Python 的 50+ 練習：資料科學學習手冊

> *使用流程控制管理程式區塊的執行*

數據交點 | 郭耀仁 yaojenkuo@datainpoint.com

這個章節會登場的保留字與函數

- `if` 保留字。
- `else` 保留字。
- `elif` 保留字。
- `for` 保留字。
- `while` 保留字。

這個章節會登場的保留字與函數（續）

- `break` 保留字。
- `continue` 保留字。
- `try` 保留字。
- `except` 保留字。
- `as` 保留字。
- `range()` 函數。

關於流程控制

什麼是流程控制

多數程式語言都會從程式碼的第一列開始按照列（Row-wise）的順序往下讀取並且執行，但是在某些情況下，我們會希望依據特定的條件來決定程式的執行與否、重複次數以及錯誤發生時該如何應對，這時就可以透過流程控制的結構機制來滿足這些情況。

我們將要學習的流程控制

- 條件敘述。
- 迴圈。
- 例外處理。

認識程式區塊

什麼是程式區塊

> *程式區塊（Code block）有時也被稱為複合語句，是將程式組合並產生依附關係的結構，由一個或多個敘述所組成。*

來源：https://en.wikipedia.org/wiki/Block_(programming)

Python 使用四個空白作為縮排（Indentation）標註程式區塊

- 多數程式語言使用大括號 `{}` 來標註程式碼所依附的特定保留字。
- 一段程式碼的依附關係從縮排開始直到第一個未縮排的結束。
- 縮排必須隨著依附保留字的數量而增加。

什麼時候需要用到程式區塊

- 流程控制。
- 定義函數與類別。

決定程式區塊是否被執行的條件敘述

什麼是條件敘述

> *條件敘述是依指定運算的結果為 `False` 或 `True`，來決定是否執行一段程式區塊。*

來源：https://en.wikipedia.org/wiki/Conditional_(computer_programming)

使用「條件」與「縮排」建立條件敘述

- 條件指的是一段能夠被解讀為 `bool` 的敘述。
- 縮排是 Python 用來辨識程式碼依附區塊的結構，要特別留意。

使用 `if` 依據條件決定是否執行程式區塊

```
if 條件:
    # 依附 if 敘述的程式區塊。
    # 當條件為 True 的時候程式區塊才會被執行。
```

使用關係運算符或者邏輯運算符描述條件

- 關係運算符：`==`, `!=`, `>`, `<`, `>=`, `<=`, `in`, `not in`
- 邏輯運算符：`and`, `or`, `not`

In [1]:

```
def return_message_if_positive(x):
    if x > 0:
        return f"{x} is positive."

print(return_message_if_positive(56))
print(return_message_if_positive(-56))
print(return_message_if_positive(0))
```

```
56 is positive.
None
None
```

使用 `if...else...` 依據條件決定執行兩個程式區塊其中的一個

```
if 條件:
    # 依附 if 敘述的程式區塊。
    # 當條件為 True 的時候會被執行。
else:
    # 依附 else 敘述的程式區塊。
    # 當條件為 False 的時候會被執行。
```

In [2]:

```
def return_message_whether_positive_or_not(x):
    if x > 0:
        return f"{x} is positive."
    else:
        return f"{x} is not positive."

print(return_message_whether_positive_or_not(56))
print(return_message_whether_positive_or_not(0))
print(return_message_whether_positive_or_not(-56))
```

```
56 is positive.
0 is not positive.
-56 is not positive.
```

使用 `if...elif...else...` 依據條件決定執行多個程式區塊其中的一個

```
if 條件一:
    # 依附 if 敘述的程式區塊。
    # 當條件一為 True 的時候會被執行。
elif 條件二:
    # 依附 elif 敘述的程式區塊。
    # 當條件一為 False 、條件二為 True 的時候會被執行。
else:
    # 依附 else 敘述的程式區塊。
    # 當條件一、條件二均為 False 的時候會被執行。
```

In [3]:

```python
def return_message_whether_positive_negative_or_neutral(x):
    if x > 0:
        return f"{x} is positive."
    elif x < 0:
        return f"{x} is negative."
    else:
        return f"{x} is neutral."

print(return_message_whether_positive_negative_or_neutral(56))
print(return_message_whether_positive_negative_or_neutral(-56))
print(return_message_whether_positive_negative_or_neutral(0))
```

```
56 is positive.
-56 is negative.
0 is neutral.
```

使用 `if...elif...` 把所有的條件都寫清楚

不一定非要加入 `else`

In [4]:

```python
def return_message_whether_positive_negative_or_neutral(x):
    if x > 0:
        return f"{x} is positive."
    elif x < 0:
        return f"{x} is negative."
    elif x == 0:
        return f"{x} is neutral."

print(return_message_whether_positive_negative_or_neutral(56))
print(return_message_whether_positive_negative_or_neutral(-56))
print(return_message_whether_positive_negative_or_neutral(0))
```

```
56 is positive.
-56 is negative.
0 is neutral.
```

一組條件敘述的結構僅會執行「其中一個」程式區塊

- 如果條件彼此之間**互斥**，寫作條件的先後順序**沒有**影響。
- 如果條件彼此之間**非互斥**，寫作條件的先後順序**有**影響。

以自行定義的 `return_message_whether_positive_negative_or_neutral()` 函數為例

- 我們將條件一 `x > 0` 更改為 `x >= 0` 讓條件一與條件三**非互斥**
- 我們將條件二 `x < 0` 更改為 `x <= 0` 讓條件二與條件三**非互斥**

維持原本寫作條件的先後順序

輸入零使得條件一為 `True`，因為一組條件敘述的結構僅會執行「其中一個」程式區塊的特性，條件三以及它的程式區塊將永遠沒有派上用場的機會。

In [5]:
```
def return_message_whether_positive_negative_or_neutral(x):
    if x >= 0:
        return f"{x} is positive."
    elif x <= 0:
        return f"{x} is negative."
    elif x == 0:
        return f"{x} is neutral."

print(return_message_whether_positive_negative_or_neutral(0))
```

```
0 is positive.
```

調整寫作條件的先後順序

將條件三與條件一的順序互換，這時函數的運作才會跟原本條件彼此之間**互斥**時相同。

In [6]:
```
def return_message_whether_positive_negative_or_neutral(x):
    if x == 0:
        return f"{x} is neutral."
    elif x <= 0:
        return f"{x} is negative."
    elif x >= 0:
        return f"{x} is positive."

print(return_message_whether_positive_negative_or_neutral(0))
```

```
0 is neutral.
```

以 Fizz buzz 為例

> *從 1 數到 100，碰到 3 的倍數改為 Fizz、碰到 5 的倍數改為 Buzz，碰到 15 的倍數改為 Fizz Buzz，其餘情況不改動。*

來源：https://en.wikipedia.org/wiki/Fizz_buzz

Fizz buzz 值得注意的地方

條件彼此之間**非**互斥（15 是 3 與 5 的公倍數），寫作條件的先後順序**有**影響。

使用 `if...elif...` 定義 `fizz_buzz()` 函數

In [7]:

```
def fizz_buzz(x):
    if x % 3 != 0 and x % 5 != 0 and x % 15 != 0:
        return x
    elif x % 15 == 0:
        return "Fizz Buzz"
    elif x % 3 == 0:
        return "Fizz"
    elif x % 5 == 0:
        return "Buzz"

print(fizz_buzz(2))
print(fizz_buzz(3))
print(fizz_buzz(5))
print(fizz_buzz(15))
```

```
2
Fizz
Buzz
Fizz Buzz
```

使用 `if...elif...else...` 定義 `fizz_buzz()` 函數

In [8]:

```
def fizz_buzz(x):
    if x % 15 == 0:
        return "Fizz Buzz"
    elif x % 3 == 0:
        return "Fizz"
    elif x % 5 == 0:
        return "Buzz"
    else:
        return x

print(fizz_buzz(2))
print(fizz_buzz(3))
print(fizz_buzz(5))
print(fizz_buzz(15))
```

```
2
Fizz
Buzz
Fizz Buzz
```

假如在寫作條件敘述時不想要去考慮條件的先後順序

那就要記得把條件描述為**互斥**。

In [9]:

```
def fizz_buzz(x):
    if x % 3 == 0 and x % 15 != 0:
        return "Fizz"
    elif x % 5 == 0 and x % 15 != 0:
        return "Buzz"
    elif x % 15 == 0:
        return "Fizz Buzz"
    else:
        return x

print(fizz_buzz(2))
print(fizz_buzz(3))
print(fizz_buzz(5))
print(fizz_buzz(15))
```

```
2
Fizz
Buzz
Fizz Buzz
```

讓程式區塊被重複執行的迴圈

什麼是迴圈

> *迴圈是流程控制的其中一種技巧，可以讓寫作一次的程式區塊被重複執行，常見的應用是重複執行直到條件不成立時或走訪可迭代類別中的所有元素。*

來源：https://en.wikipedia.org/wiki/Control_flow#Loops

迴圈的三個要素

1. 起始。
2. 終止。
3. 如何從起始到終止。

兩種常見的迴圈

1. `while` 迴圈：重複執行程式區塊直到條件為 `False` 的時候。
2. `for` 迴圈：走訪可迭代類別中的所有元素。

使用 `while` 依據條件決定是否重複執行程式區塊

```
while 條件:
    # 依附 while 敘述的程式區塊。
    # 當條件為 True 的時候程式區塊會被重複執行。
    # 當條件為 False 的時候停止執行程式區塊。
```

使用關係運算符或者邏輯運算符描述條件

- 關係運算符：`==`, `!=`, `>`, `<`, `>=`, `<=`, `in`, `not in`
- 邏輯運算符：`and`, `or`, `not`

如何寫作一個 `while` 迴圈

- 在迴圈程式區塊之前定義一個物件設定起始值。
- 設計條件讓程式區塊重複執行的次數符合我們的需求。
- 記得在程式區塊中更新物件的值。

如何寫作一個 `while` 迴圈：印出 5 次 `"Hello, world!"`

透過 pythontutor.com 拆解程式執行的每個步驟。

In [10]:
```
number_of_prints = 0
while number_of_prints < 5:
    print("Hello, world!")
    number_of_prints = number_of_prints + 1
```

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

在程式區塊中更新物件的值更常會使用複合運算符（Compound operators）

- `integer += 1` 等同於 `integer = integer + 1`
- `integer -= 1` 等同於 `integer = integer - 1`
- `integer *= 1` 等同於 `integer = integer * 1`
- `integer /= 1` 等同於 `integer = integer / 1`
- …等。

如何寫作一個 `while` 迴圈：印出小於 10 的奇數

透過 pythontutor.com 拆解程式執行的每個步驟。

In [11]:

```
odd = 1
while odd < 10:
    print(odd)
    odd += 2 # odd = odd + 2
```

```
1
3
5
7
9
```

如何寫作一個 `while` 迴圈：從週一印到週五

透過 pythontutor.com 拆解程式執行的每個步驟。

In [12]:

```
weekdays = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
index = 0
while index < len(weekdays):
    print(weekdays[index])
    index += 1
```

```
Monday
Tuesday
Wednesday
Thursday
Friday
```

使用 `for` 走訪可迭代類別（Iterables）中的所有元素

```
for 元素 in 可迭代類別:
    # 依附 for 敘述的程式區塊。
    # 當可迭代類別還沒有走訪完的時候程式區塊會被重複執行。
    # 當可迭代類別走訪完的時候停止執行程式區塊。
```

什麼是可迭代類別

具有一次回傳其中一個資料值特性的類別、輸入到內建函數 `iter()` 不會產生錯誤的類別，都屬於可迭代類別（Iterables），常見的有 `str` 與資料結構。

- 資料類別：`str`
- 資料結構類別：`list`、`tuple`、`dict`、`set`

In [13]:
```
luke = "Luke Skywalker"
primes = [2, 3, 5, 7, 11]
iter(luke)
iter(primes)
```

Out[13]:
```
<list_iterator at 0x7fb8977799a0>
```

什麼是不可迭代的類別

任何輸入到內建函數 `iter()` 會產生錯誤的類別都是不可迭代類別，像是 `int`、`float` 與 `bool` 等。

In [14]:
```
i_am_int = 5566
try:
    iter(i_am_int)
except TypeError as error_message:
    print(error_message)
```

```
'int' object is not iterable
```

In [15]:
```
i_am_float = 5566.0
try:
    iter(i_am_float)
except TypeError as error_message:
    print(error_message)
```

```
'float' object is not iterable
```

In [16]:
```
i_am_bool = False
try:
    iter(i_am_bool)
except TypeError as error_message:
    print(error_message)
```

```
'bool' object is not iterable
```

如何寫作一個 `for` 迴圈

- 建立一個可迭代類別。
- 可迭代類別如果是數列，可透過內建函數 `range()` 建立。

`range()` 函數有三個參數可以設定數列內容

1. `start` 數列的起始整數，即第 0 個整數（包含），預設值為 0。
2. `stop` 數列的終止整數，即第 -1 個整數（排除）。
3. `step` 數列的公差，預設值為 1。

In [17]:
```
help(range)
```

```
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |
 |  Return an object that produces a sequence of integers from start (inclusive)
 |  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2, ..., j-1.
 |  start defaults to 0, and stop is omitted!  range(4) produces 0, 1, 2, 3.
 |  These are exactly the valid indices for a list of 4 elements.
 |  When step is given, it specifies the increment (or decrement).
 |
 |  Methods defined here:
 |
 |  __bool__(self, /)
 |      self != 0
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
```

```
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __reduce__(...)
|      Helper for pickle.
|
|  __repr__(self, /)
|      Return repr(self).
```

```
 |
 |  __reversed__(...)
 |      Return a reverse iterator.
 |
 |  count(...)
 |      rangeobject.count(value) -> integer -- return number of occurrences of v
alue
 |
 |  index(...)
 |      rangeobject.index(value) -> integer -- return index of value.
 |      Raise ValueError if the value is not present.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  start
 |
 |  step
 |
 |  stop
```

如何寫作一個 `for` 迴圈：印出 5 次 `"Hello, world!"`

透過 pythontutor.com 拆解程式執行的每個步驟。

In [18]:
```
for element in range(0, 5, 1):
    print("Hello, world!")
```

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

（沒有什麼用的冷知識）如果程式區塊中並沒有用到可迭代類別中的元素

可以將元素命名為 `_` 特別點明。

In [19]:
```
for _ in range(0, 5, 1):
    print("Hello, world!")
```

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

如何寫作一個 `for` 迴圈：印出小於 10 的奇數

透過 pythontutor.com 拆解程式執行的每個步驟。

In [20]:
```
for odd in range(1, 10, 2):
    print(odd)
```

```
1
3
5
7
9
```

如何寫作一個 `for` 迴圈：從週一印到週五

透過 pythontutor.com 拆解程式執行的每個步驟。

In [21]:

```
weekdays = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
for weekday in weekdays:
    print(weekday)
```

```
Monday
Tuesday
Wednesday
Thursday
Friday
```

如何抉擇使用哪種迴圈，`for` 迴圈或 `while` 迴圈

- 先思考問題是否能夠建立可迭代類別？
- 如果可以，代表程式區塊被重複執行的次數**已知**，選擇 `for` 迴圈。
- 如果不可以，代表程式區塊被重複執行的次數**未知**，選擇 `while` 迴圈。

重複執行的次數未知：輾轉相除法

In [22]:

```
def divide_reiteratively(x, divisor):
    while x > 0:
        remainder = x % divisor
        quotient = x // divisor
        print(f"{x} / {divisor} = {quotient} ... {remainder}")
        x //= divisor
```

In [23]:

```
divide_reiteratively(5, 2)
print("")
divide_reiteratively(56, 2)
```

```
5 / 2 = 2 ... 1
2 / 2 = 1 ... 0
1 / 2 = 0 ... 1

56 / 2 = 28 ... 0
28 / 2 = 14 ... 0
14 / 2 = 7 ... 0
7 / 2 = 3 ... 1
3 / 2 = 1 ... 1
1 / 2 = 0 ... 1
```

常見的迴圈應用

- 走訪 `str` 或資料結構。
- 可迭代類別的加總、乘積與計數。
- 合併資料成為 `str`、`list` 或者 `dict`。

走訪 `str` 或資料結構

1. 走訪 `str`、`list`、`tuple`、`set`
2. 走訪 `dict`

如何走訪 `str`、`list`、`tuple`、`set`

In [24]:

```
def iterate_str_list_tuple_set(an_iterable):
    for element in an_iterable:
        print(element)

luke = "Luke Skywalker"
weekdays_list = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
weekdays_tuple = tuple(weekdays_list)
weekdays_set = set(weekdays_list)
```

In [25]:
```
iterate_str_list_tuple_set(luke) # iterate over a str
```

```
L
u
k
e

S
k
y
w
a
l
k
e
r
```

In [26]:
```
iterate_str_list_tuple_set(weekdays_list)  # iterate over a list
```

```
Monday
Tuesday
Wednesday
Thursday
Friday
```

In [27]:
```
iterate_str_list_tuple_set(weekdays_tuple) # iterate over a tuple
```

```
Monday
Tuesday
Wednesday
Thursday
Friday
```

In [28]:
```
iterate_str_list_tuple_set(weekdays_set)   # iterate over a set
```

```
Friday
Thursday
Tuesday
Monday
Wednesday
```

如何走訪 `dict`

善用三個 `dict` 方法：

1. `dict.keys()`
2. `dict.values()`
3. `dict.items()`

In [29]:
```
the_shawshank_redemption = {
    'title': 'The Shawshank Redemption',
    'year': 1994,
    'rating': 9.3,
    'director': 'Frank Darabont'
}
type(the_shawshank_redemption)
```

Out[29]:
```
dict
```

預設走訪 `dict` 的「鍵」

In [30]:
```
for key in the_shawshank_redemption:
    print(key)
```

```
title
year
rating
director
```

In [31]:
```
for k in the_shawshank_redemption.keys():
    print(k)
```

```
title
year
rating
director
```

指定走訪 `dict` 的「值」

In [32]:
```
for value in the_shawshank_redemption.values():
    print(value)
```

```
The Shawshank Redemption
1994
9.3
Frank Darabont
```

同時走訪 `dict` 的「鍵」與「值」

In [33]:

```
dict_items = the_shawshank_redemption.items()
print(dict_items)
for key, value in dict_items:
    print(f"{key}: {value}")
```

```
dict_items([('title', 'The Shawshank Redemption'), ('year', 1994), ('rating', 9.
3), ('director', 'Frank Darabont')])
title: The Shawshank Redemption
year: 1994
rating: 9.3
director: Frank Darabont
```

可迭代類別的加總、乘積與計數

- 在迴圈程式區塊之前定義一個物件設定起始值。
- 在程式區塊中更新物件的值。

透過 pythontutor.com 拆解程式執行的每個步驟。

In [34]:

```
summation = 0
product = 1
count = 0
primes = [2, 3, 5, 7, 11]
for prime in primes:
    summation += prime
    product *= prime
    count += 1
print(summation)
print(product)
print(count)
```

```
28
2310
5
```

可迭代類別的加總與計數

善用內建函數 `sum()` 以及 `len()` 就可以得知加總與計數。

In [35]:
```
print(sum(primes))
print(len(primes))
```

```
28
5
```

合併資料成為 `str`、`list` 或者 `dict`

- 運用 `+` 運算符連接元素成為 `str`
- 運用 `+` 運算符連接 lists
- 運用 `list.append()` 方法合併元素成為 `list`
- 運用 `dict[key]=value` 合併元素成為 `dict`

運用 `+` 運算符連接元素成為 `str`

透過 pythontutor.com 拆解程式執行的每個步驟。

In [36]:
```
vowels = ["a", "e", "i", "o", "u", "A", "E", "I", "O", "U"]
vowels_str = str() # an empty str
for vowel in vowels:
    vowels_str += vowel
print(vowels_str)
```

```
aeiouAEIOU
```

運用 `+` 運算符連接 lists

透過 pythontutor.com 拆解程式執行的每個步驟。

In [37]:
```
vowels = [["a", "e", "i", "o", "u"], ["A", "E", "I", "O", "U"]]
flat_vowels = list()
for vowel in vowels:
    flat_vowels += vowel
flat_vowels
```

Out[37]:
```
['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']
```

運用 `list.append()` 方法合併元素成為 `list`

透過 pythontutor.com 拆解程式執行的每個步驟。

In [38]:
```
vowels = [["a", "e", "i", "o", "u"], ["A", "E", "I", "O", "U"]]
flat_vowels = list()
for v_list in vowels:
    for v in v_list:
        flat_vowels.append(v)
flat_vowels
```

Out[38]:
```
['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']
```

運用 `dict[key]=value` 合併元素成為 `dict`

透過 pythontutor.com 拆解程式執行的每個步驟。

In [39]:

```
days_of_week = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
days_of_week_dict = dict()
for day in days_of_week:
    day_upper = day.upper() # upper-case
    day_abbreviation = day_upper[:3] # abbreviation
    days_of_week_dict[day_abbreviation] = day
days_of_week_dict
```

Out[39]:

```
{'SUN': 'Sunday',
 'MON': 'Monday',
 'TUE': 'Tuesday',
 'WED': 'Wednesday',
 'THU': 'Thursday',
 'FRI': 'Friday',
 'SAT': 'Saturday'}
```

以兩個保留字調整迴圈的重複執行次數

1. `break` 保留字可以提早結束。
2. `continue` 保留字可以略過某些執行次數。

遇到星期四提早結束

透過 pythontutor.com 拆解程式執行的每個步驟。

In [40]:
```
days_of_week = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
for day in days_of_week:
    if day == "Thursday":
        break
    print(day)
```

```
Sunday
Monday
Tuesday
Wednesday
```

略過週末

透過 pythontutor.com 拆解程式執行的每個步驟。

In [41]:

```
days_of_week = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
for day in days_of_week:
    if day in {"Sunday", "Saturday"}:
        continue
    print(day)
```

```
Monday
Tuesday
Wednesday
Thursday
Friday
```

決定程式區塊出錯時如何應對的例外處理

什麼是例外處理

> *程式出現錯誤是極為常見的情況，一般碰到錯誤發生時，Python 會發起例外（Raise exception）並且中斷程式的執行，假如我們希望在錯誤發生的情況下「不要」中斷程式的執行，這樣的技巧就稱為例外處理（Exception handling）。*

來源：https://docs.python.org/3/tutorial/errors.html

常見的錯誤種類有三種

1. 語法錯誤（Syntax errors）。
2. 語意錯誤（Semantic errors）。
3. 執行錯誤（Runtime errors）。

語法錯誤：對 Python 而言無效的敘述

例如不使用大括號來標註程式區塊。

In [42]:
```
def add(x, y){
    return x + y
}
```

```
  File "<ipython-input-42-e462a405c867>", line 1
    def add(x, y){
                  ^
SyntaxError: invalid syntax
```

語法錯誤：對 Python 而言無效的敘述（續）

例如 `for` 迴圈沒有冒號 `:`

In [ ]:

```
for i in range(10)
    print(i)
```

語意錯誤

- 程式能夠順利執行但是輸出的結果與預期不相符。
- 例外處理無用武之地，只能慢慢找出邏輯的錯誤所在。

執行錯誤：例外處理主要應對的錯誤種類

- 命名錯誤（`NameError`）。
- 類別錯誤（`TypeError`）。
- 分母為零錯誤（`ZeroDivisionError`）。
- 索引值錯誤（`IndexError`）。
- …等。

命名錯誤（`NameError`）

例如使用沒有定義的函數。

In [ ]:
```
function_which_is_not_defined(5566)
```

類別錯誤（`TypeError`）

例如對文字應用減號。

In [ ]:
```
"Luke Skywalker" - "Luke"
```

分母為零錯誤（`ZeroDivisionError`）

In [ ]:
```
5566/0
```

索引值錯誤（`IndexError`）

例如 `list` 的長度不如預期。

In [ ]:

```
primes=[2, 3, 5, 7, 11]
primes[5]
```

如何進行例外處理

利用 `try...except` 敘述把可能會產生錯誤的程式放在附屬於 `try` 保留字的程式區塊，錯誤發生時的應對程式碼放在附屬於 `except` 保留字的程式區塊。

```
try:
    # 可能產生錯誤的程式區塊。
except 執行錯誤種類:
    # 指定的執行錯誤產生時的應對程式區塊。
except:
    # 執行錯誤產生時的應對程式區塊。
```

利用 `as` 保留字將錯誤訊息記錄起來

```
try:
    # 可能產生錯誤的程式區塊。
except 執行錯誤種類 as error_message:
    # 指定的執行錯誤產生時的應對程式區塊。
    # 錯誤訊息可用物件 error_message 參照。
except:
    # 執行錯誤產生時的應對程式區塊。
```

命名錯誤（`NameError`）：例外處理

例如使用沒有定義的函數。

In [ ]:

```
try:
    function_which_is_not_defined(5566)
except NameError as error_message:
    print(f"NameError occurred and the error message is: {error_message}")
```

類別錯誤（`TypeError`）：例外處理

例如對文字應用減號。

In [ ]:

```
try:
    "Luke Skywalker" - "Luke"
except TypeError as error_message:
    print(f"TypeError occurred and the error message is: {error_message}")
```

分母為零錯誤（ `ZeroDivisionError` ）

In [ ]:

```
try:
    5566/0
except ZeroDivisionError as error_message:
    print(f"ZeroDivisionError occurred and the error message is: {error_message}")
```

索引值錯誤（`IndexError`）

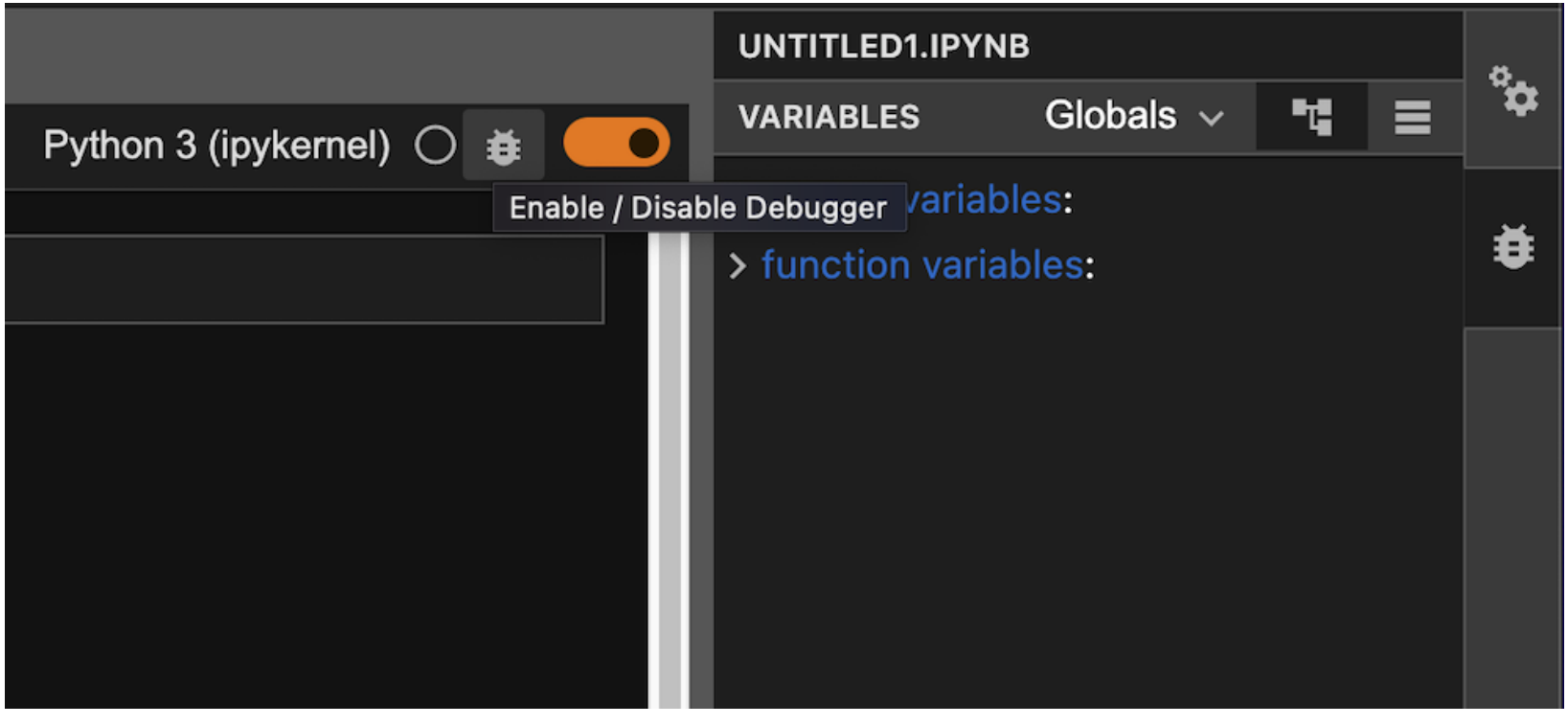
例如 `list` 的長度不如預期。

In [ ]:

```
primes=[2, 3, 5, 7, 11]
try:
    primes[5]
except IndexError as error_message:
    print(f"IndexError occurred and the error message is: {error_message}")
```

如何輔助學習流程控制

- 將流程控制的程式碼貼至程式碼視覺化工具網站協助理解
  https://www.pythontutor.com/visualize.html#mode=edit
- 點選右上角的 Enable / Disable Debugger 觀察 JupyterLab 的物件清單。



重點統整

- 流程控制包含：條件敘述、迴圈與例外處理。
- 條件敘述中，如果條件彼此之間互斥，寫作條件的先後順序沒有影響；如果條件彼此之間非互斥，寫作條件的先後順序有影響。
- 兩種常見的迴圈：`while` 迴圈與 `for` 迴圈。

重點統整（續）

- 常見的迴圈應用：
    - 走訪 `str` 或資料結構。
    - 可迭代類別的加總乘積與計數。
    - 合併資料。
- 運用例外處理應對執行錯誤。