一、 基礎 Python

內建函數

對於一個程式語言，最基礎的就是輸出了。以下是 Python 的輸出函數：

```
print("你好，世界！")
print(1, 2, 3) # 會輸出 1 2 3
```

print 函數中可以傳入兩個**可選參數**，分別為 sep，以及 end，分別代表多個值輸出時所使用的分隔符號（預設為一格 space），以及結尾要自動加上的字串（預設為換行符號 \n）。

```
print(1,2,3,4,5, sep=", ")
print("a", end="")
print("b", end="")
print("\n", end="")
```

在上方的 code 中，不難預測輸出結果，結果為

```
1, 2, 3, 4, 5
ab
```

輸入同樣也是一個程式語言重要的一部分。Python 使用 input 函數從**標準輸入**讀取資料。以下是 input 的用法：

```
line = input()
print("你剛剛輸入了：")
print(line)
```

上方的程式中，使用者一整行的輸入，將由 input() 讀取，並回傳至左方宣告的變數 line。

變數

變數用來儲存一個數值、字串、或是其他資料。在 python 中，我們使用 「=」來宣告變數。以下是一些簡單的範例：

```
a = 10
b = "hello"
a = foo()  # 假設 foo 函數存在
```

上方的程式中，第一行將 10 賦予給 a，將 "hello" 賦予給 b。比較複雜的第三行，則是將 foo() 的回傳值賦予給 a，而原本 a 的值就被覆蓋掉了。

型態（Type）

以下的表格是一些常見的變數型態：

型態	說明	Pass by ref/val
int/float	整數、小數	val
str	字串	val
tuple/list/dict	數組、串列、鍵-值表	val
各種 Class	其他 Class	ref

其中，pass by value 指的是在傳（函數呼叫、賦值）的時候，將變數的值複製；Pass by reference 則是傳變數的參照，即修改變數的內容會影響到原來的變數。這個部分在 numpy slice 中會用到。

數值變數

下方的程式會宣告一些數值變數：

```
a = 10
b = 20
```

將 10 賦予 a，將 20 賦予 b。

數值變數可以做基礎的四則運算，以及取餘數。詳細的用法請參考 10-basic-syntax/05-numeric-type.py。

我們再看一段程式：

```
a = 10
a = a + 20
```

一開始將 a 宣告為 10，之後 a = a + 20。此時會先執行右邊，計算 (a(10) + 20) = (30)，並將 30 重新賦予 a。所以執行結束後，a = 30。此外，a = a + X 可以縮寫為 a += X，加減乘除 mod 都是同樣的作法。

字串（String）

顧名思義，字串就是用來處理文字內容的。Python 中的字串可以直接用 + 連接。以下是一些簡單的範例：

```
a = "this is an apple."
b = 'this is a banana.'
c = a + " " + b
```

上方的程式中，c 將 a 跟空白跟 b 連接起來，所以 c 的內容會是 "this is an apple. this is a banana."。

複雜的字串拼接與格式化可以透過 string format 的技巧來處理，以下是範例：

```
x = 10.0001
a = f"x: {x}"
b = f"x: {x: .2f}"
```

上面的程式我們可以看到 a 就會變成 "x: 10.0001" 而 b 就會變成 "x: 10.00"，這樣就可以將數字轉換成我們想要的格式。

字串有多種表達方式，如下例：

```
str1 = "asdf"
str2 = 'adsf'
str3 = '''this is multiple line string
the second line
'''
str4 = """this is another multi line string
with 中文字 in here
"""
```

可以看到用三個單引號或雙引號可以製造出包含多行字的字串。

串列（List）

我們可以用一個變數來包含多個變數，這在 Python 裡面叫做串列（List）。表達的方式如下：

```
a = [1, 2, 3, 4, 5]
b = []
```

可以看到範例中的 a 包含了 1, 2, 3, 4, 5，而 b 什麼都沒有包含。

串列在宣告後，還是可以追加內容，如下例所示：

```
a = []
b = 10
a.append(b)
a.append("hello")
```

這段程式執行後可以看到 a 包含了 10 與 "hello"，這些值都是在 a 被宣告後再追加到串列的最後面的。也可看到一個串列中可以同時儲存不同的資料，不限型態。

使用 [n] 來取得第 n 項元素，n 從 0 開始數，如下例：

```
a = [0, "hello", 3.1415926, 'a']
print(a[3])
print(a[0])
```

所以這個範例將輸出 "a" 與 "0" 兩行字。取得的元素除了可以存取也可以修改，如下例所示：

```
a = [0, "hello", 3.1415926, 'a']
a[0] = "Blah"
print(a[0])
```

這個程式將會輸出 "Blah" ，因為串列中的第一項已經被修改成 "Blah" 了。

要注意到 list 是可以裝 list 的，如下所示：

```
a = [1, 2, 3 [0 ,1 ,2], 4, "a", ["hello", "kitty"]]
print(a[3])     # 輸出: [0, 1, 2]
print(a[6][1])  # 輸出: kitty
```

另外要注意串列的指定是不做複製的，我們用範例解釋這個現象：

```
a = [0, "hello", 3.14, 'a']
b = a
b[2] = 1
print(a[2])
```

這個範例的輸出將**不會**是 3.14 ，而會是 1 ，因為 a 與 b 是同一個串列，而不是不同的串列。我們稱這個概念叫做取參考（Reference）。

我們可以將串列中的某一段複製出來，這個動作叫做切片（slice），我們可以用 [n:m] 的語法來表示要取用 n ~ m-1 區間，語法如範例所示：

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(a[1:4])
print(a[:4])
print(a[1:])
print(a[4:-2])
```

這支程式會輸出：

```
[2, 3, 4]
[1, 2, 3, 4]
[2, 3, 4, 5, 6, 7, 8, 9, 10]
[5, 6, 7, 8]
```

其中比較令人難懂的應該是 n 與 m 皆允許出現負數，用來表示相對於串列結尾的位置，比如範例中最後一行的輸出便是從第 4 項取到倒數第 2 項（包含倒數第 2 項）。

接著是反轉一個串列，在 Python 裡面這可以很方便的使用 reverse 這個函數做到，如範例所示：

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
reversed(a) # 回傳一個 iterator
list(reversed(a)) # 輸出: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

可以注意到 reverse 回傳的結果並不是 list，若要取得反轉後的 list 則需要轉型，reverse 回傳值的型別會在後面的章節提及。

數組（Tuple）

可以看作是不可增長也不可修改值的 list，這在函數需要回傳多個值的時候非常好用，tuple 基本的語法如下：

```
a = (1, 'a')
b = "hello", "kitty" # 等號左右與return可以忽略括號
print(b[0]) # 輸出: hello
print(b[1]) # 輸出: kitty
```

要注意的是，因為 tuple 不可修改，所以 [] 只有存取是合法的，寫入則會導致錯誤。

字典（Dictionary）

字典是一種鍵值對應表，鍵值對應表可以想成是一個容器，就好像串列，可以儲存元素，但不同之處在於，字典沒有順序的概念，取用元素時使用「鍵」（key）來取代「第幾個」。聽起來有點難懂，看範例會比較好解釋：

```
a = {
    "hello": 1,
    "wut": "meow",
    1: 2,
}
print(a["hello"])   # 輸出: 1
print(a[1])         # 輸出: 2
print(a['wut'])     # 輸出: meow
```

使用{}來建立字典，使用[]來取得裡面的元素，只不過存取字典時，[]內不只可以用數字存取，其他型別 (比如字串) 也都能使用。

編輯字典也與編輯 list 相似，可以看以下範例：

```
a = {
    "hello": 1,
    "wut": "meow",
}
a["hello"] = a["wut"]
a["wut"] = 1
a[10] = 100
print(a) # 輸出: {'hello': 'meow', 'wut': 1, 10: 100}
```

字典也如 list 一樣允許嵌套結構，如範例

```
a = {
    "hello": 1,
    "cat": "meow",
    1: {
        "wolf": "狗狗",
        "cat": "Something different"
    },
}
print(a[1]['cat'])  # 輸出: Something different
print(a[1])         # 輸出: {"wolf": "狗狗","cat": "Something
differenct"}
print(a["cat"])     # 輸出: meow
```

字典中有一些常用的函數這裡拿出來介紹。.key()與.values()可以用來取得所有的鍵與所有的值。.items()則會取得 (鍵,值) 的數組。這些函數的回傳值都是 iterator，所以如果想要作為 list 存取則需要額外轉型。可以看範例：

```
    a = {'hello': 'meow', 'wut': 1, 10: 100}
    print(list(a.keys()))        # 輸出: ['hello', 'wut', 10]
    print(list(a.values()))      # 輸出: ['meow', 1, 100]
    print(list(a.items()))       # 輸出: [('hello', 'meow'),
('wut', 1), (10, 100)]
```

流程控制

if

if 是判斷並執行指定敘述，他大概要寫成這個樣子：

```
    if condition:
        expressions... # 若條件成立時
    elif another condition:
        expressions... # 若第一個條件不成立但第二個成立
    else: # 條件都不成立
        expressions...
```

裡面的 expression 可以寫任何你想要的敘述，包括變數的宣告、修改，甚至是再寫更多的 if。

condition 則要寫入判斷的條件，數值的判斷有 > 、 < 、 >= 、 <= ，用 == 來表示相等，!= 來表示不相等。此外可以用 and 來表示兩個條件要同時發生，用 or 來表示其中一個發生就好，而 not 表示條件發生時則不成立，反之亦然。來看以下的範例：

```
    a = 100
    b = 10
    if a > b:
        print("a > b")
    else:
        print("a < b")

    if a % b == 0 and b < a:
        print("Logic correct")

    """
    程式輸出:
    a > b
    Logic correct
    """
```

for

for 是一個迴圈的語法，可以將一段程式以指定的方式重複執行，常用於迭代 (iterate) 串列或字典等容器，或是生成有規則的數列。

在講 for 之前，我們應該講講迭代器 (iterator)，迭代器是一個只可以不斷取得下一項的值的型別，我們可以在 for 的語法中用 in 這個關鍵字來遍歷迭代器內每一個可以迭代的值。如下所示：

```
a = {
    "hello": "kitty",
    "world": "didn't say hello back",
    100: 10
}

for n in a.keys():
    print(n)

"""
程式輸出:
hello
world
100
"""
```

我們可以看到 a.keys() 的結果沒有被變換成 list ，而是直接將他用來迭代， n 會依序變成 a 裡面每一個鍵。

接著如果想要產生一個順序的數列，我們可以使用 range(n) 來產生一個迭代器，讓我們可以從 0 迭代到 n-1，見範例：

```
for n in range(10):
    print(n)
"""
程式輸出:
0
1
2
3
4
5
6
7
8
9
"""
```

迭代串列也是可以的，如範例：

```
a = [1, 2, 3, 5, 8, 13, 21]
for n in a:
    print(n)


"""
程式輸出:
1
2
3
5
8
13
21
"""
```

如果要迭代串列，但又要同時得到目前是第幾項，可以使用 enumerate 的語法，如範例所示：

```
a = [1, 2, 3, 5, 8, 13, 21]
for i, val in enumerate(a):
    print(i, val)


"""
程式輸出:
0 1
1 2
2 3
3 5
4 8
5 13
6 21
"""
```

就可以分別得到現在是第幾項，還有該項的內容。

while

while 也是迴圈的語法，他在條件成立時就繼續執行裡面的程式，反之離開。範例：

```
n = 13
while n != 1:
    print(n)
    if n % 2 == 0:
        n = n // 2
    else:
        n = 3 * n + 1
"""
程式輸出:
13
40
20
10
5
16
8
4
2
"""
```

while 的語法很適合在不知道迴圈確切會跑幾次，只知道結束條件的時候使用。 要注意條件是繼續的條件不是結束的條件！

函數（Function）

函數可以封裝並複用你的程式片段。函數借用了數學上函數的概念，與數學中的函數一樣有參數(argument)與回傳值(return value)。但有少許的不同，比如有可選參數、多重回傳值（透過回傳數組）以及允許函數沒有回傳值。使用 def 關鍵字來宣告函數，如範例所示：

```
def foo(a, b):
    print("foo:", a, b)
    return (a + b) * (a - b)

x = 1
y = 2
a = foo(1, 2)
b = foo(x, y)
foo(x, y)
"""
程式輸出:
foo: 1 2
foo: 1 2
foo: 1 2
"""
```

可以發現就算忽略函數的回傳值也不會出現錯誤。

我們可以透過回傳數組來達成回傳多個值，見範例：

```
def my_func(a, b):
    return a + b, a - b

x, y = my_func(1, 2) # x = 3, y = -1
ans = my_func(1, 2)  # ans = (3, -1)
```

如果想要設計函數擁有某些可選用的參數，可以預先設定這些參數的預設值，這樣這個參數沒有被傳進來的時候就會被設為預設值，否則就使用傳進來的值。見範例：

```
def foo(a, b=10, c=20):
    print(f"a: {a}, b: {b}, c:{c}")
foo(1, c=200) # a b c 代 1, 10, 200
foo(1, 2, 3)  # a b c 代 1, 2, 3
foo(b=10)     # 發生錯誤! 需要填 a
```

函數之中可以呼叫其他函數，此外呼叫自己也是合法的，我們稱這種情況叫做遞迴，我們看以下的範例：

```
def fac(x):
    if x <= 1:
        return 1
    return x * fac(x - 1)
```

範例中實作了一個數學上的階層函數，在某些時候，使用遞迴形式的定義會使函數意外的好寫，但也可能導致麻煩。

錯誤處理

舉個例子，假設這支程式會輸出輸入數字的平方：

```
a = int(input())
print(a ** 2)
```

然後我輸入了"Apple"，那我們應該會得到錯誤並且程式會強制退出，如果我們已經預期到可能有錯，並打算忽略或排解這個錯誤的

話，我們可以使用錯誤處理來做到這件事。請見以下範例：

```
reinput = True
a = None
while reinput:
    try:
        reinput = False
        a = int(input())
    except ValueError:
        reinput = True

print(a ** 2)
```

在範例中，我們捕捉了 ValueError 這個轉型錯誤，在發生這個錯誤時，使使用者重新輸入一個新的值，直到使用者輸入數字，才輸出這個數字的平方。使用 try-except 的語法可以很好的在預期的錯誤發生時排除或忽略錯誤。

常見的錯誤型別如表：

- Exception　　會比對到所有錯誤
- ValueError　　例如 int() 中包含無法解析的字串
- KeyboardInterrupt　　被使用者打斷
- KeyError　　陣列索引不存在 等

如果想要忽略錯誤，可以使用 pass 來不做任何事。

二、 物件導向

物件導向的核心思想是將程式的每個功能包裝在一個物件裡面。這種設計程式的方式已經被廣泛應用在各種程式語言，Python 也不例外。

簡報中已經有詳細的基礎使用的範例。由於這門課主要為應用為主，較少需要撰寫 class，因此在這僅補充 operator overloading 的範例：

```python
class Foo:
    def __init__(self, a=10, b=10):
        self.a = a
        self.b = b

    def __add__(self, b):
        return Foo(self.a + b.a, self.b + b.b)

    def __sub__(self, b):
        return Foo(self.a - b.a, self.b - b.b)

    def __mul__(self, b):
        return self.a * b.a + self.b * b.b

    def __str__(self):
        return f"({self.a}, {self.b})"


v1 = Foo(1, 2)
v2 = Foo(2, -1)

print("v1 + v2", v1 + v2)
print("v1 - v2", v1 - v2)
print("v1 * v2", v1 * v2)

"""
程式輸出：
v1 + v2 (3, 1)
v1 - v2 (-1, 3)
v1 * v2 0
"""
```

簡單來說，我們定義了 v1 這個 class 在使用加法、減法、乘法所呼叫的函數。當我們使用 v1 + v2 時，事實上就等同於使用 v1.__add__(v2) 。

三、 套件與專案管理

Python 中使用 PIP 來管理第三方的套件。以下是簡單的使用方式：

```
pip install <package-name>
pip uninstall <package-name>
```

很顯然，第一行指的是安裝套件，第二行是移除套件。以安套件 `tqdm` 為例，我們只需要在終端機（cmd、powershell）輸入 `pip install tqdm`

這樣就完成安裝了。