2020/10/15 林隆鴻@政大數理資訊社

Python 是甚麼?

1. 直譯式，物件導向的程式語言。
2. 語法簡潔。
3. 1989 年由 Guido van Rossum 創造。
4. 最新版本: 3.9.0 (10/5 released)

資料型態

- Numeric type(數值型態)
  - 整數(Int)
  - 浮點數(Float)
  - 布林值(Boolean)
- Text Sequence Type(字串)
  - 字串(string)
- Sequence Type(序列型態)：串列(List)、元組(Tuple)
- Mapping Type(對映型態)：字典(Dict)
- Set Type(集合型態)：集合(Set)

整數:

In [1]:

```
a = 5
type(a) # print(type(a)) also works
```

Out[1]:

```
int
```

`type()` 是一個內建的 `function` (函數)。
其功能是回傳 `parameter` (參數)的資料型態。

In [2]:

```
a = 5
type(a)
```

Out[2]:

```
int
```

浮點數:

In [3]:

```
a = 5.3
b = 9.

type(a)
type(b)
```

Out[3]:

```
float
```

In [4]:

```
int(a + b)
```

Out[4]:

```
14
```

布林值:

In [5]:

```
a = True
type(a)
```

Out[5]:

```
bool
```

In [6]:

```
b = False
a + b
```

Out[6]:

```
1
```

布林值可以相加? 在 Python 裡:

> `False` 代表數字 `0`。
> `True` 代表不是 `0` 的所有數字。Python 預設為 `1` 。

In [7]:

```
a = True
b = False
# b - a
```

Builtin Functions for data types:

In [8]:

```
float(5)
```

Out[8]:

```
5.0
```

In [9]:

```
bool(5)
```

Out[9]:

```
True
```

In [10]:

```
int(True)
```

Out[10]:

```
1
```

基本數值運算(算術運算子):

`+, -, *, **, /, //, %`

In [11]:

```
5 - 2
```

Out[11]:

```
3
```

In [12]:

```
5 * 2
```

Out[12]:

```
10
```

In [13]:

```
5 ** 2
```

Out[13]:

```
25
```

In [14]:

```
5 / 2
```

Out[14]:

```
2.5
```

In [15]:

```
5 // 2 # // 代表取商。
```

Out[15]:

```
2
```

In [16]:

```
5 % 2 # 原來 % 代表取餘數。
```

Out[16]:

```
1
```

In [17]:

```
a = 17 // 3
a
```

Out[17]:

```
5
```

In [18]:

```
a = a + 3 * 2
a
```

Out[18]:

```
11
```

Why?
Python 裡的 = 代表賦值運算子 (Assignment Operator)。

規則是: 將右邊的算式運算完後指派給左邊。*

字串("" and ''):

In [19]:

```
s = "I love python."
print(s)
```

```
I love python.
```

In [20]:

```
# '' also works
s = 'I love python.'
print(s)
```

```
I love python.
```

In [21]:

```
s = 'I love python's icon!'
```

```
  File "<ipython-input-21-c5b1fdf910dd>", line 1
    s = 'I love python's icon!'
                        ^
SyntaxError: invalid syntax
```

如果我們就是想打 ' 呢?
使用 逸出字元(Escape character) :

In [22]:

```
s = 'I love python\'s icon!'
print(s)
```

```
I love python's icon!
```

讀入檔案路徑:

In [23]:

```
directory = 'C:\Users\User\'
```

```
  File "<ipython-input-23-7667dfb56b3f>", line 1
    directory = 'C:\Users\User\'
                                ^
SyntaxError: EOL while scanning string literal
```

In [24]:

```
directory = r'C:\Users\User'
print(directory)
```

```
C:\Users\User
```

如果想要印出多行的字串呢? (''')

In [25]:

```
s1 = 'I love
python'
print(s1)
```

```
  File "<ipython-input-25-6e508420b358>", line 1
    s1 = 'I love
                ^
SyntaxError: EOL while scanning string literal
```

In [26]:

```
s2 = '''
This is a paragraph.
I love python.
'''
```

In [27]:

```
print(s2)
```

```
This is a paragraph.
I love python.
```

計算 string 的長度:

In [28]:

```
len(s)
```

Out[28]:

```
21
```

字串連接(+)

In [29]:

```
s3 = 'I ' + 'love ' + 'python' + '!'
print(s3)
```

```
I love python!
```

直覺來說，這樣的作法很合理。
但事實上，是 python 給我們這個方便:
在其他語言，需要用到 operator overloading 的技術*。
以 C++ 為例子:

```
class AddString {

public:
    char s1[25], s2[25];
    AddString(char str1[], char str2[])
    {
        strcpy(this->s1, str1);
        strcpy(this->s2, str2);
    }

    void operator+()
    {
        cout << "\nConcatenation: " << strcat(s1, s2);
    }
};
```

同理，如果將 * operator，套用在 string 上:

In [30]:

```
s3 = "I love python!"
print(s3 * 4)
```

```
I love python!I love python!I love python!I love python!
```

字串分割(.split()):

.split() 是 string 這個 class (類別) 裡的一個 method (方法)。

在下面的程式碼，我們創造一個 string 的 Object (物件)。
s 是這個物件的名稱。
可以使用 isinstance() 做確認。

In [31]:

```
s = 'I love python!'
```

In [32]:

```
isinstance(s, str) # s 的確是 string 的一個 instance。
```

Out[32]:

```
True
```

In [33]:

```
s.split(' ')
```

Out[33]:

```
['I', 'love', 'python!']
```

更多字串相關方法(method):

In [34]:

```
s = 'python'
s.upper()
```

Out[34]:

```
'PYTHON'
```

In [35]:

```
s = 'I LOVE CODING'
s.lower()
```

Out[35]:

```
'i love coding'
```

In [36]:

```
s = '   Today is a good day.   '
s.strip()
```

Out[36]:

```
'Today is a good day.'
```

格式化字串(str.format()):

In [37]:

```
greeting = "Hello, my name is Kevin Durant from USA."
print(greeting)
```

```
Hello, my name is Kevin Durant from USA.
```

In [38]:

```
greeting2 = "Hello, my name is Lionel Messi from Argentina."
print(greeting2)
```

```
Hello, my name is Lionel Messi from Argentina.
```

好像有一點麻煩，這時候可以考慮將字串格式化:

In [39]:

```
first_name = "Kevin"
last_name = "Durant"
nation = "USA"

greeting = "Hello, my name is {} {} from {}.".format(first_name, l
print(greeting)
```

```
Hello, my name is Kevin Durant from USA.
```

優點:

- Code 看起相對乾淨。
- 將你要打出的內容存為變數，以後要修改就容易了。

缺點:

- 如果字串太長，閱讀起來仍很吃力。

更好的方法: fstirng

In [40]:

```
greeting = f"Hello, my name is {first_name} {last_name}."
print(greeting)
```

```
Hello, my name is Kevin Durant.
```

也可以加入不同方法:

In [41]:

```
greeting = f"Hello, my name is {first_name.upper()} {last_name.upp
print(greeting)
```

```
Hello, my name is KEVIN DURANT.
```

題目練習:

在一個荒島上，住著 123 隻猴子，島上一共有 4567 根香蕉。
猴王決定每隻猴子一天可以吃一根香蕉，且每隻猴子每天一定要吃香蕉。 你可以用

1. 島上的香蕉足夠吃幾天?
2. 到數量不夠的那天會少幾根香蕉?

References:

1. 關於運算子與運算元:

   ```
   a = a + 3 * 2
   ```

   在這個算式裡，右邊的 a, 3, 2 稱為operand (運算元)。
   這三個運算元藉由 + 與 * 兩個 operator (運算子)來連接。
   而由運算元與運算子結合者，稱為 expression (表達式)。 舉例來說: 3 * 2 是
   2 是另一個 expression。
2. 關於 Operator Overloading:

> Different operators have different implementations depending on their
> Operator overloading is generally defined by a programming language
> both. (from wikipedia)

簡單來說，一個 `operator` 可以有自己定義的功能。