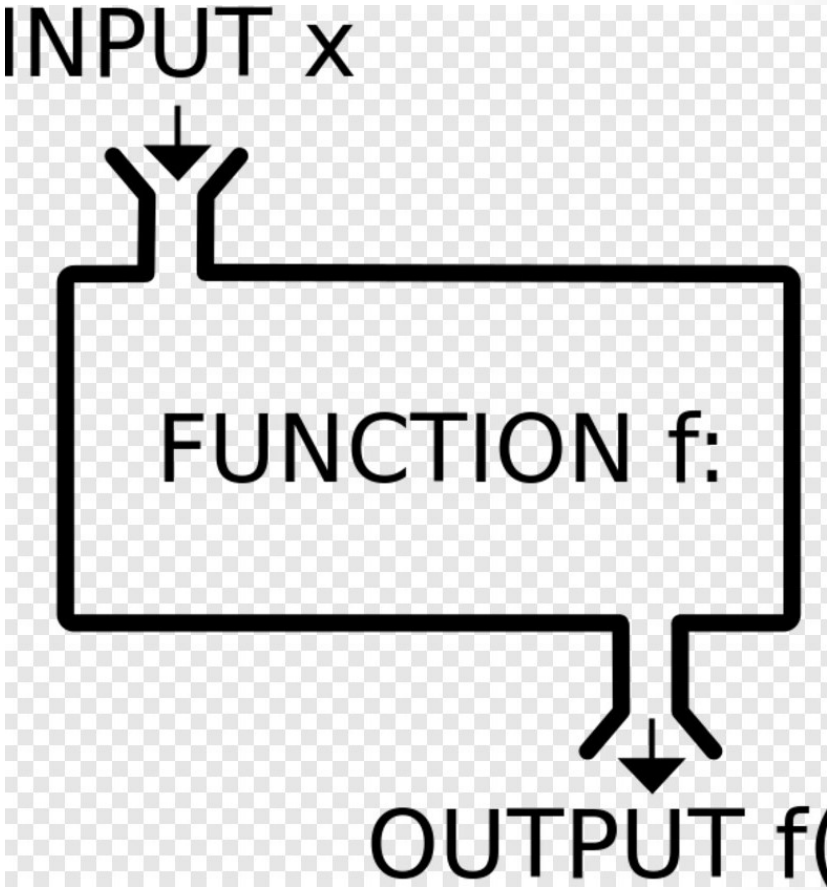
FUNCTION AND IMPORTING PACKAGES



Chih-Chung Hsu (許志仲)
Institute of Data Science
National Cheng Kung University
https://cchsu.info

套件引用方法 Import Package

- import 套件
  - 套件只要使用「import」命令就可匯入，import 命令的語法為：

```
import 套件名稱
```

  - 通常套件中有許多函式供設計者使用，使用這些函式的語法為：

```
套件名稱.函式名稱
```

Import Package

- import 命令的第二種語法為：

```
from 套件名稱 import *
```

- 以此種語法匯入套件後，使用套件函式就不必輸入套件名稱 (輸入套件名稱也可以)，直接使用函式即可。

- 此種方法雖然方便，卻隱藏著極大風險：每一個套件擁有眾多函式，若兩個套件具有相同名稱的函式，由於未輸入套件名稱，使用函式時可能造成錯誤。為兼顧便利性及安全性，可為套件名稱另取一個簡短的別名，語法為：

```
import 套件名稱 as 別名
```

- 這樣一來，使用函式時就用「別名. 函式名稱」呼叫，既可避免輸入較長的套件

常見函數:亂數套件

- Python 中常用的亂數套件函式有：(範例中的「r」為亂數套件的別名，str1="abcdefg", list1=["ab", "cd", "ef"])

函式	功能	範例	範例結果
choice(字串)	由字串中隨機取得一個字元	r.choice(str1)	b
randint(n1,n2)	由 n1 到 n2 之間隨機取得一個整數	r.randint(1,10)	7
random()	由 0 到 1 之間隨機取得一個浮點數	r.random()	0.893398…
randrange(n1,n2,n3)	由 n1 到 n2 之間每隔 n3 的數隨機取得一個整數	r.randrange (0,11,2)	8 (偶數)
sample(字串 ,n)	由字串中隨機取得 n 個字元	r.sample(str1,3)	['c', 'a', 'd']
shuffle(串列)	為串列洗牌	r.shuffle(list1)	['ef', 'ab', 'cd']
uniform(f1,f2)	由 f1 到 f2 之間隨機取得一個浮點數	r.uniform(1,10)	6.351865…

產生整數或浮點數的亂數函式

- randint 函式
  - randint 函式的功能是由指定範圍產生一個整數亂數，語法為：

    亂數套件別名 .randint(起始值， 終止值)

  - 執行後會產生一個在起始值 (含) 和終止值 (含) 之間的整數亂數，注意產生的亂數可能是起始值或終止值。

- uniform 函式
  - uniform 函式的功能是產生一個指定範圍的浮點數亂數，語法為：

    亂數套件別名 .uniform(起始值， 終止值)

  - 執行後會產生一個在起始值和終止值之間的整數亂數

亂數函式 Functions

- randrange 函式
  - randrange 函式的功能與 randint 雷同，也是產生一個整數亂數，只是其多了一個遞增值，語法為：

```
亂數套件別名 .randrange( 起始值 , 終止值 [, 遞增值 ])
```

  - 執行後會產生一個在起始值 (含) 和終止值 (不含) 之間，且每次增加遞增值的整數亂數，遞增值值可有可無，遞增值的預設值為 1。特別注意產生的亂數可能是起始值，但不包含終止值。

- random 函式
  - random 函式的功能是產生一個 0 到 1 之間的浮點數亂數，語法為：

```
亂數套件別名 .random()
```

隨機取得字元或串列元素

- choice 函式
  - choice 函式的功能是隨機取得一個字元或串列元素，語法為：

```
亂數套件別名 .choice( 字串或串列 )
```

  - 如果參數是字串，就隨機由字串中取得一個字元。
  - 如果參數是串列，就隨機由串列中取得一個元素。

隨機取得字元或串列元素

- sample 函式
  - sample 函式的功能與 choice 雷同，只是 sample 函式可以隨機取得多個字元或串列元素，語法為：

```
亂數套件別名.sample(字串或串列, 數量)
```

  - 如果參數是字串，就隨機由字串中取得指定數量的字元；如果參數是串列，就隨機由串列中取得指定數量的元素。
  - sample 函式最重要的用途是可以由串列中取得指定數量且不重複的元素。
  - 回想上次的練習題，我們應該會發現，前面的作法可能會重複抽到同一人。

時間套件

- 時間套件函式整理
  - Python 中常用的時間套件函式有：
  - 要使用時間功能需先匯入時間套件，匯入時間套件且設別名為「t」的程式為：

```
import time as t
```

函式	功能
clock()	取得程式執行時間。
ctime([時間數值])	以傳入的時間數值來取得時間字串。
localtime([時間數值])	以傳入的時間數值來取得時間元組資訊。
sleep(n)	程式停止執行 n 秒。
time()	取得目前時間數值。

取得時間訊息函式

- time 函式
  - Python 的時間是以 tick 為單位，長度為百萬分之一秒 (微秒)。Python 計時是從 1970 年 1 月1 日零時開始的秒數，此數值即為「時間數值」，是一個精確到小數點六位數的浮點數，time 函式可取得此時間數值，語法為：

```
時間套件別名 .time()
```

- localtime 函式
  - localtime 函式可以取得使用者時區的日期及時間資訊，語法為：

```
時間套件別名 .localtime([ 時間數值 ])
```

  - 「時間數值」參數可有可無， 若省略「時間數值」參數則是取得目前日期及時間，返回值是以元組資料型態傳回。

取得時間訊息函式

- localtime 函式傳回的元組資料，其意義為：
- 取得單一項目值的方式有兩種：一種為「物件. 名稱」，另一種為「元組[索引]」

序號	名稱	意義
0	tm_year	西元年
1	tm_mon	月份 (1 到 12)
2	tm_mday	日數 (1 到 31)
3	tm_hour	小時 (0 到 23)
4	tm_min	分鐘 (0 到 59)
5	tm_sec	秒數 (0 到 60，可能是閏秒)
6	tm_wday	星期幾 (0 到 6，星期一為 0，……，星期日為 6)
7	tm_yday	一年中的第幾天 (1 到 366，可能是閏年)
8	tm_isdst	時光節約時間 (1 為有時光節約時間 ，0 為無時光節約時間)

取得時間訊息函式

- ctime 函式
  - ctime 函式的功能及用法皆與 localtime 函式相同，不同處在於 ctime 函式的傳回值為字串。ctime 函式的語法為：

```
時間套件別名 .ctime([ 時間數值 ])
```

  - ctime 函式的傳回值格式為：

```
星期幾  月份  日數  小時：分鐘：秒數  西元年
```

執行程式相關時間函式

- sleep 函式
  - sleep 函式可讓程式休息一段時間，即程式停止執行一段時間，語法為：

    時間套件別名 .sleep(休息時間)

  - 「休息時間」的單位為「秒」 。

- clock 函式
  - clock 函式的功能是取得程式執行的時間：第一次使用 clock 函式是取得從程式開始執行到第一次使用 clock 函式的時間，第二次以後使用 clock 函式則是取得與第一次使用 clock 函式之間的程式執行時間。
    - In second

MORE THAN FUNCTION

產生器(generator)

- 使用函式製作產生器，產生器可以產生一個序列的資料，產生器要使用 yield 回傳資料，而非使用return 回傳資料，使用yeild 回傳資料會紀錄上一次回傳時函式的狀態，不會從頭到尾都執行。

- 使用產生器的好處是不用一次產生所有資料，當產生的資料量很大時會占用很多記憶體空間，產生器會一次產生一個資料，紀錄上一次執行的狀態，需要時再產生下一個資料。
  - Return 很多資料時，yield就很有用

- 分批次的概念!!

哦! 那什麼是 Generator?

- 先從For loop了解起。一般來說程式迴圈定義為

```
mylist = [1, 2, 3]
for i in mylist:
  print(i, end = ',')
```

- 這樣反覆的過程讓我們稱做這樣的流程為 iterative process，其中values會自動從mylist中反覆取出。可接受這樣反覆被取出的變數或形態，稱之iterable objects.

- See! It is unnecessary to store all values of mylist in memory all the time!!
  - yield!!

產生器(generator)：Sample code

- 還記得迴圈時使用的函式range，我們使用產生器撰寫自己的函式range，取名叫函式irange，產生器程式如右。

```
1  def  irange(st,et,step):
2       i=st
3       if  st<et:
4           while  i<et:
5               yield  i
6               i+=step
7       else:
8           while  i>et:
9               yield  i
10              i+=step
11
12 for  p  in  irange(0,10,2):
13      print(p,  end='#')
```

```
0#2#4#6#8#
```

產生器(generator)

使用 yield 回傳資料的函式會被認爲是產生器，執行以下程式。

```
x = irange(1,10)
print(x)
```

印出以下結果。

```
<generator object irange at 0x00000000010FA360>
```

表示 x 是一個產生器 (generator) 。

以下程式使用 for 迴圈取出產生器所產生的序列元素。

```
for i in irange(1, 5, 1):
   print(i)
```

Generator in Common Functions

- 其實如果傳統Range或是其他自動function能不能直接轉到Generator? 不用重寫
  - 可以，例如用 () 類似 tuple 形式包起來，取代 List
  - 另一種型態的Generator，用 iter包起來，例如 iter(mylist)

```
13  mylist = (x*x for x in range(3))
14  print(type(mylist))
15  for i in mylist:
16      print(i, end=',')
17  print()
18  for i in mylist:
19      print(i, end=',')
```

```
1 2 3
0, 1, 4,
<class 'generator'>
0, 1, 4,
```

Generator 用法

- 除了用 For-loop 的概念，也可以用 .__next__() 來取得下次的值
  - 或是用 next (my_list) 來取得
  - 適用於僅跑數次，或是測試用的情況

```
21  mylist = (x*x for x in range(3))
22  print(mylist.__next__())
23  print(mylist.__next__())
24  print(mylist.__next__())
```

```
0
1
4
```

產生器(generator)

```
1  def irange(start, stop, step=1):
2      if start < stop:
3          i = start
4          while i < stop:
5              yield i
6              i = i + step
7      else:
8          i = start
9          while i > stop:
10             yield i
11             i = i + step
12 x = irange(1,10)
13 print(x)
14 for i in irange(1, 5, 1):
15     print(i)
16 for i in irange(4, 1, -1):
17     print(i)
```

```
<generator object irange at
0x000000000110A360>
1
2
3
4
4
3
2
```

內部函式

- Python 函式內部可以包含另一個函式，函式內部的函式稱作內部函式。
- 內部函式用於函式內會一直重複利用到的功能，可以獨立出來寫成一個函式，在函式內呼叫使用，例如以下範例。

```
1  def hello(msg):
2      def say(text):
3          return 'Hello,'+text
4      print(say(msg))
5      print(say(' 你好 '))
6  hello('John')
```

```
Hello,John
Hello, 你好
```

內部函式

- 函式hello 內定義函式say，在函式hello 內呼叫了兩次say，分別傳了一個字串回來，使用函式print 將字串列印出來，執行時使用以下程式呼叫函式hello。
  - hello('John')
- 程式執行結果如下。
  - Hello,John
  - Hello, 你好

Variable Scope

- nonlocal and global 兩個修飾子，分別運用在全域與半全域

```
1   ##  variable  scop
2
3   x  =  'global  variable'
4
5   def  example():
6       global  x
7       y  =  'local  variable'
8       x  =  x  *  2
9       print(x)
10      print(y)
11
12  example()
```

```
global variableglobal variable
local variable
```

```
1   ##  variable  scop
2
3   x  =  'global  variable'
4
5   def  example():
6       x='local  variable'
7       print('inner-',x)
8       def  nl():
9           nonlocal  x
10          x  =  'inner  variable'
11          print('inner2-',  x)
12
13      nl()
14      print('after  call  func-',  x)
15
16  example()
17  print('global-',  x)
```

```
inner- local variable
inner2- inner variable
after call func- inner variable
global- global variable
```