

Algorithms

授課老師：張景堯



Python Basic：寫Python的一些重點概念提示

參考書：

- Ana Bell著、魏宏達譯、施威銘研究室監修. 用 Python 學運算思維. 旗標, 2019. ISBN : 9789863125518
- 施威銘研究室. Python 技術者們 - 實踐！帶你一步一腳印由初學到精通 第二版. 旗標, 2021. ISBN : 978-9863126614

Python的創始

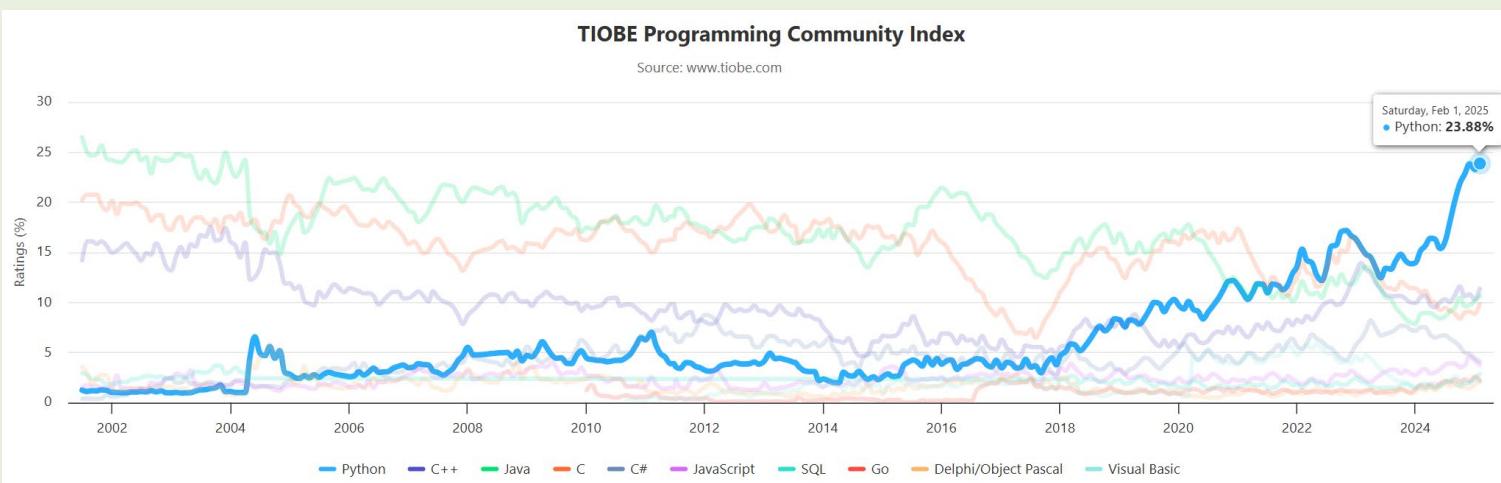
■起源：Python語言的創建者 [Guido van Rossum](#) 在1989年12月，為打發聖誕節前後的時間，著手構思一個新的語言，因本身是一個"Monty Python" 蒙提·派森飛行馬戲團的狂熱愛好者，選擇了Python作為專案的標題。

■創建者本尊：



Python熱門度

- <https://www.tiobe.com/tiobe-index/>
- 目前(Feb 2025)綜合指標排名第一，超越 C++, Java 與 C





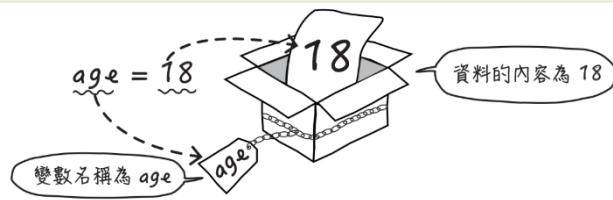
變數、資料型別與運算

名字(names)與變數(variables)

■ 嚴格地說

Python 有 name，但沒有 variable

- 在其他語言中，Variable 是獨立存在的，可以宣告定義，有其儲存空間可以操作
- Python 的 Name 是附屬的，是依附於 object，沒有可供 programmer 操作的儲存空間
- 在 Python 中，當使用 variable 這個詞時，往往只是取其傳統的語意，是指 name，而非指真實的可操作的標的。
- Python 的變數其實是名牌 (Name tag)



變數命名規則

■ 變數的命名規則

- 變數名必須以字母 (a-z 或是 A-Z) 或是底線 (_) 做開頭
- 從第二個字開始，可以是字母、數字或是底線
- 大小寫不同代表不同的名稱
- 名稱不可是 Python 的保留字 (如下表) 與內建函式名稱

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

一步一腳印

In [1]: print = 3 ← 把 print 當變數名稱用

In [2]: print(1+2) ← 哇！print() 不能當函式用了！！！
Traceback (most recent call last):

File "<ipython-input-2-3d60426f0d28>", line 1, in <module>
 print(1+2)

TypeError: 'int' object is not callable

Python 的基本型別

■ 5 種基本型別

- 整數（ int ）：0, 1, 2, -1000
- 浮點數（ float ）：0.0, 2.0, -22.7
- 布林（ bool ）：True, False
- 字串（ string ）：'hello', "we're #1!"
- None 型別（ NoneType ）：None

Python 的基本運算

■ 敘述 (statement) VS 運算式 (expression)

- 程式裡的每行程式碼稱作一行敘述
- 執行結果，可簡化成一個值，則稱為運算式
- 一行敘述裡不一定有運算式

■ 複合指派算符

一步一脚印

```
In [1]: a=6
In [2]: a=-1 ← a=a-1
In [3]: a
Out[3]: 5 ← 6-1=5
In [4]: a%=3 ← a=a%3
In [5]: a
Out[5]: 2 ← 5%3=2
```

第一個物件的型別	算符	第二個物件的型別	計算結果的型別	例子	計算結果
int	+	int	int	3 + 2	5
	-			3 - 2	1
	*			3 * 2	6
	**			3 ** 2	9
	%			3 % 2	1
int	/	float	float	3 / 2	1.5
	+			3 + 2.0	5.0
	-			3 - 2.0	1.0
	*			3 * 2.0	6.0
	**			3 ** 2.0	9.0
float	%	int	float	3 % 2.0	1.0
	+			3.0 + 2	5.0
	-			3.0 - 2	1.0
	*			3.0 * 2	6.0
	/			3.0 / 2	1.5
float	**	float	float	3.0 ** 2	9.0
	%			3.0 % 2	1.0
	+			3.0 + 2.0	5.0
	-			3.0 - 2.0	1.0
	*			3.0 * 2.0	6.0
float	/	float	float	3.0 / 2.0	1.5
	**			3.0 ** 2.0	9.0
	%			3.0 % 2.0	1.0
	+			3.0 % 2.0	1.0
	-			3.0 % 2.0	1.0

Python Bitwise Operators 位元運算子

Operator	Description	a = 60(0011 1100) b = 13(0000 1101)
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(a & b) will give 12 which is 0000 1100
	Binary OR Operator copies a bit if it exists in either operand.	(a b) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both .	(a ^ b) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~a) will give -61 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	a << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 will give 15 which is 0000 1111

It preserves the sign of the number

Python 的內建函式

■ 數值類函式

數值類函式	執行結果	功能
<code>abs(-2.5)</code>	2.5	取絕對值
<code>min(1, 2)</code>	1	取最小值, 參數可以有多個
<code>max(1, 2, 3)</code>	3	取最大值, 參數可以有多個
<code>pow(2, 3)</code>	8	2 的 3 次方
<code>pow(2, 3, 5)</code>	3	2 的 3 次方再除 5 取餘數
<code>round(1.35, 1)</code>	1.4	四捨六入到小數 1 位 (第 2 個參數表示要保留幾位小數)
<code>round(1.35)</code>	1	四捨六入到整數 (省略第 2 個參數時, 會進位到整數)

使用 `int()` 可只取整數位不做四捨六入，如 `int(1.6) -> 1`

Banker's rounding 如果是5的話會取最近的偶數，如
`round(0.5)->0`、`round(1.5)->2`



字串 string 、 tuple

字串的基本操作

■ 字串索引

- 字串中的字元是有順序性的，也就是說字串中的每個字元，都有一個特定的位置，這個位置就叫索引 (index)

P	y	t	h	o	n		r	u	I	e	s	!
Index	0	1	2	3	4	5	6	7	8	9	10	11
	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2

tuple 範例

- () — 空的 tuple.
- (1, 2, 3) — 包含 3 個整數物件的tuple.
- (1, "2", False) — 包含整數、字串和布林值物件的tuple.
- (5, (6, 7)) — 此tuple包含 1 個整數及另一個有 2 個整數的tuple物件.
- (5,) — 只包含 1 個整數物件的tuple. 注意：其中的
逗號告訴Python這是一個tuple，不是加上括號的
整數(5)。

Tuple Assignment

- Allow you to **assign** more than one variable at a time when the left side is a sequence.

```
m = ('have', 'fun')  
x, y = m  
print(x) #should print 'have'  
print(y) #should print 'fun'
```

- A particularly clever application of tuple assignment allows us to **swap the values of two variables in a single statement**.

```
a, b = b, a #same as (a, b) = (b, a)
```

切片基本操作

■ 字串切片

- 抓取部份文字的動作叫作擷取子 (substring)
- 用字串切片 (slicing) 的方式來抓取子字串，其格式如下：

[起始索引值：終止索引值：間隔值]

	P	y	t	h	o	n		r	u	I	e	s	!
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
[2:7:1]			(1)	(2)	(3)	(4)	(5)	X					
[2:11:3]			(1)			(2)			(3)			X	
[-2:-11:-3]			X			(3)			(2)			(1)	

圖 7.2 對字串 "Python rules!" 進行切割，數字代表會依此順序切割出新的字串。

再談切片(slicing)

■省略起始或終止索引時：

	0	1	2	3	結果	說明
index	0 -4	1 3	2 -2	3 -1		
a[:2]	(1)	(2)	X		(0, 1)	由索引 0 切到 (不含) 2
a[:-1]	(1)	(2)	(3)	X	(0,1,2)	由索引 0 切到 (不含) -1 (最後一個元素)
a[2:]			(1)	(2)	(2, 3)	由索引 2 切到 (含) 最後一個元素
a[-2:]			(1)	(2)	(2, 3)	由索引 -2 切到 (含) 最後一個元素
a[-1:]				(1)	(3,)	由索引 -1 切到 (含) 最後一個元素
a[:]	(1)	(2)	(3)	(4)	(0,1,2,3)	由索引 0 切到 (含) 最後一個元素

圖 9.3 在切片時省略起始或終止索引值時的各種應用

■終止索引大於元素數量 → 只會切到最後一個

■起始和終止索引相同 → 空 tuple

再談切片 (slicing)

■ 間隔值為負 → 反向切片

	0	1	2	3		
index	0 -4	1 3	2 -2	3 -1	結果	說明
a[: :-1]			(2)	(1)	(3, 2)	由最後一個元素切到 (不含) 索引 1
a[1: :-1]	(2)	(1)			(1, 0)	由索引 1 切到 (含) 第一個元素
a[2: :-2]	(2)		(1)		(2, 0)	由索引 2 切到 (含) 第一個元素，間隔 2
a[: :-1]	(4)	(3)	(2)	(1)	(3,2,1,0)	由最後一個元素切到 (含) 第一個元素
a[: :-2]		(2)		(1)	(3,1)	由最後一個元素切到 (含) 第一個元素，間隔 2

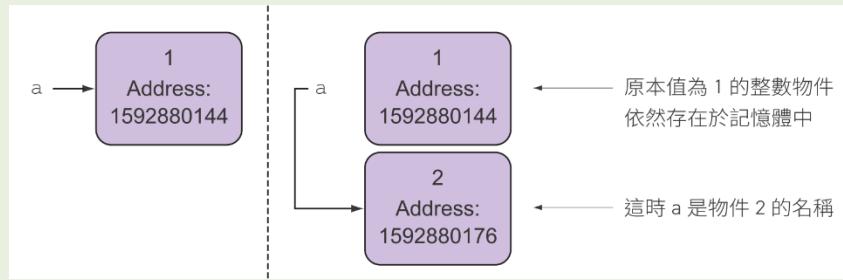
圖 9.4 將切片間隔設為負數時的各種應用



串列 list

不可變(immutable)物件

- 整數、浮點數、布林、字串和 tuple 都是不可變物件 (immutable object)
- 物件賦予特定值後，此值就無法改變



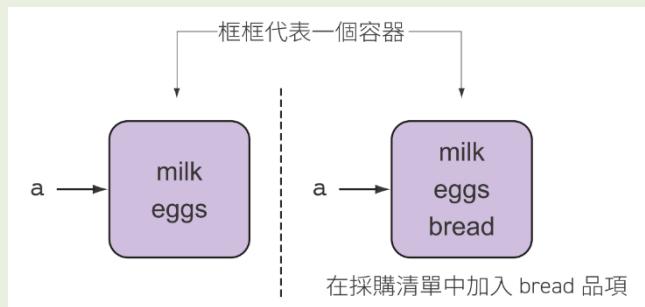
■ 不可變物件的特性

- 它們因為不可變所以比較安全，其值不會不小心被更動
- 它們操作的效率比可變物件高
- 只能用一個新記憶體位置來存放處理後的結果

可變(mutable)物件

■ 物件賦予特定值後，此值依然可被修改

■ 下圖說明了當我們修改可變物件的值時，指向該物件的變數並沒有改變



■ 常用的可變物件(mutable object)有以下 2 種，它們都是容器(Container)：

- 串列(list)
- 字典(dict)

串列(list)

- 串列(list)就是有儲存順序的可變容器
- 串列的格式如下：

a = [元素 0, 元素 1, 元素 2, 元素 3,]

 └ 串列以中括號開頭

 └ 以中括號結束

■ list 可就地修改

a = ["milk", "eggs", "bread"] ————— 以 list 建立清單

a[1] = "apple" ————— 直接就地修改第 1 個 (索引 1) 元素，

print(a) ————— 會輸出 ['milk', 'apple', 'bread'] 即快速又省空間

串列(list)

■tuple 與串列的使用時機

特性＼容器	tuple	list
容器內的元素永遠不會改變	○	×
佔用空間較小、讀取速度較快	○	×
可就地修改資料、快速又不浪費空間	×	○

建立串列(list)並以索引(index)取值

- 用中括號 [] 來建立串列
- 當然也可以建立含有元素的串列，例如：

```
grocery = ["milk", "eggs", "bread"]
```

- 用中括號 [] 來指定索引，以取得串列內該索引位置的元素，例如：

```
grocery = ["milk", "eggs", "bread"]
print(grocery[0])    # 會顯示 milk
print(grocery[1])    # 會顯示 eggs
print(grocery[2])    # 會顯示 bread
```

串列的排序

■用 `sort()`方法就地排序串列元素

- `L.sort()`就是將 `L` 串列裡的元素，以遞增方式(如果元素是數字)或字母順序(如果元素是字串)排序

串列的反轉

■用 reverse()方法就地反轉串列元素

- 例如

程式 26.1

排序和反轉串列

```
heights = [1.4, 1.3, 1.5, 2, 1.4, 1.5, 1]
```

```
heights.reverse() —— 將串列反轉  
print(heights) —— 會顯示 [1, 1.5, 1.4, 2, 1.5, 1.3, 1.4]，因為上一行程式將串列  
反轉，所以原本索引 0 的元素會變成最後一個元素，而  
原本索引 1 的元素會變成倒數第二個元素，依此類推  
heights.sort() —— 將串列作遞增排序  
print(heights) —— 顯示 [1, 1.3, 1.4, 1.4, 1.5, 1.5, 2]，  
因為上一行程式將串列作遞增排序  
heights.reverse() —— 將串列反轉  
print(heights) —— 顯示 [2, 1.5, 1.5, 1.4, 1.4, 1.3, 1]，  
因為上一行程式將串列反轉，變遞減排序了
```

- 使用 `[::-1]`也會反轉，但只傳回而不改變串列內容

關於 sorted() 及 reversed() 函式

■ 都不是就地處理的操作。

程式	執行結果	說明
sorted([3, 1, 2])	[1, 2, 3]	傳回 list 容器排序後的 list
sorted({3, 1, 2})	[1, 2, 3]	傳回 set 容器排序後的 list
sorted({3:7, 1:8, 2:9})	[1, 2, 3]	傳回 dict 容器排序後的 list
sorted('bac', reverse=True)	['c', 'b', 'a']	傳回 str 反向排序後的 list

程式	執行結果	說明
list(reversed([1,2,3,0]))	[0, 3, 2, 1]	把 list 中的元素反轉
tuple(reversed('bdac'))	('c', 'a', 'd', 'b')	把字串中的字元反轉
tuple(reversed({0,1,2}))	TypeError: 'set' object is not reversible	非序列容器不可反轉

多層的 list

一步一脚印

```
In [1]: fruit=[['蘋果', 82], ['香蕉', 45], ['芭樂', 59]] ←
```

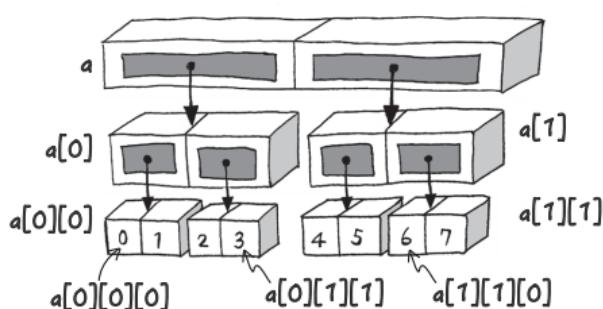
```
In [2]: fruit[0][1] ←
```

```
Out[2]: 82
```

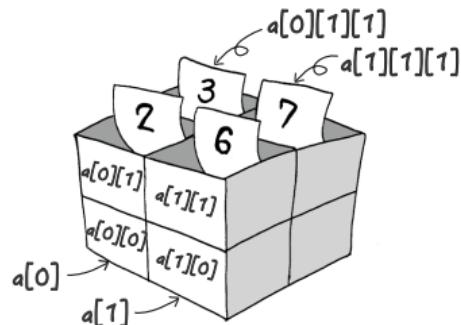
查看第 0 組的水果價格 (注意索引由 0 開始,
[0][1] 表示取第 0 個子串列的第 1 個元素)

建立 3 組水果價格組合,
在大 list 中有 3 個小 list

```
a = [[ [ 0, 1 ], [ 2, 3 ] ], [ [ 4, 5 ], [ 6, 7 ] ] ]
```



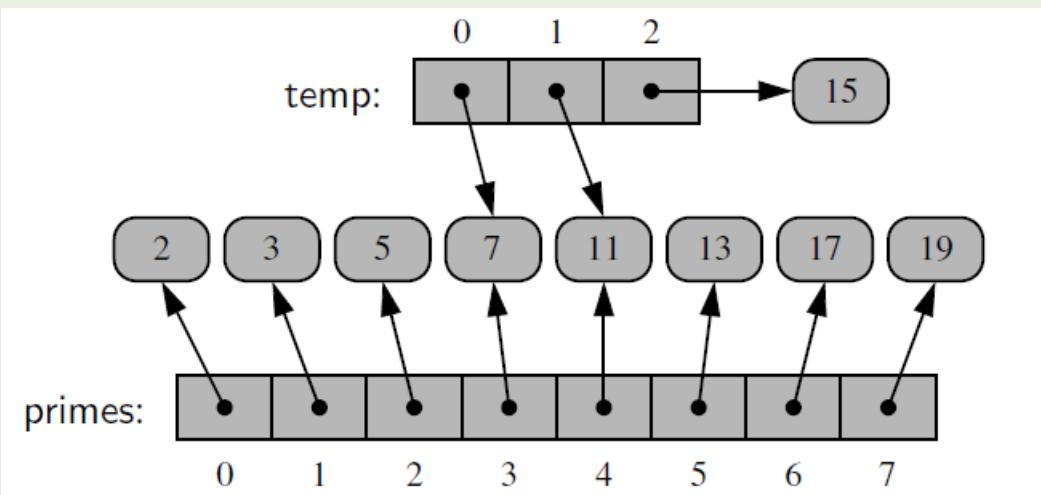
容器的階層式看法



容器的立體 (維度) 看法

Python串列的儲存結構

■ primes = [2, 3, 5, 7, 11, 13, 17, 19]
temp = [7, 11, 15]



Trap of 2D List Initialization

- 試試並比較底下兩段程式結果。

```
list2d_1 = [[0]*9]*9  
list2d_1[0][0]=1  
list2d_1  
  
list2d_2 = [[0]*9 for _ in range(9)]  
list2d_2[0][0]=1  
list2d_2
```



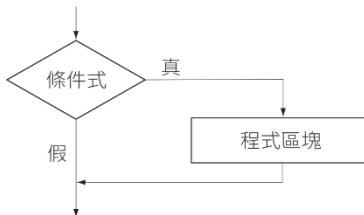
Python的流程控制

if 條件式

- if 條件式語法 【如果 ... 就 ...】

```
if 條件式 : # 注意最後要加 : (冒號)
```

 程式區塊.....



- 當條件式為 True 時就執行程式區塊內的敘述，否則就略過。
例如：

縮排很重要！

```
if a < 1:  
    a += 1  
    b = a + 3  
print(b) ----- 從 print(b) 以下的程式未縮排，不屬於 if 區塊  
...  
...
```

更直覺的邏輯比較寫法



```
In [1]: a = 4
```

```
In [2]: 3 < a < 6 ← 是否 a 大於 3 小於 6
```

```
Out[2]: True
```

```
In [3]: 3 < a and a < 6 ← 這是傳統寫法，比較不夠直覺
```

```
Out[3]: True
```

```
In [4]: 3 < a < 6 > (a + 1) ← 是否 a 大於 3 小於 6 且 6 大於 a+1
```

```
Out[4]: True
```

```
In [5]: 3 < a and a < 6 and 6 > (a+1) ← 相當於傳統寫法的多個 and
```

```
Out[5]: True
```

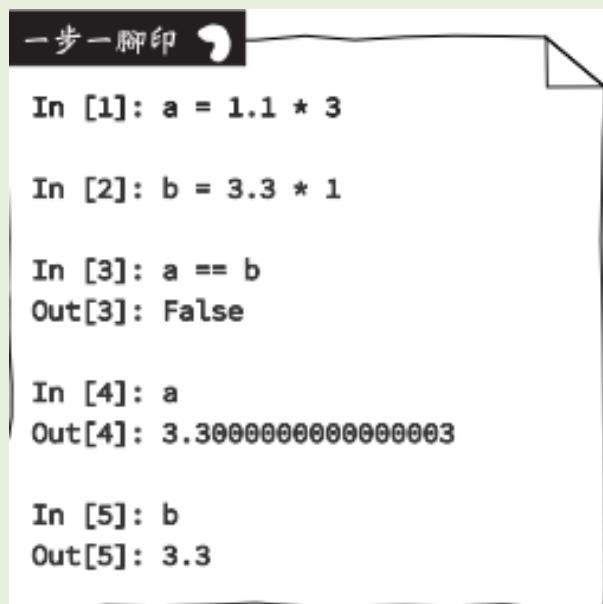
算符的優先順序

表 13.2 Python 算符的優先順序

算符	表示
()	括號
**	指數運算
*	乘法運算
/	除法運算
//	Floor division，除法取商的整數
%	餘數運算
+	加法運算
-	減法運算
==	比較算符，判斷是否相同
!=	比較算符，是否不同
>	比較算符，大於
>=	比較算符，大於等於
<	比較算符，小於
<=	比較算符，小於等於
is	判斷所儲存的物件是否和另一變數儲存的物件相同
is not	判斷所儲存的物件是否和另一變數儲存的物件不同
in	判斷所儲存的物件是否在另一個物件裡
not in	判斷所儲存的物件是否不在另一個物件裡
not	邏輯算符 not
and	邏輯算符 and
or	邏輯算符 or

注意浮點數的精確度

■ 注意!!! 使用 `==` 做條件運算時，最好不要用浮點數，因為浮點數運算的小數點精度不易拿捏



```
一步一脚印

In [1]: a = 1.1 * 3

In [2]: b = 3.3 * 1

In [3]: a == b
Out[3]: False

In [4]: a
Out[4]: 3.3000000000000003

In [5]: b
Out[5]: 3.3
```

多重選擇條件式(if-else)

■if- else 語法 【如果 ... , 就 ... , 否則就 ...】

if 條件式：

```
    程式區塊 _A      # 條件為 True 時要執行的程式  
    else :           # 注意 else 最後也要加：(冒號)  
    程式區塊 _B      # 條件為 False 時要執行的程式
```

程式 14.1 | 判斷數字是奇數或偶數

```
Num=int(input("please pick a number: "))  
if Num%2==0:  
    print("Even Number")  
else:  
    print("Odd Number")
```

可改用 bitwise operator
 $Num \& 1 == 0$

多重選擇條件式(if-elif-else)

■if-elif-else 語法

【如果 x...，就 A...，否則如果 y...，就 B...，
否則就 C...】

```
if 條件 _x :           # 如果 x  
    程式區塊 _A  
  
elif 條件 _y :         # 否則如果 y  
    程式區塊 _B  
  
else :                 # 否則  
    程式區塊 _C
```

條件運算式

傳統寫法

```
if c > 5:  
    a = 1  
else:  
    a = 2
```

條件運算式：
一行就搞定！

```
a = 1 if c>5 else 2 # 這裡 else 2 就是  
↑ 真 ↑ 假 ↑ else a = 2 的意思
```

TIPS

條件算符 if...else 的優先順序是最低的，
因此以上 $c>5$ 會優先運算。

```
price = 100  
price = price*0.8 if price > 99 else price*0.9  
print(price) ➔ 80.0
```

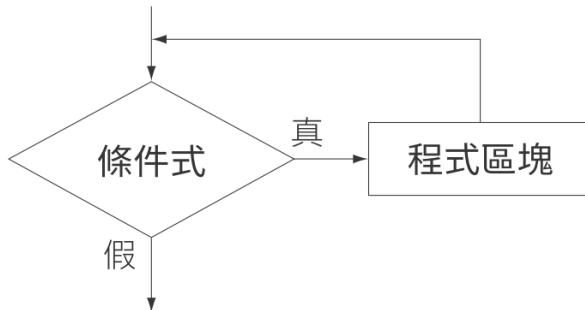
重複執行一樣的動作

■while 的語法

- 以下為 while 迴圈的基本語法。

while 條件式 : ————— 每次迴圈的起始點，會檢查條件式來決定要繼續或結束迴圈

程式區塊 ————— 要重複執行的程式區塊



- while迴圈常用在不定次數迴圈。

用 break、continue 進一步控制迴圈執行

■ 提早跳離迴圈(**break**)，猜 100 次不中就 Game Over 離開猜數字迴圈。

while 條件式：

....
continue 跳到下一圈的開頭
(略過後面程式敘述)

....
break

....
跳出迴圈

程式 16.3 使用 break 跳離迴圈

```
secret = "code"
max_tries = 100
guess = input("Guess a word: ")
tries = 1

while guess != secret:
    print("You tried to guess", tries, "times")
    if tries == max_tries:
        print("You ran out of tries.")
        break
    guess = input("Guess again: ")
    tries += 1

if tries <= max_tries and guess == secret:
    print("You got it!")
```

當 tries 的值等於 max_tries
的值時，會跳離迴圈

如果跳離迴圈的原因是猜中單字，則
會顯示 "You got it!"

用 break、continue 進一步控制迴圈執行

■ 返回迴圈的開頭(continue)

- continue 來跳過 5 不印，最後再顯示 "END" :

程式 16.4 用 continue 跳過 5 不印

```
i = 1
→while(i < 10):
    if i == 5:
        i += 1
        continue —— 若 i == 5 會跳到迴圈開頭
    print(i, end=' ')
    i += 1
print("End")
```

↓ 執行結果

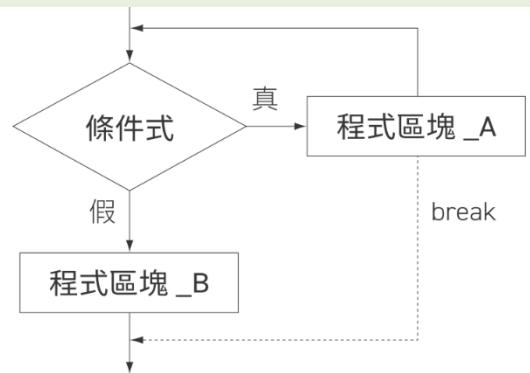
1 2 3 4 6 7 8 9 End

print() 在輸出資料後預設會自動換行，我們可用參數
end=' ' 來將換行改成只輸出一個空格，因此每次輸出資
料後只會加一空格而不會換行了。(這裡的「end= 值」
為指名參數的用法，此用法我們留到 21-1 節再詳說明)

while-else 迴圈

■ while 也可以有 else ，它跟 if 的 else 很像，就是當 while 條件不成立時，就會執行 else 的程式區塊，然後結束迴圈

```
while 條件式：  
    程式區塊 _A  
else：  
    程式區塊 _B
```



指定迴圈執行次數

■for 是 Python 的保留字，若要控制迴圈執行次數，可以搭配 range() 函式來使用，基本語法如下：

```
for 變數 in range(N):
```

```
    程式區塊
```

range()函式產生的數列範圍

■range()函式的完整參數如下：

range(起始值, 終止值, 間隔值)

■程式 17.3 分別是 $10 \sim 2$ 及 $0 \sim 8$ 偶數，兩個不同數列範圍的 for 迴圈：

程式 17.3 不同數列範圍的 for 迴圈

```
for i in range(10, 1, -1): ----- 間隔值為負數，起始值要大於終止值，  
    print(i, end=" ")           產生  $10 \sim 2$  由大到小的數列  
print()  
for i in range(0, 9, 2): -產生  $0 \sim 8$           參數 end=" " 可將 print() 最後的換行改成只  
    print(i, end=" ")           輸出一個空格 (「end= 值」為指名參數的用  
                                法，此用法我們留到 21-1 節再詳說明)
```

使用 break、continue 與 for-else

程式 17.5 連續印出 1~10 但跳過 5 不印

```
for i in range(1, 20):
    if i == 5: _____若 i==5 就略過不印
        continue
    print(i, end=" ")
    if i == 10: _____若 i==10 就跳出迴圈
        break
print("END")
```

使用 break、continue 與 for-else

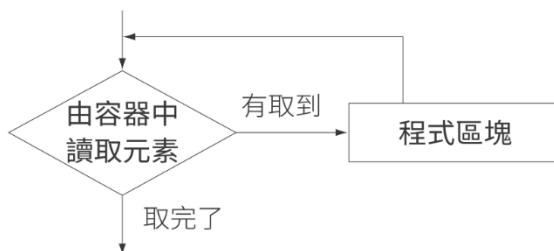
■for 也可以搭配 else，當 for 迴圈正常結束後，會接著執行 else 區塊；若是在 for 的程式區塊中用 break 跳出迴圈則不執行 else 後的程式區塊

```
for 變數 in range(…):  
    程式區塊 _A  
else:  
    程式區塊 _B
```

走訪 iteration

■走訪，則是將容器中的元素一一取出來做處理，其語法及流程圖如下：

```
for 變數 in 容器  
    程式區塊
```



走訪字串裡的字元

■ 程式 18.1 用 for 迴圈走訪 "Python is fun!" 中的每一個字元：

程式 18.1 使用 for 迴圈走訪字串中的字元

```
for ch in "Python is fun!": _____ ch 為迴圈變數  
    print("the character is", ch) _____ 顯示 ch 的值
```

■ 走訪過程。

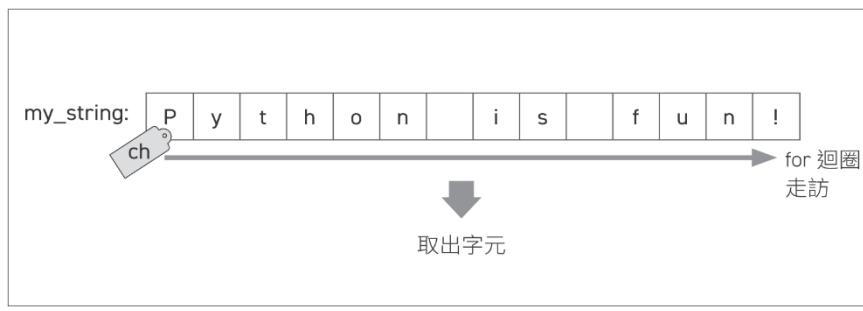


圖 18.1 程式 18.1 在每次迴圈執行時，都會將走訪到的字元指派給迴圈變數。

走訪字串裡的字元

■ 使用 for 迴圈走訪字串的索引值

程式 18.2 使用 for 迴圈走訪字串的索引值

```
my_string = "Python is fun!" ————— 將字串儲存於變數中  
len_s = len(my_string) ————— 取得字串的長度  
for i in range(len_s): ————— 走訪 0 到 len_s-1 的數列  
    print("the character is", my_string[i]) —— 用索引值讀取字串中的字元
```

■ 走訪過程。

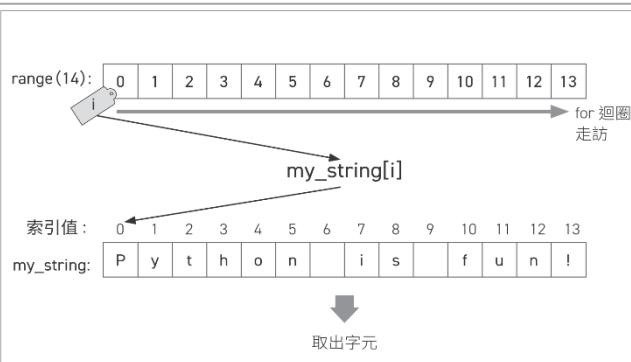


圖 18.2 程式 18.2 在每次迴圈執行時，將走訪到的整數指派給迴圈變數，然後將迴圈變數的直當作索引，到字串中讀取對應的字元。

for 的容器生成式

■生成串列的就稱為串列生成式 (List Comprehensions)，生成字典的就叫做字典生成式，其他以此類推。各種生成式的語法如下：

串列生成式：[運算式 for 變數 in 容器] ← 用中括號

集合生成式：{ 運算式 for 變數 in 容器 } ← 用大括號

字典生成式：{ 運算式 k: 運算式 v for 變數 in 容器 } ← 注意

字典生成式：{ 運算式_k: 運算式_v for 變數 in 容器 } ← 注意裡頭有：

```
print([i*i for i in range(1, 6)])
```

```
print({i*i for i in range(1, 6)})      # 集合  
print({i: i*i for i in range(1, 6)})
```

```
print({i: i*i for i in range(1, 6)})
```

除了用 range() 外，也可以用其他容器哦！ 5: 25} # 字典

■ 在生成式中還可以加上 if 來做篩選

```
s = [i*i for i in range(1, 11) if i%2 == 0]
print(s) ➔ [4, 16, 36, 64, 100]
```

for 迴圈 vs while 迴圈

■ for 迴圈 vs. while 迴圈

程式 18.7 for 迴圈 vs. while 迴圈

for 迴圈

```
for x in range(3):——迴圈的起始點  
    print("var x is", x)
```

while 迴圈

```
x = 0 ————— 初始化迴圈變數  
while x < 3:  
    print("var x is", x)  
    x += 1 —————  
                    自行控制並修改迴圈變數
```



函式 Function

撰寫函式

■在 Python 裡，實作一個函式需要使用 `def` 保留字，語法如下：

```
def 函式名稱(參數 1, 參數 2, ...):
```

```
.....
```

程式區塊

```
.....
```

■Python 在執行程式時，會依順序先看到程式中 `def` 所定義的函式，並將 **函式名稱綁定**（如同指派），以便後續程式以函式名稱呼叫時能找到相應程式碼。

傳回值

■函式的傳回值

- 在 Python 裡是用 `return` 來傳回函式的結果，當執行到 `return` 這一行程式時，表示該函式已處理完畢，接著要傳回執行後的結果。

程式 21.4 實作函式來計算 2 個字串的總長度

```
def get_word_length(word1, word2):————定義函式，包含函式名稱  
    word = word1+word2————串接兩個字串    及需要什麼參數  
    return len(word)————傳回串接字串的長度  
    print("this never gets printed")——在 return 之後的程式碼不會被執行
```

■多個傳回值

- Python 規定函式只能傳回一個值(物件)，如果想要傳回多個值，則可使用 `tuple`

函式參數

■位置參數與指名參數

- 可以直接用參數的名稱來指定參數值，這樣就不用依照順序了，此方式稱為指名參數法(using named parameter)或關鍵字參數法(using keyword parameters)，例如底下的程式：

程 21.3 呼叫函式時的 2 種參數傳遞方式

```
sayHi('王小明', title='同學')——— 第 2 個參數值用名稱指定  
sayHi(title='同學', name='王小明')—— 全部參數值都用名稱指定，  
sayHi(title='同學', '王小明')—— 可以不照順序
```

錯誤了！因為 '王小明' 並未指定參數名，所以 不是指名參數而是位置參數，要放在前面才行

- 規則：位置參數不能放在指名參數後。

呼叫函式

- 依據函式定義的參數傳入對應的值，這些值就稱為**實際參數(actual parameter)**
- 它們會自動指派給函式中定義的形式參數(formal parameter)

程式 21.5 如何呼叫函式

```
def word_length(word1, word2):  
    word = word1+word2  
    return len(word)  
    print("this never gets printed")  
  
length1 = word_length("Rob", "Banks")  
length2 = word_length("Barbie", "Kenn")  
length3 = word_length("Holly", "Jolley")  
  
print("One name is", length1, "letters long.")  
print("Another name is", length2, "letters long.")  
print("The final name is", length3, "letters long.)
```

函式 word_length 的函式定義及內容

呼叫函式並帶入不同的參數，將傳回結果指派給不同的變數

顯示結果

呼叫函式
word_length("Rob", "Banks")

② ④

①

實際參數

函式的範圍
word_length(word1, word2)

word1: Rob
word2: Banks

③ ⑤

形式參數

word: RobBanks

⑥

Returns: 8

⑦

函式傳遞參數方式

- Python函式傳遞參數方式就是用指派來傳遞的。這種傳遞方式被稱為 pass by assignment。
- 整數、浮點、字串、tuple這些不可變物件當參數的話，其行為就像 call by value那樣。
- 而 list、dict、set這些可變物件當參數的話，當形式參數與實際參數經過呼叫後就互為別名(alias)，像 call by reference，不過若要保持別名關係，形式參數只能在函式內做原地(in-place)修改，不能re-assignment。

變數的有效範圍(Scope)

- 有效範圍僅在函式的內部，而不能在函式之外存取
 - 程式 22.1 在主程式和函式中定義相同名稱的變數，其有效範圍各自不同。

程式 22.1 在主程式和函式中定義相同名稱的變數

定義函式 | def fairy_tale():——定義函式
 peter = 5——函式內的變數 peter，初始化為 5
 print(peter)——畫面顯示 5

主程式 | peter = 30——在主程式裡建立變數 peter，並初始值為 30
 fairy_tale()——呼叫函式，在函式中會輸出函式內建立的變數 peter，
 print(peter)——所以會顯示 5。該函式沒有 return，所以會傳回 None

畫面顯示 30

5

30

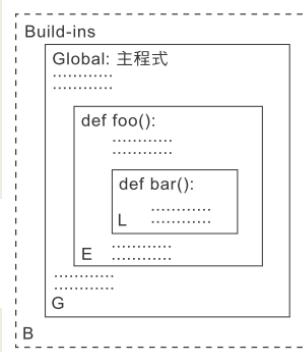
變數的有效範圍(Scope)

■ 變數有效範圍的規則

- 在函式中要存取某個變數時，會優先在函式的範圍中尋找此變數，如果有找到就使用它。如果沒有，就往外層的範圍尋找。這裡說的外層，通常是主程式，但也可能是巢狀函式(稍後會介紹)的外層函式。
- 如果在外層範圍中有找到，就使用該變數，否則繼續往更外層尋找。
- 如果在最外層的主程式中也沒找到，就會發生變數找不到的錯誤。

■ Python 的 scope rule 是依照 LEGB 的順序來尋找名稱 (name) 的 (包含變數和函式名)

Local → Enclosing → Global → Built-ins



global 宣告

■ 在函式中指定要變更全域變數值

```
a = b = c = 1
```

```
def test(b):
    global c      ← 在定義函式時,指明要使用全域變數c
    a = 2        ← 這時的 a 並非上層的全域變數,而是新建立的區域變數
    c = 33       ← 因為已經宣告了 global c, 所以這個 c 是指上層的
                  全域變數,而非新建一個區域變數
    print(a, b, c) ← 輸出區域變數 a、b 及全域變數 c
```

```
test(3)          → 2 3 33 ← 呼叫 test(3) 會輸出區域變數 a、b 及全域變數 c
print(a, b, c) → 1 1 33 ← 全域變數 c 的值被改為 33 了
```



其他可變(Mutable)物件 : set, dict

set：一堆資料的集合

■為了確保集合中的「元素是不重複的」，所以集合不允許放入「元素可以改變(可變物件，如串列、字典、集合)」的資料

一步一腳印

```
In [1]: {[1,2], 3, 4} ← 用 list 做為集合的元素  
Traceback (most recent call last):
```

```
File "<ipython-input-2-085e1e7e730f>", line 1, in <module>  
 {[1,2], 3, 4}
```

```
TypeError: unhashable type: 'list' ← 會發生 TypeError:  
unhashable type 的錯誤
```

set 的算符

■ 傳統的比較算符 ($>$, \geq , $<$, \leq , $=$, $!=$) 也可以使用，其比較方式如下

- 若 a 中的元素和 b 完全相同(無關順序)，則視為相等
- 若 a 中的元素在 b 中都有，而且 b 還更多，則 $a < b$ 。



```
In [1]: 1 in {1, 2}  
Out[1]: True
```

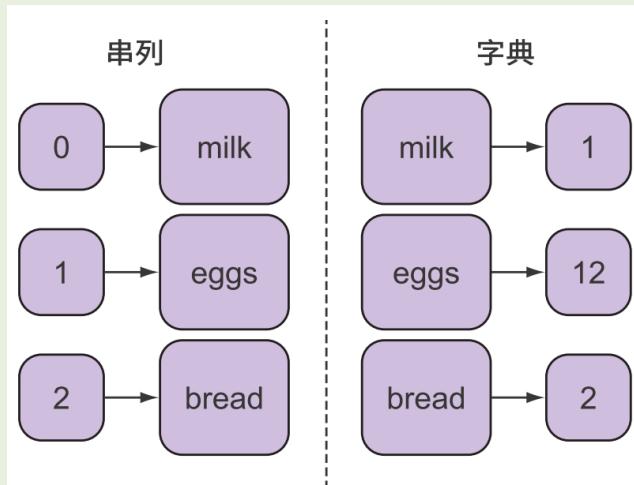
```
In [2]: 0 in {1, 2}  
Out[2]: False
```

何時需要使用集合？

- 用集合「記錄事物是否曾出現過」
 - 使用集合來記錄每個單字/數值/字元/自訂物件是否曾出現過。

字典

- 字典的存取方式是以鍵取值，例如 `d['milk']` 可以讀取字典 `d` 中鍵為 '`milk`' 的值
- 字典則更有彈性，其索引(鍵)不一定要是整數，也可以是其他物件



建立字典並以鍵(key)取值

■在 Python 裡，可以用大括號來建立空字典：

```
grocery = {}
```

■圖 27.1 的超市採買清單為例，建立一個包含商品名稱(鍵)和購買的數量(值)的字典：

```
grocery = {"milk": 1, "eggs": 12, "bread": 2}
```

milk

1

鍵

eggs

12

值

bread

2

鍵

值

取得字典裡所有的鍵值對

- 用 `items()`方法取得字典裡所有的鍵值tuple

```
songs = {"believe":3, "roar":3, "let it be":4}  
print(songs.items())  
#會顯示 dict_items([('believe', 3), ('roar', 3), ('let it be', 4)])
```

- 使用 `for` 迴圈來走訪這個物件，利用tuple assignment逐筆取得字典裡的每對鍵與值：

```
for key, value in songs.items():
```

字典裡的資料

■字典裡的資料沒有固定順序

- 試著執行下面兩行程式，來觀察字典和串列是如何比較「是否相等」：

```
print({"Angela": 70, "Bruce": 50} == {"Bruce": 50, "Angela": 70}) —
```

顯示 True

```
print(["Angela", "Bruce"] == ["Bruce", "Angela"]) — 顯示 False
```

何時需要使用字典？

- 用字典「記錄事物的次數」
 - 使用字典來記錄每個單字的出現次數。
- 用字典「以鍵執行對應的函式」
 - 一個很實用的技巧，就是用字典來將字串和函式配對



類別 class

建立類別

- 使用 `class` 保留字定義一個新的類別。
- 例如：我們想建立一個類別，此類別名稱叫 `Circle`：

```
class Circle:  
    |  
    | 保留字 類別名稱
```

定義類別的屬性(attribute)

■ 透過 `__init__` 類別函式初始化物件

```
class Circle:  
    def __init__(self):  
        # 在此撰寫程式碼
```

■ `__init__` 必須接收一個 `self` 參數，當建立新物件時，
Python 會自動以新物件當作參數來呼叫 `__init__`，以便程
式可對新物件來進行初始化操作。

■ 在 `__init__` 方法裡建立類別的屬性

- `Circle` 類別的例子，我們可以在 `__init__` 裡，定義此類別有一
個屬性叫 `radius`，並初始化其值為 0：

```
class Circle:  
    def __init__(self):  
        self.radius = 0
```

代表未來新建立的物件 屬性名稱 初始值

定義類別的方法(method)

- 實作一個可以修改圓形半徑的方法來做示範

```
class Circle:  
    def __init__(self):  
        self.radius = 0  
    def change_radius(self, radius):  
        self.radius = radius
```

方法的名稱 _____ 類別中定義的方法，第一個參數必須是 self
self.radius = radius _____ 方法實際要執行的動作

常見Class Special Method Names

- 試試建立下列類別，並測試 len()、str()/print()及以該類別物件做for迴圈。

```
class Test:  
    def __init__(self, w):  
        self.word = w  
    def __len__(self):  
        return len(self.word)  
    def __str__(self):  
        return f"Hello, {self.word}!"  
    def __iter__(self):  
        for c in self.word:  
            yield c
```



Asymptotic notation(漸近表示法)

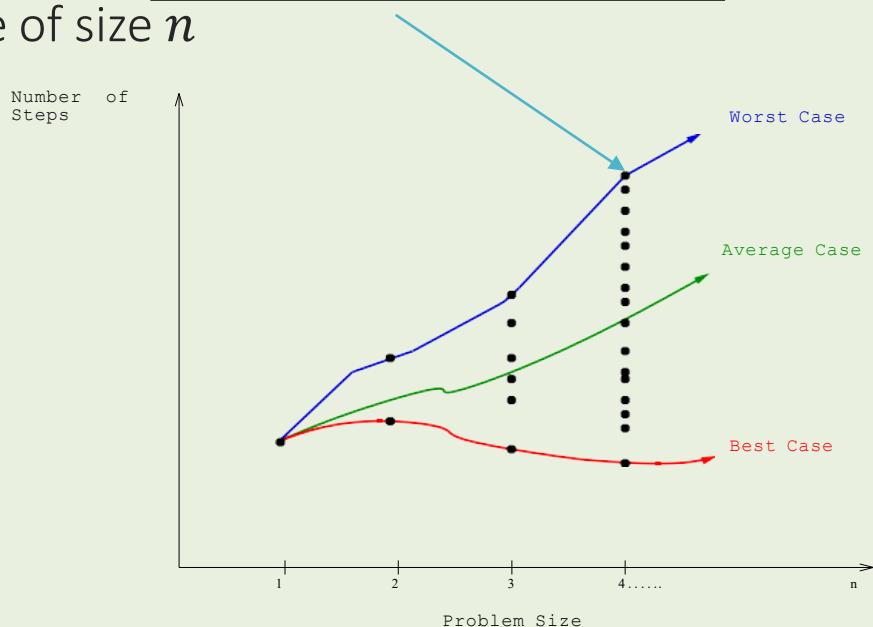
Chapter.2-Algorithm Analysis

The RAM Model of Computation

- Algorithms are an important and durable part of computer science because they can be studied in a machine/language *independent* way.
- This is because we use the **RAM model of computation** for all our analysis.(分析演算法用的隨機存取計算模型的基本假設)
 1. Each “simple” operation (+, -, =, if, call) takes 1 step.
 2. Loops and subroutine calls are *not* simple operations. They depend upon the size of the data and the contents of a subroutine. “Sort” is not a single step operation.
 3. Each memory access takes exactly 1 step.
- We measure the run time of an algorithm by counting the number of steps.
- This model is useful and accurate in the same sense as the flat-earth model!(地球是圓的但蓋房子還是視為平面來設計建造)

Worst-Case Complexity(最壞狀況複雜度)

- The *worst case complexity* of an algorithm is the function defined by the maximum number of steps taken on any instance of size n



Best-Case and Average-Case Complexity

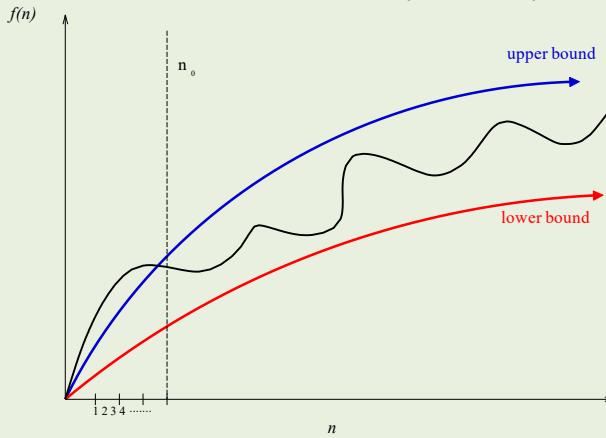
- The *best case complexity* of an algorithm is the function defined by the **minimum** number of steps taken on any instance of size n .
- The *average-case complexity* of the algorithm is the function defined by an **average** number of steps taken on any instance of size n .
- Each of these complexities defines a numerical function: time vs. size!(時間與資料大小的關係)

Our Position on Complexity Analysis

- What would the reasoning be on buying a lottery ticket on the basis of best, worst, and average-case complexity?
- Generally speaking, we will use the worst-case complexity as our preferred measure of algorithm efficiency.
- Worst-case analysis is generally easy to do, and “usually” reflects the average case. Assume I am asking for worst-case analysis unless otherwise specified! (通常以最壞狀況來分析)
- Randomized algorithms are of growing importance, and require an average-case type analysis to show off their merits. (平均狀況雖不好計算,但多用於分析隨機演算法)

Exact Analysis is Hard!

- Best, worst, and average case are difficult to deal with because the *precise* function details are very complicated:

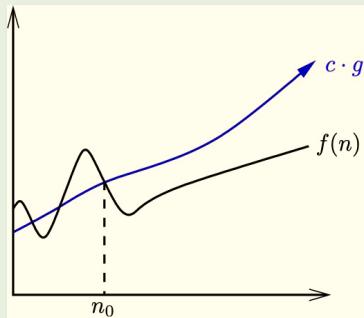


- It easier to talk about *upper and lower bounds* of the function. Asymptotic notation (O, Θ, Ω) are as well as we can practically deal with complexity functions. (以漸近表示法表達時間複雜度)

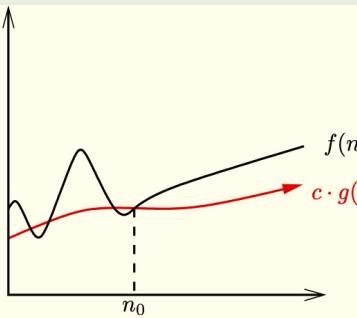
Names of Bounding Functions

- $f(n) = O(g(n))$ means $C \times g(n)$ is an *upper bound* on $f(n)$.(漸近上限)
- $f(n) = \Omega(g(n))$ means $C \times g(n)$ is a *lower bound* on $f(n)$.(漸近下限)
- $f(n) = \Theta(g(n))$ means $C_1 \times g(n)$ is an upper bound on $f(n)$ and $C_2 \times g(n)$ is a lower bound on $f(n)$, i.e., *tight bound*.(漸近緊約束)
- C , C_1 , and C_2 are all constants independent of n .

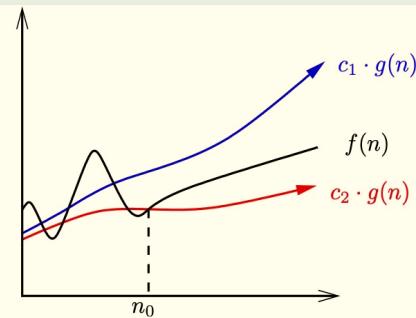
O , Ω , and Θ



(a)



(b)



(c)

- The definitions imply a constant n_0 beyond which they are satisfied. We do not care about small values of n .

Formal Definitions

- $f(n) = O(g(n))$ if there are positive constants n_0 and c such that to the right of n_0 , the value of $f(n)$ always lies on or *below* $c \cdot g(n)$.
- $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that to the right of n_0 , the value of $f(n)$ always lies on or *above* $c \cdot g(n)$.
- $f(n) = \Theta(g(n))$ if there exist positive constants n_0 , c_1 , and c_2 such that to the right of n_0 , the value of $f(n)$ always lies *between* $c_1 \cdot g(n)$ and $c_2 \cdot g(n)$ inclusive.

Big Oh Examples

- $3n^2 - 100n + 6 = O(n^2)$ because $3n^2 > 3n^2 - 100n + 6$,
when $n > 0.06$
- $3n^2 - 100n + 6 = O(n^3)$ because $.01n^3 > 3n^2 - 100n + 6$,
when $n > 300$
- $3n^2 - 100n + 6 \neq O(n)$ because $c \cdot n < 3n^2$, when $n > c$
- Think of the equality(=) as meaning *in the set of functions*.
 $n^2 = O(n^3)$ means n^2 is one of functions that are $O(n^3)$

Big Omega Examples

- $3n^2 - 100n + 6 = \Omega(n^2)$ because $2.99n^2 < 3n^2 - 100n + 6$,
when $n > 10000$
- $3n^2 - 100n + 6 \neq \Omega(n^3)$ because $3n^2 - 100n + 6 < n^3$,
when $n > 3$
- $3n^2 - 100n + 6 = \Omega(n)$ because $10^{10^{10}}n < 3n^2 - 100n + 6$,
when $n > 100 \cdot 10^{10^{10}}$

Big Theta Examples

- $3n^2 - 100n + 6 = \Theta(n^2)$ because O and Ω
 - $3n^2 - 100n + 6 \neq \Theta(n^3)$ because O only
 - $3n^2 - 100n + 6 \neq \Theta(n)$ because Ω only
-
- Is $2^{n+1} = \Theta(2^n)$?
 - Is $2^{n+1} = O(2^n)$? Can you find the c that makes $2^{n+1} \leq c \cdot (2^n)$?
 - Is $2^{n+1} = \Omega(2^n)$? Can you find the c that makes $2^{n+1} \geq c \cdot (2^n)$?

Big Oh Addition/Subtraction

- Suppose $f(n) = O(n^2)$ and $g(n) = O(n^2)$.
 - What do we know about $g'(n) = f(n) + g(n)$? Adding the bounding constants shows $g'(n) = O(n^2)$.
 - What do we know about $g''(n) = f(n) - |g(n)|$? Since the bounding constants don't necessarily cancel, $g''(n) = O(n^2)$
- We know nothing about the lower bounds on g' and g'' because we know nothing about lower bounds on f and g .

Big Oh Multiplication by Constant

■ Multiplication by a constant does **not** change the asymptotes:

- $O(c \cdot f(n)) \rightarrow O(f(n))$
- $\Omega(c \cdot f(n)) \rightarrow \Omega(f(n))$
- $\Theta(c \cdot f(n)) \rightarrow \Theta(f(n))$

■ The “old constant” C from the Big Oh becomes $c \cdot C$.

Big Oh Multiplication by Function

■ But when both functions in a product are increasing, both are important:

- $O(f(n)) \cdot O(g(n)) \rightarrow O(f(n) \cdot g(n))$
- $\Omega(f(n)) \cdot \Omega(g(n)) \rightarrow \Omega(f(n) \cdot g(n))$
- $\Theta(f(n)) \cdot \Theta(g(n)) \rightarrow \Theta(f(n) \cdot g(n))$

■ This is why the running time of two nested loops is $O(n^2)$.



Python 片段程式》

矩陣相加

執行次數

```
def add(a, b, c, n):
    for i in range(n):
        for j in range(n):
            c[i][j] = a[i][j] + b[i][j]
```

$n+1$
 $n(n+1)$
 n^2

$2n^2+2n+1$



Exercise

Problem of the Day

■ What is the result of the execution of the program shown in the figure? That is, how many times is the statement `x = x + 1` executed when $n = 10$?

■ How many times is the `x=x+1` statement executed in the program shown in the figure?

■ For each of the following pairs of functions $f(n)$ and $g(n)$, state whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$, or none of the above.

1. $f(n) = n^2 + 3n + 4$, $g(n) = 6n + 7$

2. $f(n) = n \cdot \sqrt{n}$, $g(n) = n^2 - n$

3. $f(n) = 2^n - n^2$, $g(n) = n^4 + n^2$

```
n=10
x=0
for i in range(1, n+1):
    k=i+1
    while k <= n:
        x=x+1
        k+=1
    print(x)
```

```
for i in range(n):
    for j in range(n):
        x=x+1
```

Program Exercises (Moodle CodeRunner)

■ Exercise 01 (close at 3/4 23:59)

1. **Bits Counting**: Given a positive integer n , please count the number of bits that are set to 1 in n , and then return the result. E.g. $n=5 \rightarrow 2$; $n=15 \rightarrow 4$
2. **Remove element**: Given a list $nums$ and a value val , remove all instances of that value in-place and return the new length of $nums$ list. E.g. $nums=[3,2,2,3]$, $val=3$ return 2; $nums=[2, 2]$
3. **The 3n+1 Problem**: Consider the following algorithm:
 - 1. input n
 - 2. print n
 - 3. if $n = 1$ then STOP
 - 4. if n is odd then $n = 3n + 1$
 - 5. else $n = n/2$
 - 6. GOTO 2

Given the input 22, the following sequence of numbers will be printed
(22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1), length is 16.

For any two numbers i and j you are to determine the **maximum cycle length** over all numbers between and including both i and j . For example, the largest cycle length between 1 and 10 is 20.