Python速成班 - 02

上週課程勘誤

- 動態語言：不用明確宣告型態
  - Ex：Python , JavaScript
- 靜態語言：必須明確宣告型態
  - Ex：C , C++ , Java
- 強型別：程式不會自動轉換型別
  - Ex：Python , Java
- 弱型別：會自動轉換型別
  - Ex：JavaScript , C , C++

內建資料結構（list, tuple, dict, set）

list 串列

- 有序的資料集合（跟平常排隊一樣）
- 可以動態增減元素
- 可以用索引取得元素
- 可以修改元素
- 用 `[ ]` 或 `list()` 表示

Initialize List

- List 從 0 開始數
- 可以一次數多個，Ex：`a[0:2]` 可以同時印出`[1,2]`

- 可以新增 (append) / 刪除 (remove) 元素
    - 不是用索引指定欲更動的數值，而是直接寫數值來指定

```
a = [1, 2, 3, 4, 5]
print(a[0]) # 取得元素: 1
print(a[0:2]) # 取得子串列: [1, 2]
a.append(6) # 增加元素: [1, 2, 3, 4, 5, 6]
a.remove(3) # 刪除元素: [1, 2, 4, 5, 6]
```

Tuple 元組

- 有序的資料集合
- 不可以動態增減元素
- 可以用索引取得元素
- **不可以修改元素**
- 用 `( )` 或 `tuple()` 表示

Initialize Tuple

```
a = (1, 2, 3, 4, 5)
print(a[0]) # 取得元素: 1
print(a[0:2]) # 取得子元組: (1, 2)
```

dict 字典

- 有序的資料集合
- 可以動態增減元素
- 可以用索引取得元素（索引可以是**任意型態**）
- 可以修改元素
- 用 `{ }` 或 `dict()` 表示

Initialize dict

- 尋找不存在的值會出錯
- 可以指定要新增的 key 跟 value 內容
- 刪除透過指定特定的 key 來刪

```
a = {'a': 1, 'b': 2, 'c': 3}
print(a['a']) # 取得元素: 1
print(a['d']) # keyError: 'd'
a['d'] = 4 # 增加元素
del a['a'] # 刪除元素
print(a) # {'b': 2, 'c': 3, 'd': 4}
```

set 集合

- 有序的資料集合
- 內容不會重複
- 可以動態增減元素
- 用 `{ }` 或 `set()` 表示

Initialize set

- 較常用在儲存**不重複的東西**
  - 只會保存一份相同的元素（重複的元素不會保留）
- 刪除：指定要刪除的值

```
a = {1, 2, 3, 4, 5}
a.add(6) # 新增元素
a.remove(3) # 刪除元素 （不可以刪除不存在的元素）
```

其他補充

- list：中括號
- tuple：小括號
- set：大括號

```
a = [1, 2, 3, 4, 5] #list
print(len(a)) #5
b = (1, 2, 3, 4, 5) #tuple
print(len(b)) #5
c = {1, 2, 3, 4, 5} #set
print(len(c)) #5
```

函式（function）

- 用來封裝程式碼的一種結構
  - 類似手機裡面的App，不想每次要算1+1都重寫，而是有一個**寫好的邏輯**可以重複使用
- 可以重複使用
- 可以**傳入參數**
- 可以有**回傳值**
  - 將某些東西（parameter）丟進去，可以吐東西（return value）出來
  - 丟進去（parameter）的東西可以很多，但吐（return value）只能吐一個

```
def function_name(parameter1, parameter2)
  # do something
  return value
# e.g.
def add(a, b)
  return a + b
```

Parameter1, parameter2 稱為參數，可以有預設值

```
# e.g.
def add(a, b = 0)
  return a + b
print(add(1)) # 1
```

順序：必要參數 > 預設參數

```
# e.g.
def add(a = 0, b)
  return a + b
print(add(1)) # SyntaxError: non-default argument follows default argument
```

可變參數

```
# e.g.
def add(*args):
  result = 0
  for i in args:
    result += i
  return result
print(add(1, 2, 3, 4, 5)) # 15
```

變數範圍（scope）

- function 和變數結合，會發生一些奇怪的事…
  - **變數範圍**會協助我們處理這件事情
- 變數的作用範圍
- 分為**全域變數（global variable）**和**區域變數（local variable）**
  - 區域變數存在於函式中，或迴圈中

全域變數

```
a = 1 # 全域變數
def foo():
  a = 2 # 區域變數
def bar():
  global a = 2 # 全域變數
print(a) # 1
foo() #雖然foo有賦值給a，但foo更改的a是區域變數，不會更改到全域變數的a
print(a) # 1
bar()
print(a) # 2
```

Big-O Notation

💡 **它是一個衡量演算法速度的數量級的 Upper Bound**

- 計算演算法的複雜度
- 用來比較演算法的好壞
- 用來估計演算法的執行時間
- 用來估計演算法的記憶體使用量
- 程式中，log 都是以 2 為底

```
----------------最快-------------------
O(1): 常數時間（取 index）
O(log n)：對數時間（二分搜尋）
O(n)：線性時間（迴圈）
O(n log n)：線性對數時間（快速排序）
O(n^2)：平方時間（兩層迴圈）
O(2^n)：指數時間（遞迴）
----------------最慢-------------------
註：2^n 比 n^2 慢
```

迴圈與串列

新增一個範圍內的元素到 List 裡面

```
#透過變數當作後面的陣列list或範圍range中的所有元素
[expression for item in iterable]
```

```
#用一個變數i代表陣列中所有的資料
c = [i for i in range(10)]
print(c) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
c = [i for i in range(10) if i % 2 == 0]
print(c) # [0, 2, 4, 6, 8]
```

進階內建資料結構

```
from collections import namedtuple, defaultdict, OrderedDict, Counter
```

namedtuple - 命名元組

- 用來建立自訂的 tuple

```
Point = namedtuple('Point', ['x', 'y'])
p = Point(1, 2)
```

defaultdict - 有預設值的 dict

- 如果 key 不存在，defaultdict 會自動建立 value
    - Ex：原本的 dict 只有 1,2,3，若要讀取 4 會出錯，但 defaultdict 可以協助新增

```
d = defaultdict(list)
d['a'].append(1)
d['a'].append(2)
d['b'].append(3)
d['c'] = 4
print(d) # defaultdict(<class 'list'>, {'a': [1, 2], 'b': [3], 'c': 4})
```

OrderedDict - 有順序的 dict

- 在 Python3.7 之後，會記錄 key 的插入順序

```
od1 = OrderedDict()
od1['a'] = 1
od1['b'] = 2

od2 = OrderedDict()
od2['b'] = 2
od2['a'] = 1
print(od1 == od2) # False
```

- 如果 OrderedDict 放入 key 的順序不同，那兩者就會不一樣

```
d1 = dict()
d1['a'] = 1
d1['b'] = 2

d2 = dict()
d2['b'] = 2
```

```
d2['a'] = 1
print(d1 == d2) # True
```

- 一般的 dict 並不會在乎順序

Counter

- 用來計算元素出現的次數

```
c = Counter('hello world')
print(c) # Counter({'l': 3, 'o': 2, 'h': 1, 'e': 1, ' ': 1, 'w': 1, 'r': 1, 'd': 1})
```

Lambda

在 Amazon 的 AWS 上叫做 Lambda function（匿名函式），但跟這裡的 Lambda 沒關係！

- 可以一行解決加法功能
- 若想重複使用一個 Lambda function，可以用一個變數（add）去存他

```
lambda arguments: expression
# e.g.
add = lambda x, y: x + y
print(add(1, 2)) # 3
```

Map

- 將 iterable 的元素逐一傳入 function，並將結果傳回

```
map(function, iterable, ...)
```

```
a = [1, 2, 3, 4, 5]
b = map(lambda x: x * 2, a)
print(list(b)) # [2, 4, 6, 8, 10]
```

Filter

- 將 iterable 的元素逐一傳入 function，並將結果為 True 的元素傳回

```
filter(function, iterable)
```

```
a = [1, 2, 3, 4, 5]
b = filter(lambda x: x % 2 == 0, a)
print(list(b)) # [2, 4]
```

Reduce

- 將 iterable 的元素逐一傳入 function，並將結果傳回 function，直到 iterable 結束

```
reduce(function, iterable[, initializer])
```

```
from functools import reduce
a = [1, 2, 3, 4, 5]
b = reduce(lambda x, y: x + y, a)
print(b) # 15
```

Partial

- 暫存函式的參數，並回傳一個新的函式，新的函式可以接受剩餘的參數

```
from functools import partial
```

```
def add(x, y):
return x + y
add_1 = partial(add, 1)
print(add_1(2)) # 3
```

Zip

- 將多個 iterable 的元素逐一組合成 tuple

```
zip(iterable,...)
```

- 範例：將兩個List組合成Tuple

```
a=[1,2,3]
b=[4,5,6]
c=list(zip(a, b))
print(c)#[(1,4),(2,5),(3,6)]
```

- **解壓縮**
  - 將組合起來的元素再分開
  - Ex：$[(a_1, b_1), (a_2, b_2), (a_3, b_3)]$ 變成 $(a_1, a_2, a_3), (b_1, b_2, b_3)$

```
a,b=zip(*c)
print(a,b)
#(1,2,3)(4,5,6)
```

Enumerate

- 把一個List改成Tuple的形式，並加上index
- Tuple的index會從0開始算

```
enumerate(iterable, start=0)
```

```
a = [1, 2, 3]
b = list(enumerate(a))
print(b) # [(0, 1), (1, 2), (2, 3)]
```

Reversed

- 將 iterable 的元素逆序

```
reversed(seq)
```

```
a = [1, 2, 3]
b = list(reversed(a))
print(b) # [3, 2, 1]
c = a[::-1]
print(c) # [3, 2, 1]
```

Sorted

- 將 iterable 的元素排序
  - 預設從小排到大
  - Key：一個 Lambda Function，依照該 Key 做排序

```
sorted(iterable, key=None, reverse=False)
```

```
a = [3, 2, 1]
b = sorted(a)
print(b) # [1, 2, 3]
```

閉包(closure)

- 一個函式裡面包含另一個函式
  - Python 可以在一個 Function 裡面包另一個 Function

```
def outer():
  def inner():
    pass
```

- 一個函式裡面包含另一個函式，並且另一個函式會使用到外面函式的變數
- nonlocal：設定非區域變數，outer 函式可以影響到 inner 函式的變數

```
def outer():
  x = 1
  def inner():
    nonlocal x
    x += 1
```

```
    print(x)
  return inner
```