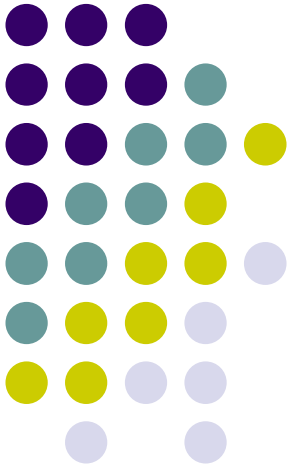
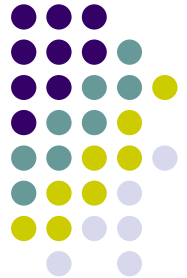
Python程式設計入門 基礎(1/2)

葉難

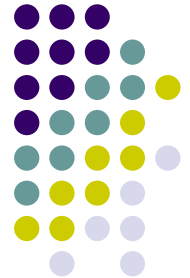


大綱



- 物件、型別、名稱、指派（assignment）
- 運算式、運算元、運算子
- 述句（statement），條件判斷、迴圈等
- 函式、參數、回傳、範圍
- 異常（exception）
- 模組、匯入、標準程式庫（內建模組）

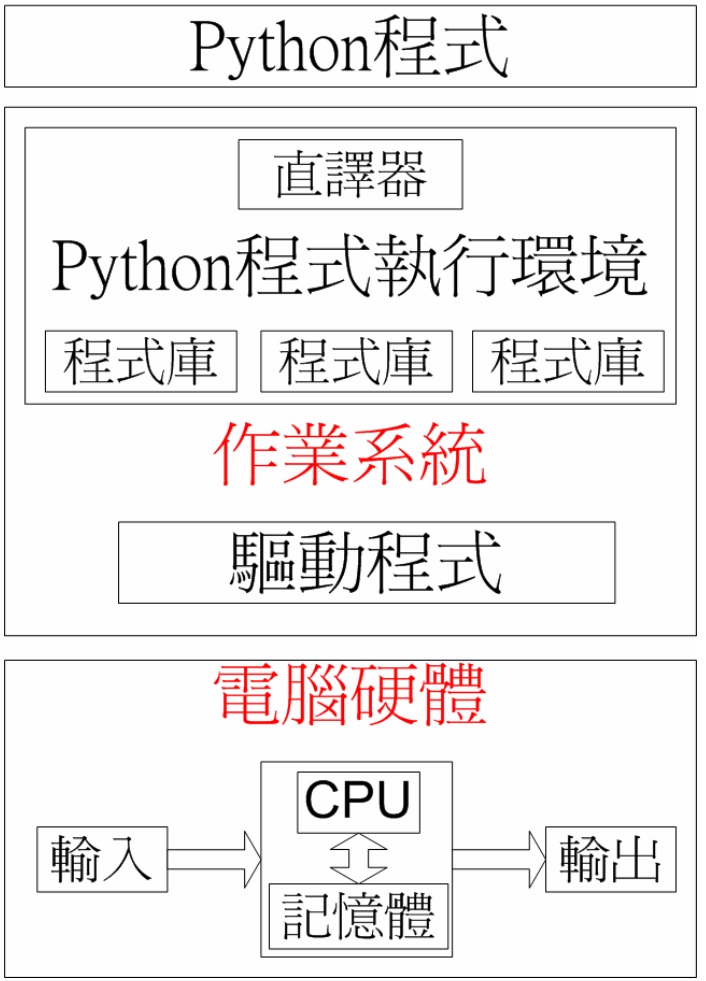
語言、語法、語意



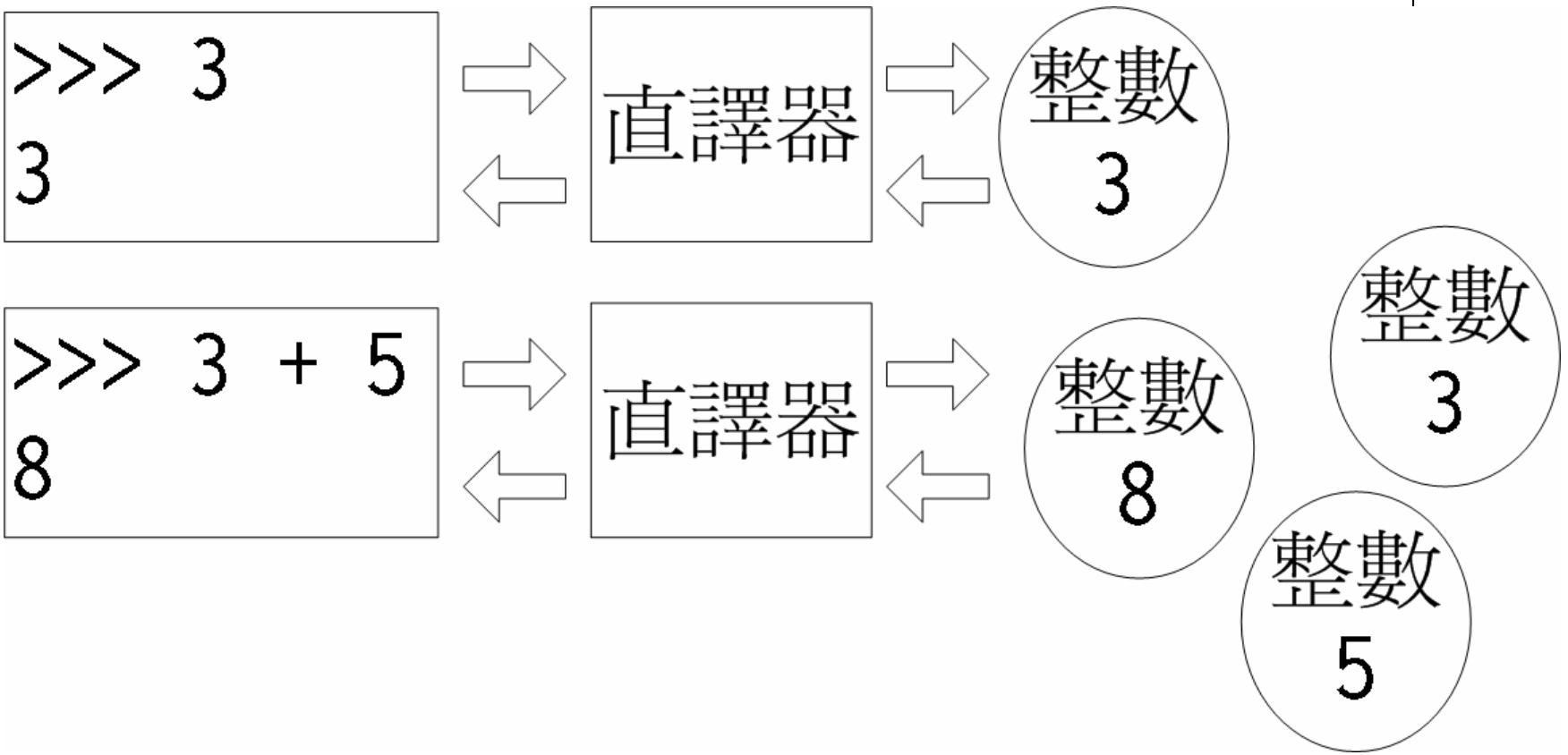
- 語法（syntax）：語言的書寫規則
- 語意（semantics）：句子（程式碼）的意義，電腦該執行什麼動作
- 不合語法：「apple good is」
- 合語法但無語意：「colorless green」
- 程式語言擁有嚴格定義的語法，以及其相對應的語意

抽象模型

- 把底下的東西視爲黑盒子
- 以少數概念規則、解釋底層各種複雜現象
- 系統在腦袋裡的想像模樣
- Python程式語言的抽象模型：電腦如何執行程式是一回事、你如何理解程式是另一回事
- 因間隔導致上下不一致：錯誤理解

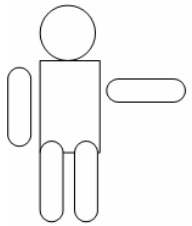


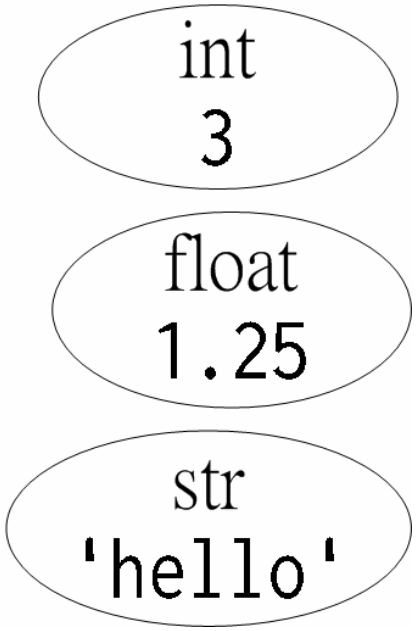
建立物件、簡單運算



物件皆有型別（**type**）

```
>>> 3
3
>>> 1.25
1.25
>>> 'hello'
'hello'
```

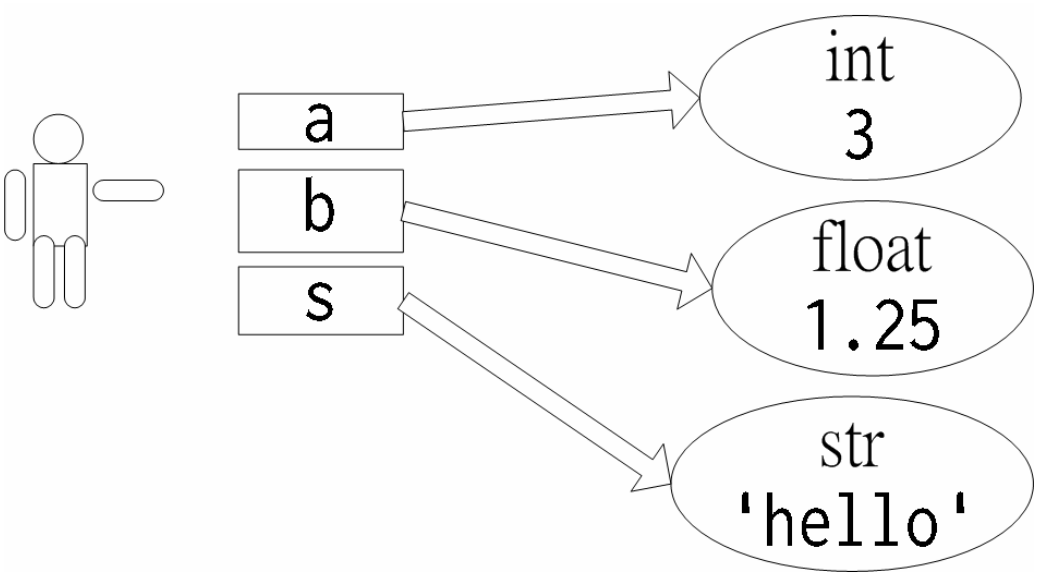




- 型別int（整數）、float（浮點數）、str（字串）

名稱、指派（**assignment**）

```
>>> a = 3
>>> b = 1.25
>>> s = 'hello'
```



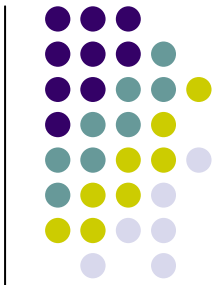
- 「=」指派運算子，建立名稱與物件之間的綁定關係
- 先執行右邊的運算式，再讓名稱指向運算結果（物件）
- 型別跟著物件，不是跟著名稱；名稱與物件相互獨立

小故事：等號「=」

- 1557年數學家Robert Recorde才發明等號
- 因爲重複書寫「等於」，他終於忍無可忍
- 呃，那以前的數學家呢？

- 數學等號和程式語言的指派運算子，雖有相似之處，但最好當成不同的東西

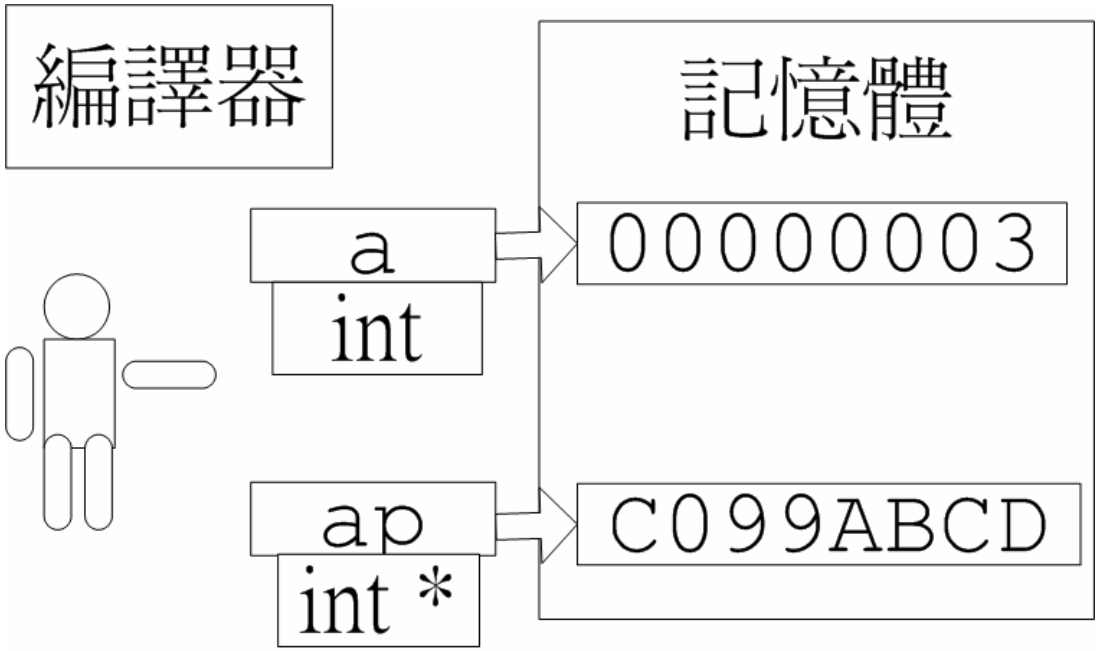
命名規則



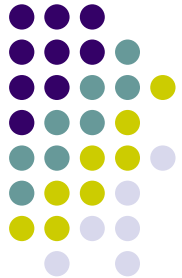
- 名稱，或稱識別字、識別名
- 可使用英文字母小寫a到z、大寫A到Z、數字0到9、底線「_」
- 第一個字不能是數字
- 區分大小寫，makewish與makeWish代表不同的名稱
- 保留字（關鍵字）：Python已賦予特定意義
- 命名慣例

在Python裡，什麼都是物件

- Python都是物件，或說是把任何資料包裝成物件，以型別區分
- 其他語言，如C++與Java，有分一般變數、指標（指位器）、參考（reference）等



型別**list**：容器

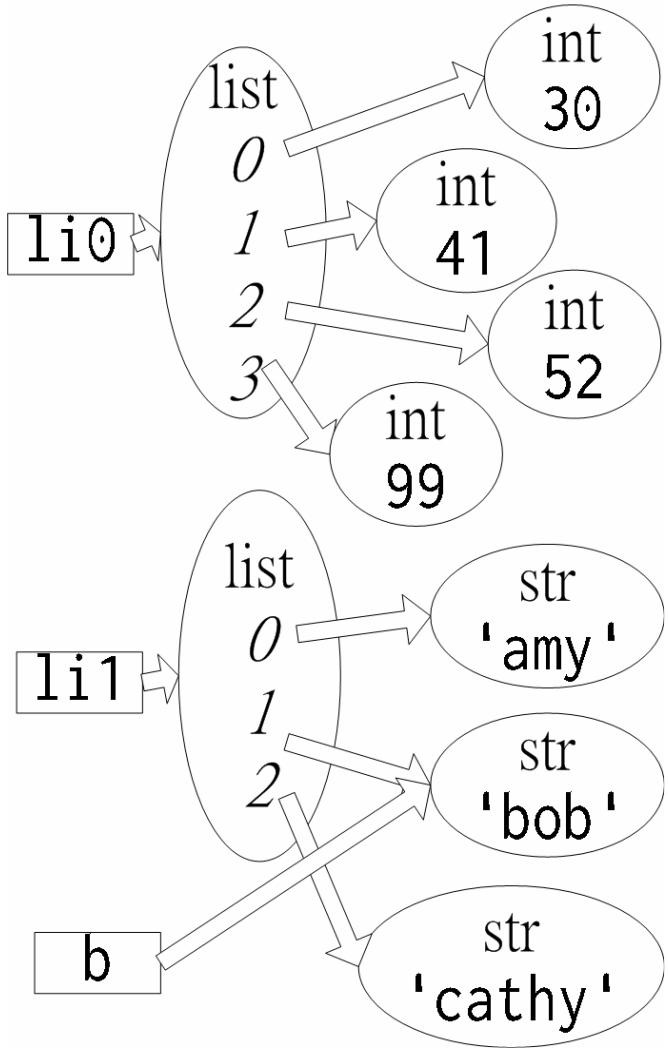


- 以方括號「[、]」來建立，逗號「,」隔開元素
- 存取時也用方括號：索引（indexing）
- 索引值從0開始起跳
- 索引值超過的話，會出現錯誤
- 裡頭放的是綁定關係，並非直接儲存物件

- list內可放入任何物件（應說是綁定關係），所以也可放入list物件

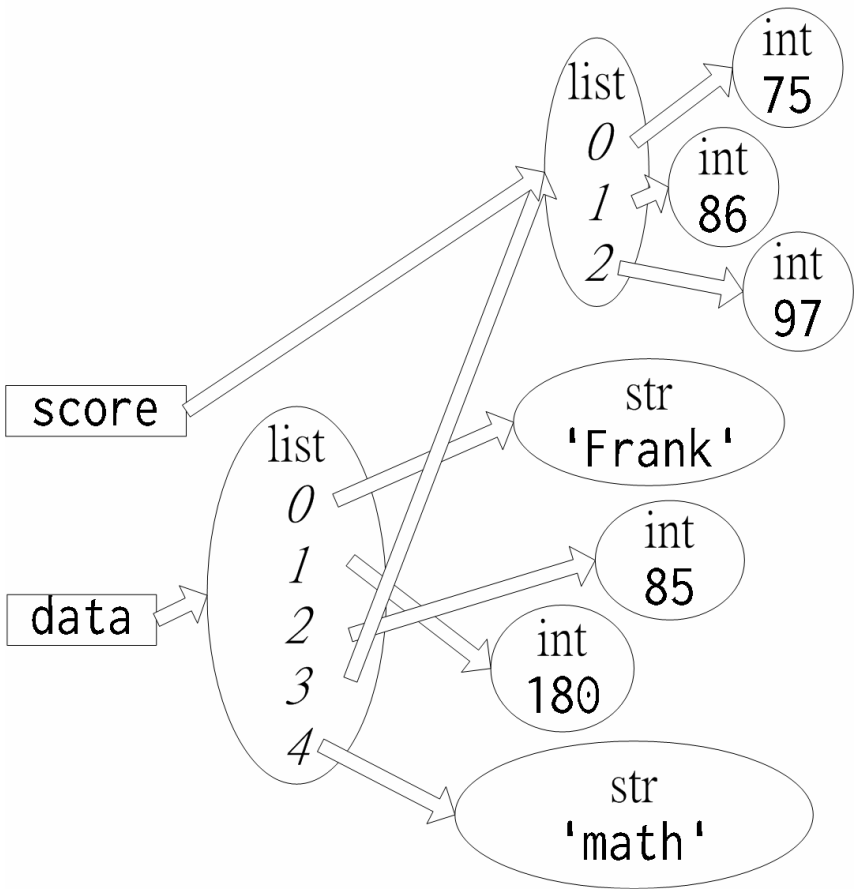
list範例

```
>>> li0 = [30, 41, 52, 63]
>>> li0[2]
52
>>> li0[3] = 99
>>> li1 = ['amy', 'bob', 'cathy']
>>> b = li1[1]
>>> b
'bob'
```

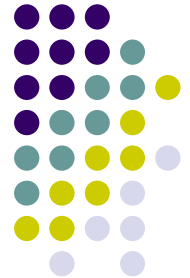


list內可含有list：二維結構

```
>>> score = [75, 86, 97]
>>> data = ['Frank', 180, 85, score, 'math']
>>> data[3]
[75, 86, 97]
>>> data[3][0]
75
```



可變、不可變

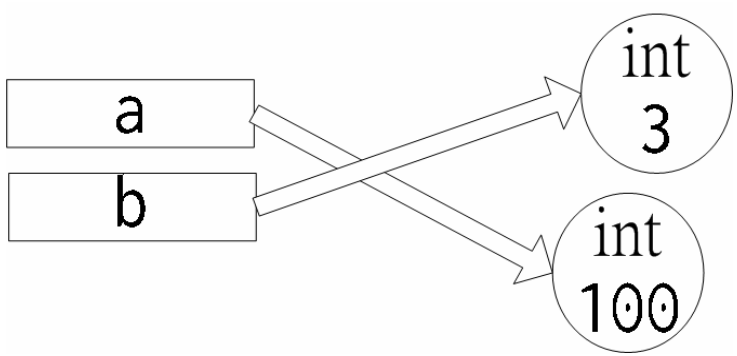


- 不可變（mutable）：建立之後，其值就不會改變。如「3 + 5」會建立新int物件
- 不可變： int、float、str、tuple
- 可變（immutable）：建立後，其內容仍可改變
- 可變：list

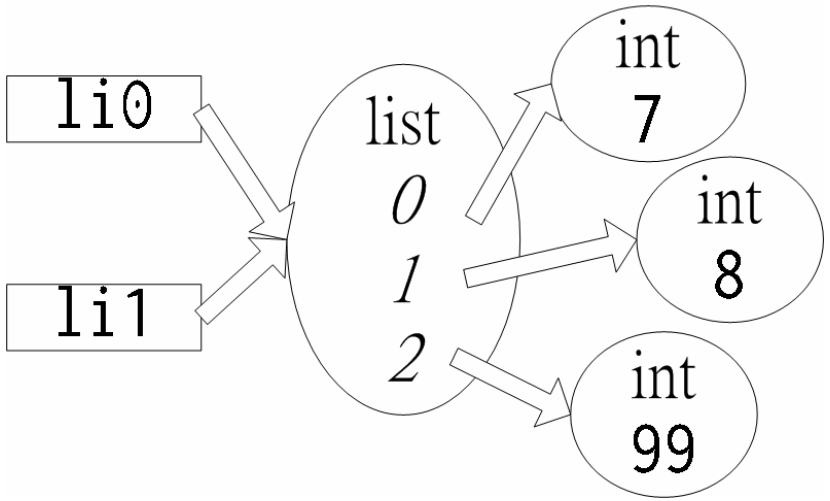
- 要注意「別名（alias）」現象

別名：多個名稱指向同一物件

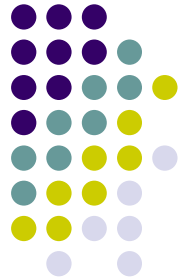
```
>>> a = 3
>>> b = a
>>> a = 100
>>> b
3
```




```
>>> li0 = [7, 8, 9]
>>> li1 = li0
>>> li0[2] = 99
>>> li1
[7, 8, 99]
```

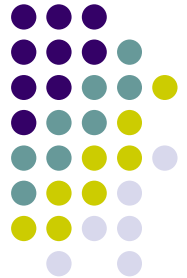


tuple（元組）：不可變的list



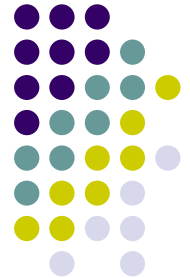
- 語法和list幾乎一樣，使用小括號「(、)」來建立，存取時仍使用方括號「[、]」
- 不可變，所以若嘗試修改，會出現錯誤
- 若想建立僅含一個元素的tuple，需多個逗號，如「('hi',)」
- 因爲小括號在Python裡具備多重用途（意義）：tuple、運算式括號、函式呼叫、產生器運算式等

tuple範例



```
>>> t0 = (30, 41, 52)  # 井字號代表註解
>>> t0[2]  # 井字號後面的東西會被忽略
52
>>> t0[0] = 99              # 不支援變更的動作
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> ()  # 空tuple
()
>>> t1 = ('hi', )  # 只含一個元素
```

運算式（**expression**）

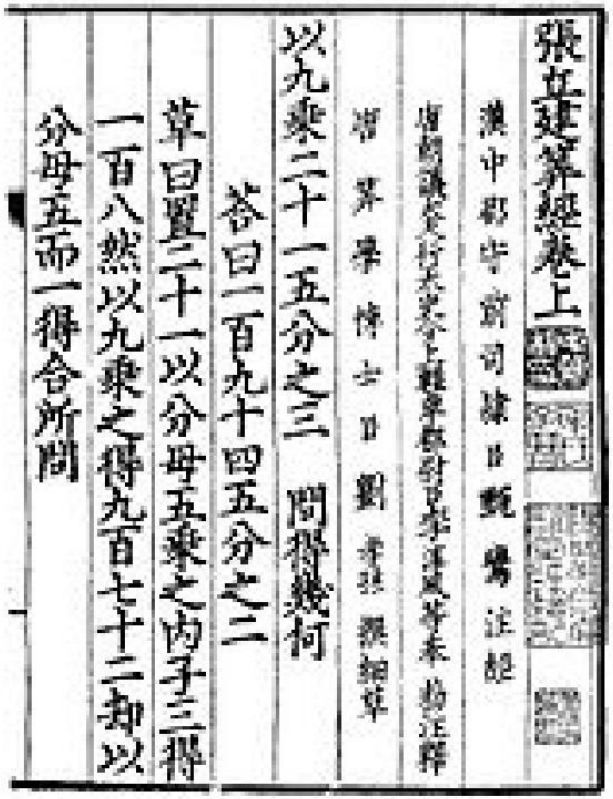


- 由運算元（operand）和運算子（operator）組成
- 運算元：物件，可以是字面值（literal）形式（直接寫在程式裡）、或經由名稱取得、或是其他運算式的結果
- 運算子：分別代表不同的動作，如加法、比較大小、邏輯運算、位元運算、檢查list裡是否含有某元素等等

數學運算式

$$\pi = \frac{426880\sqrt{10005}}{\sum_{n=0}^{\infty} \frac{(6n)!(545140134n+13591409)}{(n!)^3(3n)!(-640320)^{3n}}}$$

$$\sqrt{\frac{(x^2+x-4)\sin\alpha}{\nabla x\beta\left(\sqrt{\frac{\alpha+jx}{\beta+3}}\right)^2}} + \int_{x=0}^{N-1} \cos\left(\frac{x}{\beta}\right) dx$$


張丘建算經卷上

漢中郡守前司隸校尉甄鸞注述

唐朝議大夫太史令上輕車都尉賜紫金魚袋臣李淳風等奉 勅注釋

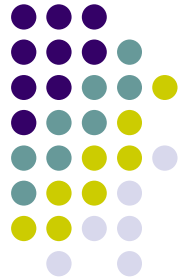
增算學博士臣劉孝孫 錄細草

以九乘二十一五分之三 問得幾何

荅曰一百九十四五分之三

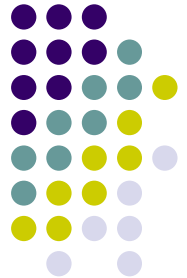
草曰置二十一以分母五乘之內子三得一百八然以九乘之得九百七十二却以分母五而一得合所問

運算子(1/2)



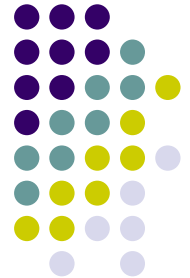
- 算術運算子「+、-、*、/、//、%、**、@」
- 正負號「+、-」
- 比較運算子「<、<=、>、>=、!=、==」
- 成員關係運算子「in、not in」
- 同等關係運算子「is、is not」
- 邏輯運算子「and、or、not」
- 位元運算子「&、^、|、~、<<、>>」

運算子(2/2)



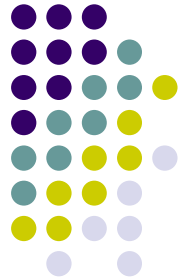
- (expr…)、[expr…]、{key:value…}、`expr...`、{expr…}
- x[index]、x[index:index]、x(arguments…)、x.attribute
- 條件運算子「if-else」
- lambda運算子「lambda」
- 運算子優先順序與結合順序

算術運算子



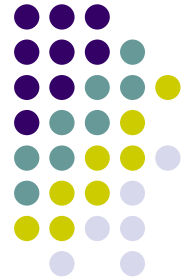
- int（整數）的加法「+」、減法「-」、乘法「*」、冪次方「**」，皆無問題
- 但除法「/」非封閉運算，「7 / 3」得到「2.3333333333333335」非整數
- 地板除法「//」
- 餘數「x%y」，行爲由「x - (x//y) * y」定義
- 優先順序：先乘除後加減
- 使用小括號「(、)」改變運算順序

「+」、「-」、「*」、「**」



```
>>> 3 + 5 - 2
6
>>> (180 - 80) * 0.7   # 男性標準體重算法
70.0
>>> 3.14 * 10 * 10     # 圓面積
314.0
>>> 3 ** 2 + 4 ** 2    # 畢氏定理，未完
25
>>> -5 * 3 + 6 - (2 ** 3)
-17
```

問題：我愛你一生一世的十倍

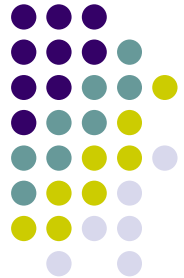


```
>>> 5201314   # 男孩說：我愛你一生一世
5201314
>>> (520 + 1314) * 10   # 女孩回答
```

結果是？

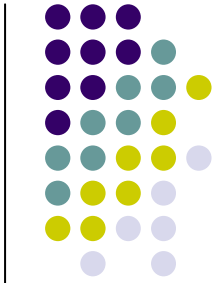
- 男孩以爲是十倍
- 其實是「18340」，意思「一巴掌搧死你」

「**」的結合順序：從右到左



```
>>> 2 ** 3 ** 2
512
>>> 2 ** (3 ** 2)
512
>>> (2 ** 3) ** 2
64
```

str與list的「+」與「*」

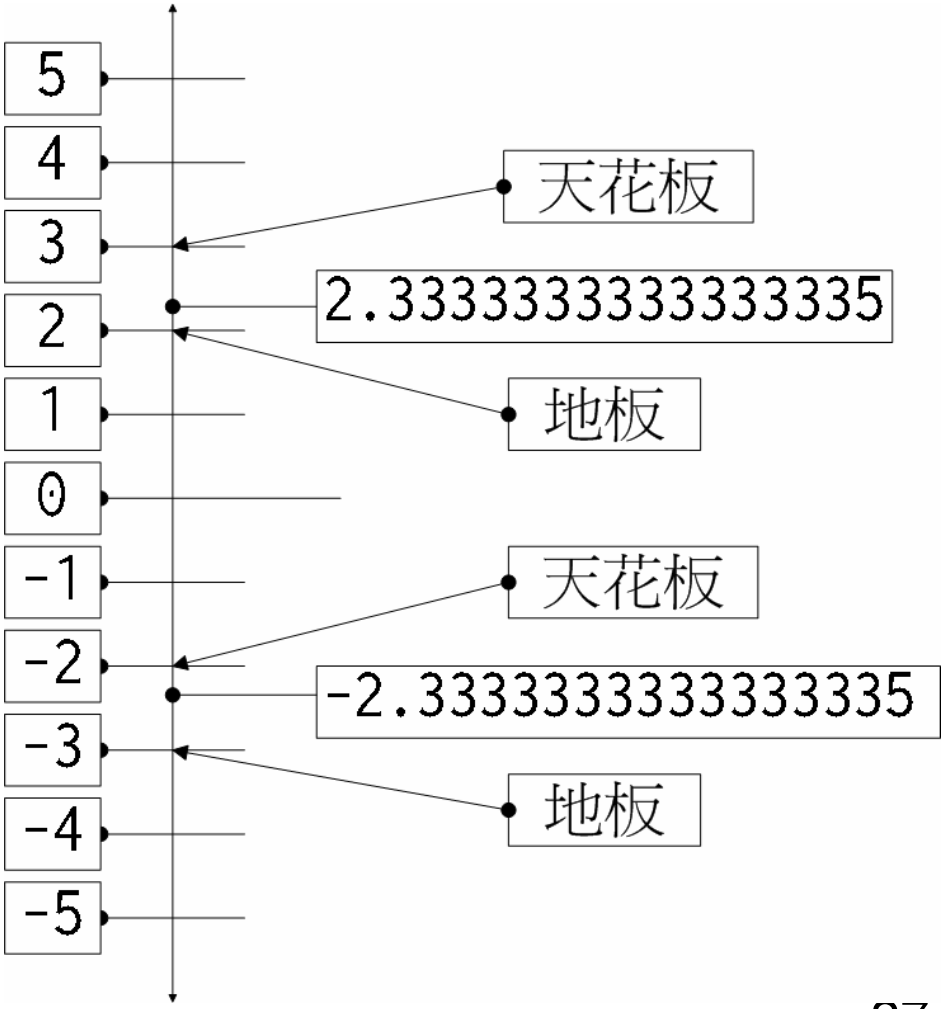


```
>>> 'Hello' + ', ' + 'Python!'
'Hello, Python!'
>>> 'banana' * 4
'bananabananabananabanana'
>>> '-' * 20
'--------------------'
>>> [1, 2, 3] + [4, 5]          # 串列加法
[1, 2, 3, 4, 5]
>>> [1, 2, 3] * 3               # 串列乘法
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

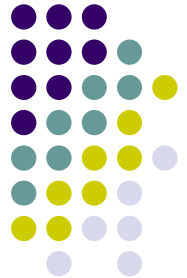
「//」：地板除法

```
>>> 7 // 3
2
>>> -7 // 3
-3
>>> 7.0 // 3
2.0
```

- 天花板（ceiling）
- 地板（floor）



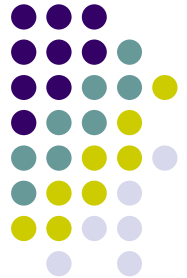
問題：Unix時間



- 某些Unix系統採用32位元有號整數，記錄自1970年1月1日0時0分0秒以來的秒數
- 請問到哪一年會爆掉？

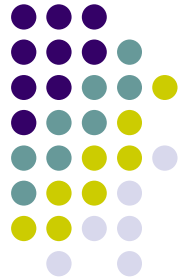
```
>>> 2 ** 31                    # 最多可表達這麼多秒
2147483648
>>> (2 ** 31) // 60 // 60 // 24 // 365 + 1970
2038
```

「/」和「%」



```
>>> 17 / 3                  # 運算元是int
5.666666666666667           # 結果型別是float
>>> -17 / 3
-5.666666666666667
>>> 17 % 3
2
>>> -17 % 3                 # 負數，需小心
1
>>> -17 - (-17//3) * 3
1
```

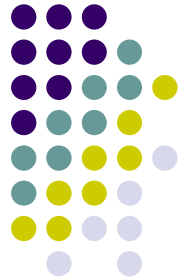
2.x版差異：運算子「/」



```
>>> 17 / 3
5
>>> 17.0 / 3
5.666666666666667
```

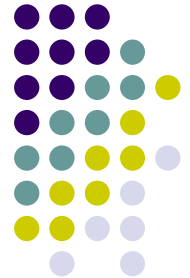
- 2.x版的「/」，運算元若是int，則爲地板除法，運算元若是float，則爲一般除法
- 2.x版分爲int與long，數值超過界限就會自動轉成long，所以一般來說毋須擔心

運算子的意義，由運算元型別決定



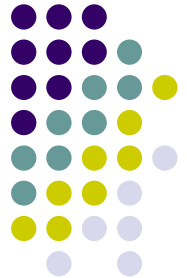
- 運算元若是int，「+」代表加法
- 運算元若是str，「+」代表連接字串
- 運算元若是list，「+」代表串連兩個list物件、產生新list物件
- 所以，運算子本身就只是個代表符號罷了

型別bool（布林）、True、False



- 比較運算子「<、<=、>、>=、!=、==」、成員關係運算子「in、not in」、同等關係運算子「is、is not」
- 運算結果產生的物件，型別爲bool
- 只有兩種值：True、False，已內建
- 邏輯運算子「and、or、not」

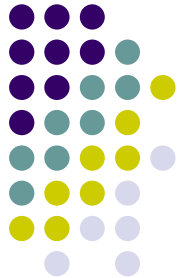
比較運算子
「<、<=、>、>=、!=、==」



- 比較大小，意義由型別決定

```
>>> a = 11
>>> b = 23
>>> a < b        # a大於b嗎？
True             # 是、真
>>> a != 11      # a不等於11嗎？
False            # 錯、假
>>> b == 23      # b等於23嗎？
True
```

比較其他型別

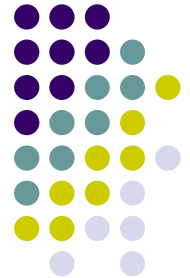


```
>>> li0 = [31, 42, 53]
>>> li1 = [42, 53]
>>> li0 == li1              # 比較兩個串列
False
>>> li0 == [31] + li1       # 串列的「+」
True
>>> 'abc' == 'abc'          # 比較兩個字串
True
>>> 'Amy' < 'Bob'           # 什麼意思？
True
```

成員關係運算子「in、not in」，檢查某物件（元素）是否在容器裡

```
>>> a = 4
>>> li = [31, 4, 53, 64]
>>> a in li          # a指向的物件是否存在於li裡
True
>>> 75 in li         # li裡有int物件75嗎
False
>>> 4 not in li
>>> False
>>> 'you' in 'How are you?'   # 可把字串當容器
True
```

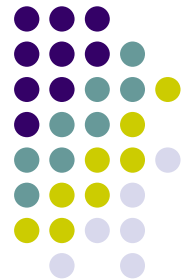
同等關係運算子「is、is not」，檢查兩個名稱是否指向同一個物件



```
>>> a = 3
>>> b = 4
>>> a is b
False
>>> a is (1 + 2)
True   # 注意
>>> a = 2**32
>>> a is (2**32)
False   # 注意
>> a == (2**32)
True
```

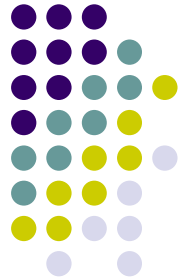
```
>>> t0 = (3, 4, 5)
>>> t1 = t0
>>> t0 is t1
True
>>> t2 = (3, ) + (4, 5)
>>> t0 is t2
False   # 不是同一個
>>> t0 == t2
True    # 但內容相同
```

邏輯運算子「and、or、not」，結合多個比較運算子



```
>>> x = 75
>>> 0 <= x and x <= 100   # 且
True
>>> x < 0 or 100 < x      # 或
False
>>> result = x > 90       # 運算結果是物件，
>>> result                # 可指派給某名稱
False
>>> not result            # 反轉真假值
True
```

比較運算子的簡短寫法



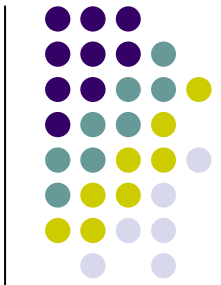
```
>>> x = 75
>>> 0 <= x and x <= 100
True
>>> 0 <= x <= 100          # 簡短寫法
True
y = 100
>>> x == 75 < y            # 有時意思不明顯
True
>>> x == 75 and 75 < y
True
```

問題：閏年（公元年份）

- 可被4整除的年份，是閏年
- 以上年份，若可被100整除，則不是閏年
- 以上年份，若可被400整除，則是閏年
- 閏年：1600, 1992, 1996, 2000, 2060, 2400
- 不是閏年：1800, 1900, 2100, 1997, 2057

```
>>> y = 2000
>>> (y % 4 == 0) and (y % 100 != 0) or (y % 400 == 0)
True
>>> y % 400 == 0 or y % 4 == 0 and y % 100 != 0
True
```

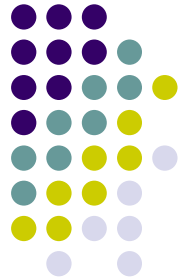
真假值一般化



- 除了True與False，其他物件也可作爲真假值
- None、0、0.0、空字串「"」、空list或空tuple（或空的容器），會被當做「False」（假）
- 其餘皆會被當做「True」（真）

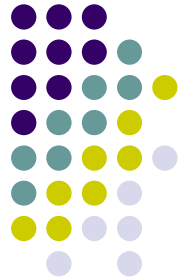
- 另外要注意邏輯運算式的「短路（short-circuit）」行爲

範例



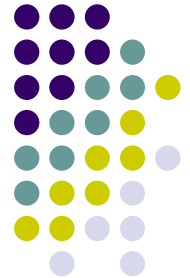
```
>>> li0 = []
>>> li1 = [61, 72, 83]
>>> li0 and li0[0] > 60   # 短路行爲
[]                       # 根本不會執行後半部
>>> li1 and li1[0] > 60
True
>>> [] or 0 or '0'   # 由最後的'0'決定真假值
'0'
>>> not '0'          # '0'爲真，反轉後爲假
False
```

型別NoneType，唯一值None



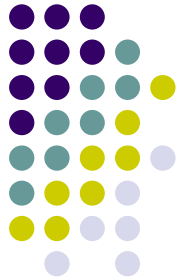
- None用來代表：沒有、尚未定義、無效
- 型別爲NoneType
- None是系統預設建立的名稱
- 有時需要先定義名稱，但尚未具備有效值，可以None作爲其值

述句（**statement**）



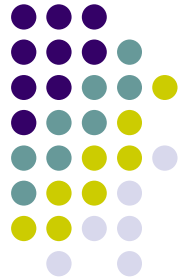
- Python程式的基本執行單位是「模組」，簡單地說就是Python程式檔案
- 最小的執行單位是「述句」，述句可由分號「;」隔開，但非必要
- 每種述句皆有其特定語法和語意
- 述句含有「運算式」
- 運算式建立與操作「物件」

述句種類（1/2）



- 運算式述句，所以才能執行「運算式」
- 「=」指派述句（assignment）
- if：條件判斷述句
- while、for，迴圈述句
- continue、break述句
- def：函式定義述句
- del、global、nonlocal：名稱相關述句

述句種類（2/2）

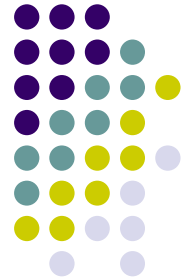


- try、raise、assert：異常相關述句
- with：文脈述句
- yield：產生器相關述句
- class：類別定義述句
- import：模組匯入述句
- pass：什麼也不做的述句
- exec、print述句（2.x版）

指派述句、分號「;」

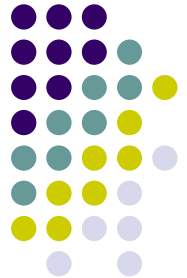
```
>>> a = 3; b = 4; c = 5;     # 分號代表述句結尾
>>> a, b, c                  # tuple的小括號，
(3, 4, 5)                    # 有時可省略
>>> (a, b, c)
(3, 4, 5)
>>> x = y = z = [0, 1, 2]    # 多重指派
>>> y[0] = 99
>>> z = ['hi', 'hello']      # 改爲指向另一個物件
([99, 1, 2], [99, 1, 2], ['hi', 'hello'])
```

序列指派



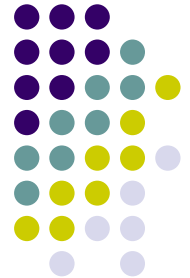
```
>>> a, b, c = 3, 4, 5        # 以下三行意思相同
>>> (a, b, c) = (3, 4, 5) # 但若左右個數不一樣，
>>> a, b, c = [3, 4, 5]    # 會出現錯誤
>>> a, b, c
(3, 4, 5)
>>> a, b, c = 'xyz'    # 把字串當容器
>>> a, b, c
('x', 'y', 'z')
>>> a, b = b, a          # 互換，很方便
>>> a, b
('y', 'x')
```

星號序列指派



```
>>> data = ['Amy', 170, 52, 23, ('C', 'Python' )]
>>> name, h, w, age, (p0, p1) = data
>>> name, *n, (p0, p1) = data
>>> n
[170, 52, 23]
>>> name, *n, (*p, ) = data
>>> n, p
([170, 52, 23], ['C', 'Python'])
```

增強型指派述句

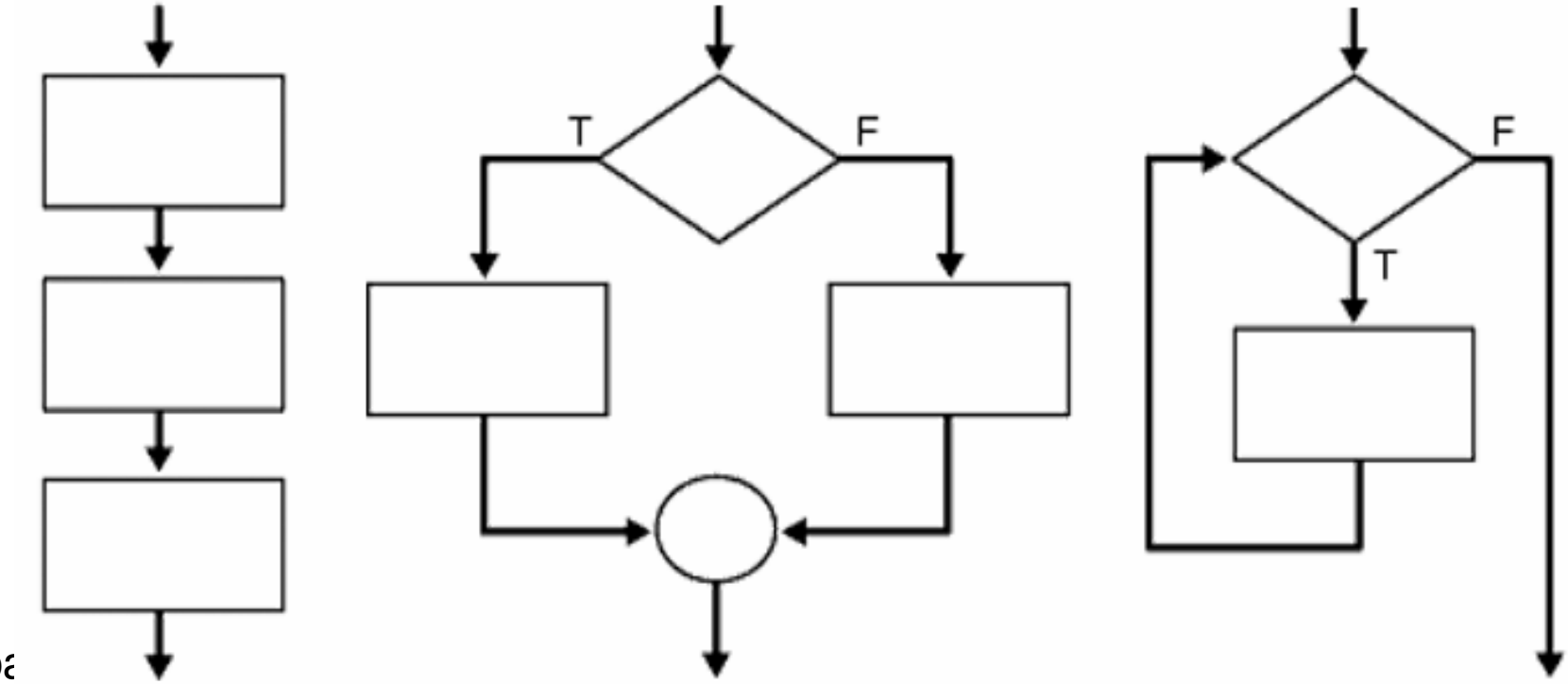


- +=、-=、*=、/=、//=、%=、**=、>>=、<<=、%=、^=、|=
- 「x += y」，若左邊物件不可變，則等同於「x = x + y」；若可變，則爲原地（in-place）修改

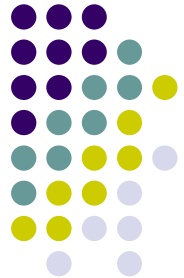
```
>>> x = 3                          # int不可變
>>> x += 5; x
8
>>> x = y = [0, 1, 2]              # list可變
>>> x += [3]; x
[0, 1, 2, 3]
>>> x is y
True
```

三種基本程式執行流程

- 循序：一個接著一個
- 條件判斷（分支、選擇）
- 迴圈（重複、迭代）



if條件判斷述句

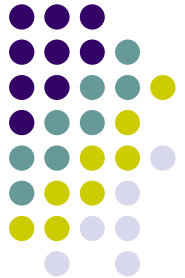


- 語法

```
if 運算式:
    述句…
elif 運算式:    # 子句
    述句…
elif 運算式:    # 子句
    述句
else:           # 子句
    述句…
```

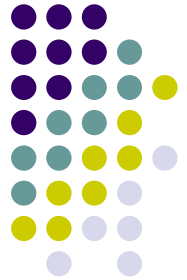
- 運算式的結果會被當做真假值，決定程式流程
- 子句屬於if述句的一部分，不可獨立存在；選用性
- He is one of the few boys who are liked by many.
- 別忘記冒號「:」
- 縮排、程式區塊

if範例



```
>>> result = ''; t = 28
>>> if t >= 30:
...     result = 'hot'      # 熱
... elif t >= 18:
...     result = 'normal'   # 一般
... else:
...     result = 'cold'     # 冷
...
>>> result
'normal'
```

縮排

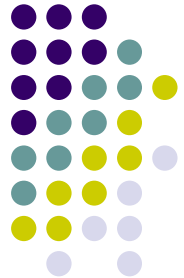


- 其他語言，如C/C++/Java/JavaScript，使用大括號「{、}」標示程式區塊，Python使用縮排
- 程式碼簡潔俐落，但有些人不喜歡
- 一層縮排可以是兩個空格、四個空格、一個TAB、或其他混合形式；重點是維持一致
- 建議：四個空格
- 注意：有些編輯器會轉換空格與TAB，不同編輯器對TAB的處理方式可能不同

找出三數字之最大的那一個

```
a, b, c = 3, 5, 7; x = None
if a < b:                  # 縮排零層
    if b < c:                  # 縮排一層
        x = c                      # 縮排二層
    else:                      # 縮排一層
        x = b                      # 縮排二層
else:                      # 縮排零層
    if a < c:                  # 縮排一層
        x = c                      # 縮排二層
    else:                      # 縮排一層
        x = a                      # 縮排二層
```

if-else運算式



- 語法，x、C、y皆爲運算式，若C爲真，則結果爲x，若C爲假，則結果爲y

x if C else y

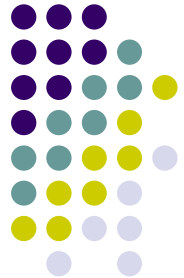
- 範例

```
bigger = x if x > y else y

n = 0 if n == None else n    # 賦予預設值
```

- 找出三數字之最大的那一個

```
(c if b < c else b) if a < b else (c if a < c else a)
```

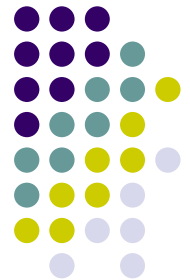
pass述句：什麼也不做



- 爲了符合語法，不能空著
- 有時尚無東西可寫，先以pass頂替
- print函式，可印出字串或其他物件

```
a, b, c = 0, '', []
if a or b or c:
    print('hi')
else:
    pass
```

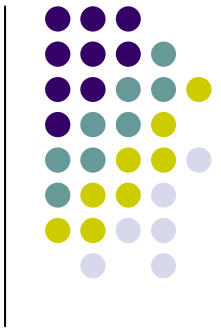
del述句：刪除名稱，連帶刪除與物件之間的綁定關係



- 物件只要有名稱指向它，就會「存活」
- 若無，則變成垃圾被系統自動回收
- 垃圾收集機制（garbage collection）

```
>>> a = 3
>>> del a
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

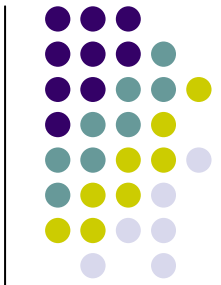
while迴圈述句



- 重複執行相同的程式碼
- 迴圈頭的運算式若爲假，離開迴圈；若爲真，進入迴圈體執行，執行完畢後再次執行運算式，重複上述動作
- 語法

```
while 運算式:    # 迴圈頭（head）
    述句          # 迴圈體（body）
    述句…
```

while範例



- 從1累加到100

```
i = 1
x = 0
while i <= 100:
    x += i
    i += 1
print(x)
```

- 總分與平均

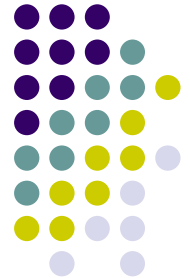
```
scores = [60, 81, 72, 45]
i = 0
total = 0
while i < 4:
    total += scores[i]
    i += 1

avg = total / 4
print(total, avg)
```

for迴圈述句

- 特製化、更方便的迴圈
- 所謂「能提供一串東西的物件」，目前僅知是list、tuple、str，後續再詳細介紹
- 語法

```
for 名稱 in 能提供一串東西的物件:
    述句
    述句…
```

for範例



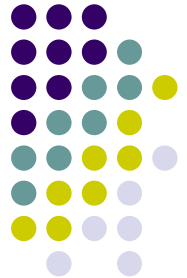
- 總分與平均

```
scores = [60, 81, 72, 45]
total = 0

for x in scores:
    total += x

avg = total / 4
print(total, avg)
```

- 多個名稱

```
data = (('Amy', 60), ('Bob',
  81), ('Cay', 45))

for a, b in data:
    print(a, b)
```

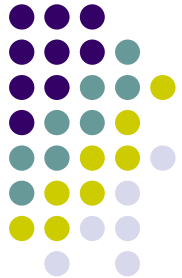
內建函式range



- range(3)產生出物件，能提供0、1、2
- range(6, 10)產生出物件，能提供6、7、8、9
- range(1, 10+1, 2)產生出物件，能提供1、3、5、7、9
- 九九乘法表，雙重迴圈

```
for x in range(2, 9+1):
    for y in range(1, 9+1):
        print(x, y, x*y)
```

2.x版差異



- 3.x版的range，等同於2.x版的xrange
- 2.x版的range，會產生list物件

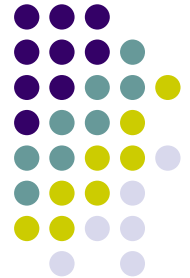
- 3.x的range，會產生某種特殊物件，符合「能提供一串東西的物件」的要求

問題：撲克牌總和

- A是1，J是11，Q是12，K是13
- 請問一副撲克牌（52張牌），總和多少？

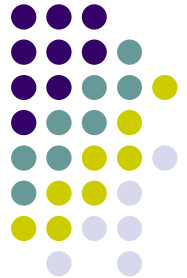
```
total = 0
for x in range(1, 13+1):
    total += x
total *= 4
print(total)
```

break述句：
立即跳出迴圈（最近的那一層）



```
scores = (98, 87, 59, 64, 77)
all_pass = True              # 預設All Pass
for x in scores:
    if x < 60:               # 只要有一科不及格，
        all_pass = False # 就不是All Pass
        break

print(passed)
```

continue述句：跳到下一輪



```
scores = (55, 73, 81, 65, 95, 34)
n = 0
high_total = 0
for x in scores:
    if x < 60:
        continue
    n += 1
    high_total += x
high_total /= n
```

Q&A