมูลที่ต้องการเก็บในลิสต์มีปริมาณมาก การสร้างลิสต์เพื่อเก็บข้อมูลทั้งหมดในหน่วยความจำเป็นสิ่งที่ไม่สมควรทำ เนื่องจากอาจทำให้โปรแกรมของเราใช้เนื้อที่หน่วยความจำมากจนเกิดการ crash ได้

วิธีการหนึ่งที่ทำได้เพื่อหลีกเลี่ยงปัญหาดังกล่าว ก็คือ การใช้ generators (สร้างจาก ฟังก์ชั่น และ โอเปอเรเตอร์ yield)

```
In [81]: def generate_even_numbers(n):
    ...:     i = 0
    ...:     while i < n:
    ...:         if i % 2 == 0:
    ...:             yield i
    ...:         i += 1
    ...:

In [82]: for i in generate_even_numbers(20):
    ...:     print(f"i: {i}")
    ...:
i: 0
i: 2
i: 4
i: 6
i: 8
i: 10
i: 12
i: 14
i: 16
i: 18
```

อีกรูปแบบหนึ่งของการใช้งานลิสต์ในภาษาไพธอน ก็คือการอ่านค่าของสมาชิกแต่ละตัวในลิสต์พร้อมกับค่าอินเด็กซ์ เราอาจเขียนโปรแกรมโดยการสร้างตัวแปรสำหรับเก็บค่าอินเด็กซ์ และอัพเดตค่าอินเด็กซ์ในแต่ละรอบของการวนซ้ำ หรืออีกวิธีหนึ่งซึ่งเป็นวิถีทางที่โปรแกรมเมอร์ภาษาไพธอนใช้บ่อยกว่า ก็คือ การใช้ฟังก์ชั่น enumerate

```
In [83]: subjects = ["Introduction to Data Science", "Programming Fundamentals II",
    ...: "Survey Project", "Research"]

In [84]: for i in range(len(subjects)):
    ...:     print(f"subject {i} is {subjects[i]}")
    ...:
subject 0 is Introduction to Data Science
subject 1 is Programming Fundamentals II
subject 2 is Survey Project
subject 3 is Research

In [85]: i = 0

In [86]: for subj in subjects:
    ...:     print(f"subject {i} is {subj}")
    ...:
subject 0 is Introduction to Data Science
subject 0 is Programming Fundamentals II
subject 0 is Survey Project
subject 0 is Research

In [87]: for i, name in enumerate(subjects):   # Pythonic way
    ...:     print(f"subject {i} is {name}")
    ...:
subject 0 is Introduction to Data Science
subject 1 is Programming Fundamentals II
subject 2 is Survey Project
subject 3 is Research
```

2.12 การสุ่มค่า

ในการเขียนโปรแกรมทางด้านวิทยาศาสตร์ข้อมูล เรามักจะต้องทำการสร้างตัวเลขแบบสุ่ม (random numbers) เพื่อใช้ในการทดสอบอัลกอริทึม หรือการรันอัลกอริทึม เราสามารถใช้โมดูล random เพื่อสร้างค่าแบบสุ่มขึ้นใช้ในโปรแกรมได้ เมธอดที่ใช้บ่อย ได้แก่ random.seed, random.random, random.randrange, random.shuffle, random.choice, random.sample

```
In [88]: import random

In [89]: random.seed(53)      # ตั้งค่า seed เพื่อให้ได้ผลลัพธ์ที่เหมือนเดิม

In [90]: numbers = [random.random() for _ in range(10)]

In [91]: numbers
Out[91]:
[0.6171414766699522,
 0.8901385532025501,
 0.45648721979605955,
 0.7114047593079196,
 0.9369813033782328,
 0.5205598276906142,
 0.7338401709097778,
 0.7280324327291228,
 0.8581866169652738,
 0.03679243409125621]
```

```
In [100]: random.randrange(10)   # randomly choose from [0, 1, ..., 9]
Out[100]: 8

In [101]: random.randrange(4, 7) # randomly choose from [4, 5, 6]
Out[101]: 6

In [102]: one_to_seven = [1, 2, 3, 4, 5, 6, 7]

In [103]: random.shuffle(one_to_seven)

In [104]: one_to_seven
Out[104]: [6, 2, 1, 4, 5, 3, 7]

In [105]: random.choice(["red pill", "blue pill"])
Out[105]: 'blue pill'

In [106]: lotto = range(100)

In [107]: second_prizes = random.sample(lotto, 2)  # sample 2 out of 100

In [108]: second_prizes
Out[108]: [19, 11]
```

2.13 zip และ Argument Unpacking

สมมุติว่า เรามีลิสต์อ็อบเจ็กต์สองตัว คือ

list1= [1, 2, 3, 4, 5]

list2 = [‘Giant’, ‘Tsuneo’, ‘Shizuka’, ‘Doraemon’, ‘Nobita’]

เราสามารถรวม list1 และ list2 เข้าด้วยกันได้โดยใช้คำสั่ง zip:

```
In [131]: list1 = [1, 2, 3, 4, 5]

In [132]: list2 = ['Gaint', 'Tsuneo', 'Shizuka', 'Doraemon', 'Nobita']

In [133]: [pair for pair in zip(list1, list2)]
Out[133]: [(1, 'Gaint'), (2, 'Tsuneo'), (3, 'Shizuka'), (4, 'Doraemon'), (5, 'Nobita')]
```

ในทางกลับกัน เราสามารถใช้เครื่องหมาย * เพื่อ unpack ลิสต์ ออกได้ ดังตัวอย่างต่อไปนี้

```
In [134]: pairs = [pair for pair in zip(list1, list2)]

In [135]: numbers, characters = zip(*pairs)

In [136]: numbers
Out[136]: (1, 2, 3, 4, 5)

In [137]: characters
Out[137]: ('Gaint', 'Tsuneo', 'Shizuka', 'Doraemon', 'Nobita')

In [138]: numbers, characters = zip((1, 'Gaint'), (2, 'Tsuneo'), (3, 'Shizuka'), (4, 'Doraemon'),
     ...:  (5, 'Nobita'))

In [139]: numbers
Out[139]: (1, 2, 3, 4, 5)

In [140]: characters
Out[140]: ('Gaint', 'Tsuneo', 'Shizuka', 'Doraemon', 'Nobita')
```

2.14 args และ kwargs

การสร้างฟังก์ชั่นที่รับ arguments ใดๆ ทำได้ดังนี้

```
In [141]: def afunc(*args, **kwargs):
     ...:     print("unnamed args:", args)
     ...:     print("keyword args:", kwargs)
     ...:

In [142]: afunc(1, 2, 3, 4, name="doraemon", magic="takecopter")
unnamed args: (1, 2, 3, 4)
keyword args: {'name': 'doraemon', 'magic': 'takecopter'}
```

args และ kwargs ถูกนำไปใช้ในการสร้างฟังก์ชั่นใหม่จากฟังก์ชั่นเดิม เช่น หากต้องการสร้างฟังก์ชั่น f2 ซึ่งมีค่าเป็นสองเท่าของฟังก์ชั่น f1 ก็สามารถทำได้โดยใช้ args และ kwargs เป็นตัวช่วย ดังตัวอย่างโปรแกรมต่อไปนี้

```
In [143]: def doubler(f1):
     ...:     def f2(*args, **kwargs):
     ...:         return 2 * f1(*args, **kwargs)
     ...:     return f2
     ...:

In [144]: import math

In [145]: def circle(radius): return math.pi * radius * radius

In [146]: circle(2)
Out[146]: 12.566370614359172

In [147]: doublesized_circle = doubler(circle)

In [148]: doublesized_circle(2)
Out[148]: 25.132741228718345
```

2.15 Regular Expressions

Regular expression คือ ลำดับของตัวอักขระที่อธิบายรูปแบบของการค้นหา เราใช้ regular expression สำหรับตรวจสอบว่ามีรูปแบบที่ระบุอยู่ในสตริงหรือไม่

ตัวอย่างที่ 1: ตรวจสอบว่าสตริงขึ้นต้นและลงท้ายด้วยคำที่ต้องการหรือไม่ โดยใช้ re.search

```
In [168]: import re

In [169]: txt = "The sky is blue"

In [170]: x = re.search("^The.*blue$", txt)

In [171]: if (x):
     ...:     print("YES! the message starts with 'The' and ends with 'blue'")
     ...: else:
     ...:     print("No match found.")
     ...:
YES! the message starts with 'The' and ends with 'blue'
```

ตัวอย่างที่ 2 การแบ่งสตริงออกเป็นส่วน ตรงจุดที่พบ pattern โดยใช้ re.split

```
In [172]: import re

In [173]: str = "The sky is blue"

In [174]: x = re.split("\s", str)

In [175]: print(x)
['The', 'sky', 'is', 'blue']
```

ตัวอย่างที่ 3 การแทน pattern ที่พบด้วยสตริงที่กำหนด โดยใช้ re.sub

```
In [176]: import re

In [177]: str = "The sky is blue"

In [178]: x = re.sub("\s", "_", str)

In [179]: x
Out[179]: 'The_sky_is_blue'
```

ตัวอย่างที่ 4 การค้นหา pattern ภายในสตริง กรณีที่พบ pattern ภายในสตริง re.search จะรีเทิร์น Match object กลับมา

```
In [194]: import re

In [195]: str = "The sky is blue"

In [196]: x = re.search("is", str)

In [197]: print(x)
<re.Match object; span=(8, 10), match='is'>
```

สำหรับตัวอย่างการใช้งานโมดูล re เพิ่มเติม สามารถหาได้จาก Python Documentation [2] หรือบทความจาก W3Schools [3]

แบบฝึกหัด

1. จงเขียนโปรแกรมนับจำนวนคำศัพท์ ที่ปรากฏในข้อความ โดยโปรแกรมจะต้องอ่านค่าจาก text file และ จะต้องแสดงผลลัพธ์ ดังตัวอย่างต่อไปนี้

   $ python wordcount.py mymessage.txt

   ไฟล์ mymessage.txt มีคำทั้งหมด 350 คำ

   - จำนวนคำ (ตัดคำที่ซ้ำกันออก) คือ 280
   - คำที่ใช้บ่อยที่สุด 5 อันดับแรกคือ is, and, the, python, data

2. จงเขียนโปรแกรมสุ่มเลือกไพ่จำนวน 3 ใบ ออกจากสำรับไพ่มาตรฐาน

เอกสารอ้างอิง

[1] Joel Grus. *Data Science from Scratch*, 2nd Edition, O'Reilly Media, Inc., 2019.

[2] https://docs.python.org/3/library/re.html

[3] https://www.w3schools.com/python/python_regex.asp