1.15 函數運用 function (額外教材)

你將學習到

1. 函數定義
2. 輸入參數
3. 區域變數
4. 返回值
5. 呼叫其他函數

函數 (function) 是一系列重複動作的別名

放入T恤
加上洗衣粉
開動洗衣機
晾乾

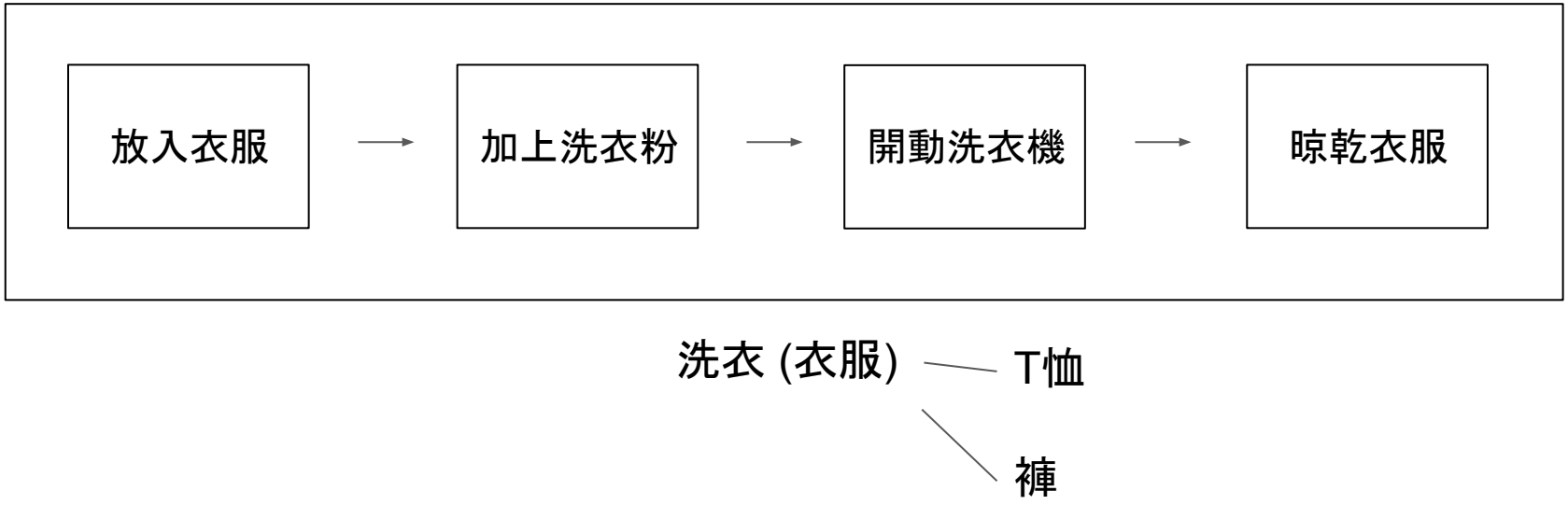
放入褲
加上洗衣粉
開動洗衣機
晾乾

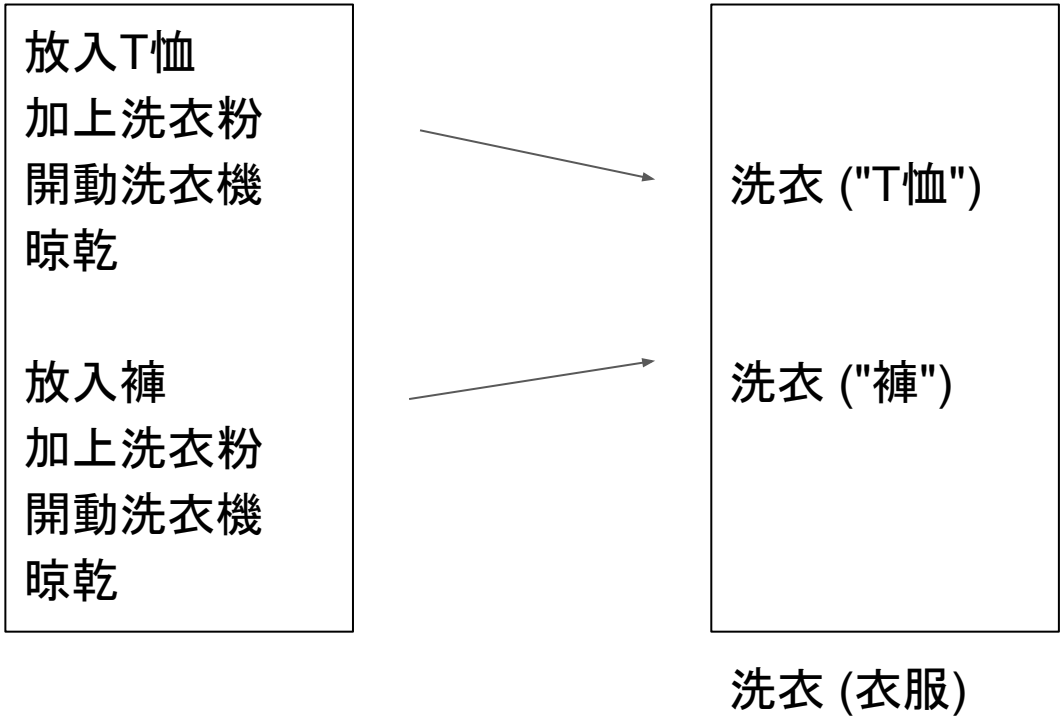
放入T恤和放入褲的動作其實很類似，後面 3 個動作更是完全重覆了，有無辦法簡化程式？

我們可以把重複的 4 個動作改一個新的名字叫「洗衣」，之後只需告訴電腦執行「洗衣」就可。

如同變數是數值的別名，函數則是一系列程式動作的別名。

我們先把兩段程式碼的不同之處 – “T恤” 和 “褲” 抽出來，統一給它們改一個別名 (變數) “衣服”，這樣兩段程式碼便變得完全一樣。





我們可以把這四個重複使用的動作打包，然後給它一個名字叫「洗衣」，它接受一個參數(變數) 叫「衣服」，每次只需分別放入參數 “T恤” 和 “褲” 就可。

這樣程式便從 8 行簡化為只有 2 行，這就是函數的概念。

函數定義

函數內的語句需以Tab或空格起始。注意函數內的動作並不會立刻執行，直到被呼叫後才會執行 (見下頁例子)。

```
def 函數名(參數):
    動作1
    動作2
    … 更多動作
```

函數定義 (無參數)

```
def doWork():
    print( "doWork!" )
    # 注意函數定義後並不會立刻執行，直到被呼叫才會執行


doWork()        # 呼叫 doWork 函數，輸出 "doWork!"
doWork()        # 再次呼叫 doWork 函數，輸出 "doWork!"
```

PYTHON 大數據採集課程

函數定義 (一個參數)

```
def doWork(a):    # a 是一個變數，名稱任改，你可以改用其他變數名
    print(a)

doWork("T恤")   # 把 "T恤" 傳給函數，函數內的變數 a 會等於 "T恤"，並 print("T恤")
doWork("褲")    # 把 "褲" 傳給函數，函數內的變數 a 會等於 "褲"，並 print("褲")
doWork(5)       # 函數參數無分型別，這裡 a 會等於 5，並 print(5)
```

函數定義 (多個參數)

```
def doWork(a, b, c):            # 多個變數用逗號分隔
    print(a+b+c)


doWork(1, 2, 3)                 # 多個參數用逗號分隔


# 參數會按位置順序傳給函數,
因此函數內的變數 a=1, b=2, c=3, print(6)
```

參數預設值

```
def doWork(a, b=2):

    print(a+b)



doWork(1)           # 函數內的變數 a=1, b=2, print(3)

doWork(1, 4)        # 函數內的變數 a=1, b=4, print(5)
```

PYTHON 大數據採集課程

函數區域變數 Local variable

真實程式會用到大量函數，但大家寫函數時改的變數名一般都是 a, b, c… 好容易撞名。因此 Python 設計是每個函數和主程式變數都是互相獨立。即使用了相同變數名，也不會互相覆寫。這樣我們定義函數變數時就不用擔心會和其他函數或主程式內的變數撞名。

```
def addAndPrint(a, b):
    c = a+b   # 函數內定義了變數 c 儲存 a+b 值，a, b, c 的數值只在函數範圍內生效
    print(c)  # 輸出 c 值


c = 10                  # 定義變數 c 儲存 10
addAndPrint(1, 2)       # 函數內 a=1, b=2, c=3, print(3)
print(c)                # print(10)，注意函數內的 c=3 不會影響主程式內的 c
```

函數返回值 return

前面提到每個函數和主程式之間變數都是互相獨立的，函數如果要把計算後的數值傳回給主程式就需要用到 return。

```
def add(a, b):
    c = a+b          # 運算 a+b，儲存到 c
    return c         # 返回 c 的數值


d = add(1, 2)        # 主程式需要定義變數 d 來接收從函數返回的數值 3
```

函數內呼叫其他函數

```
def add(c, d):
        return c+d

def addAndPrint(a, b):
        print( add(a, b) )

addAndPrint(1, 2)
```

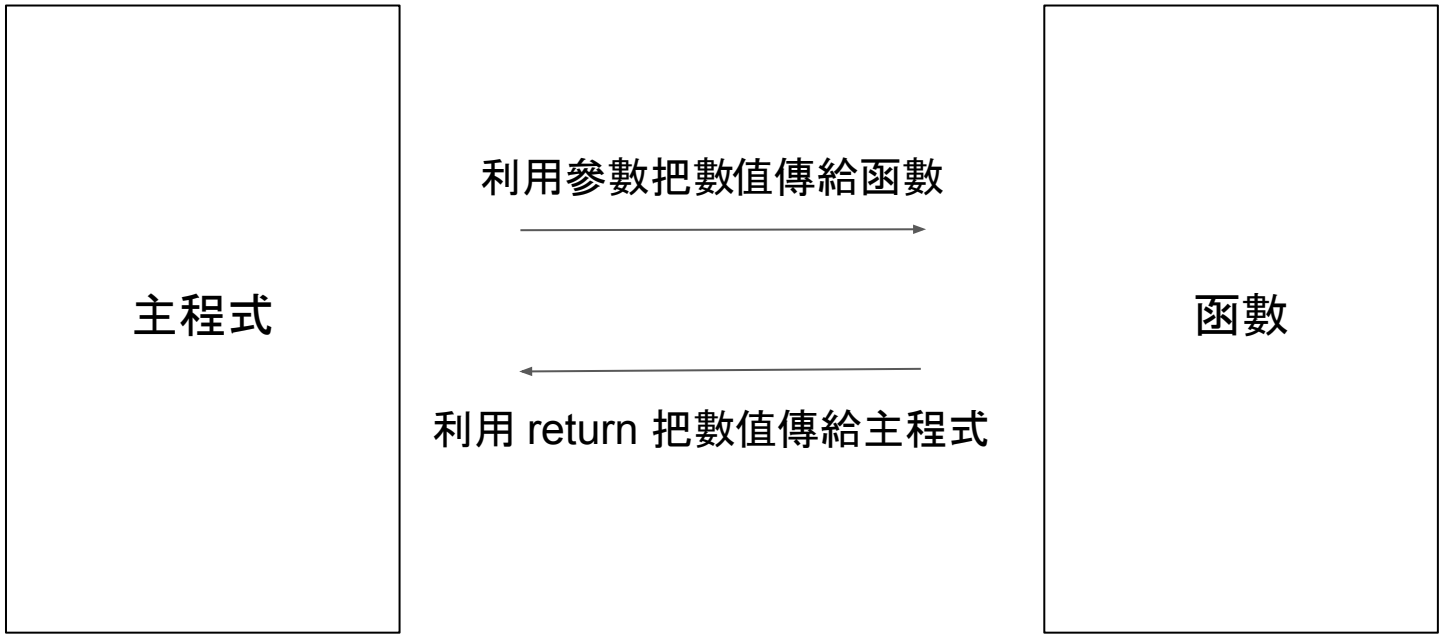
為免混淆, add 函數用了 c, d 做變數名。其實用 a, b 也可以的, 不會和 addAndPrint 內的 a, b 撞的。每個函數內的變數名都是獨立的。

add 函數內 c=a=1, d=b=2, print(3)

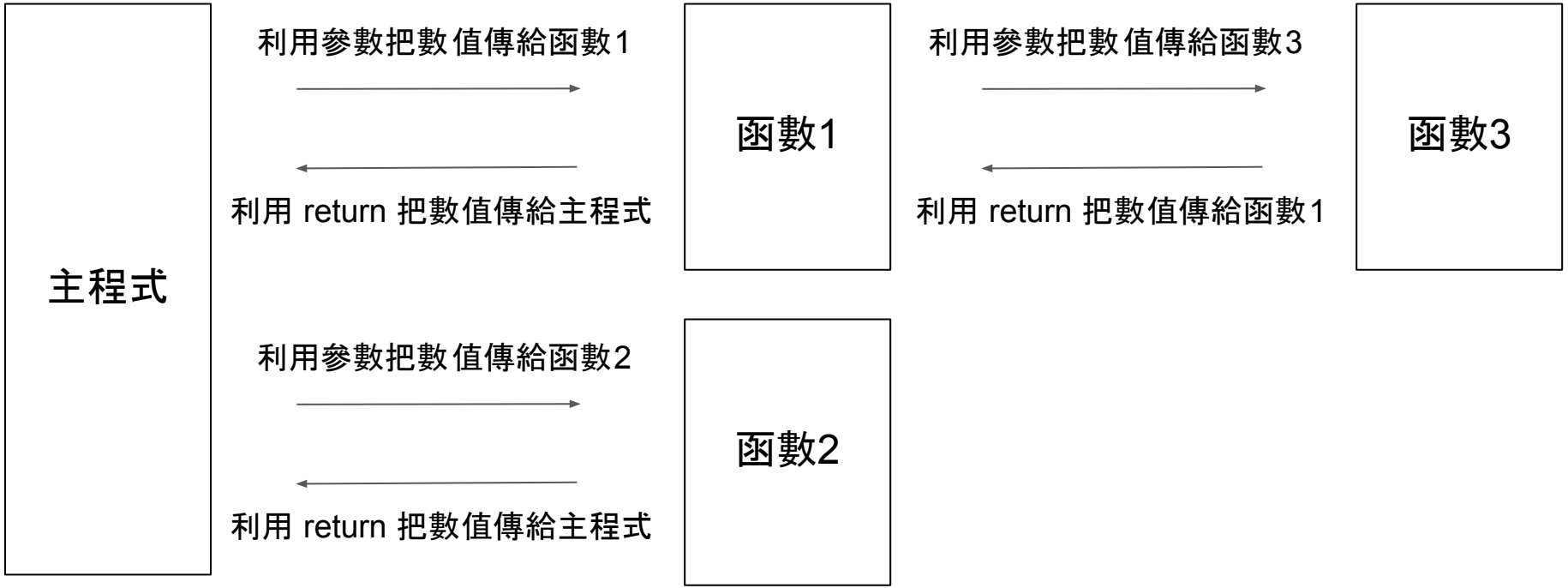
addAndPrint 函數內 a=1, b=2, print(3)

主程式呼叫函數概念圖

函數是一個黑盒 (Blackbox)。主程式不需要知道它的內部實作代碼，只需要知道它接受的參數，和返回的數值就可以了。



主程式和多個函數之間呼叫概念圖



完整示範

```
def area_circle(r):
    return 3.14 * (r ** 2)

def area_rectangle(a, b):
    return a * b

def total_area(a, b, r):
    area = area_circle(r) + area_rectangle(a, b)
    return area

area = total_area(1, 2, 3)          # area = 30.26
```

PYTHON 大數據採集課程

(補充) 全域變數 Global variable

前面提及過每個函數和主程式變數都是互相獨立的。但其實有一個例外，就是主程式定義的變數，函數其實也可以讀取到。這樣的設計是方便我們可以定義整個程式都會經常用到的 configuration parameters。

```
def func1(delay, file):
        sleep(delay)
        save(file)
def func2(delay, file):
        sleep(delay)
        save(file)

delay = 1
file = “result.csv”
func1(delay, file)
func2(delay, file)
```

每次都把變數 delay, file 傳給函數很麻煩！

```
def func1():
        sleep(delay)    # 函數可讀取到 delay, file 變數
        save(file)
def func2():
        sleep(delay)    # 函數可讀取到 delay, file 變數
        save(file)

delay = 1               # 主程式內定義 delay, file 變數
file = “result.csv”
func1()
func2()
```

運用全域變數程式就簡單多了！

練習

試實現函數 wash(clothes)

```
print("放入T恤")
print("加上洗衣粉")
print("開動洗衣機")
print("晾乾")

print("放入褲")
print("加上洗衣粉")
print("開動洗衣機")
print("晾乾")
```

改寫為

```
wash("T恤")

wash("褲")
```

效果

```
In [2]: wash("T恤")
放入T恤
加上洗衣粉
開動洗衣機
晾乾
Out[2]: 'T恤'

In [3]: wash("褲")
放入褲
加上洗衣粉
開動洗衣機
晾乾
Out[3]: '褲'
```

答案

```
def wash(clothes):
    print("放入" + clothes)
    print("加上洗衣粉")
    print("開動洗衣機")
    print("晾乾")
    return clothes
```

學習回顧

1. 函數定義
2. 輸入參數
3. 區域變數
4. 返回值
5. 呼叫其他函數