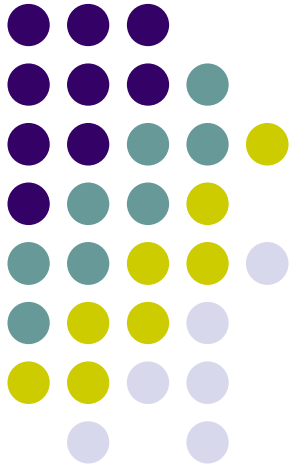
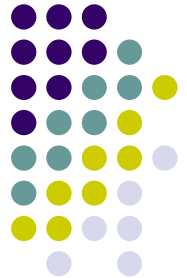
Python程式設計入門
容器-序列(1/2)

葉難

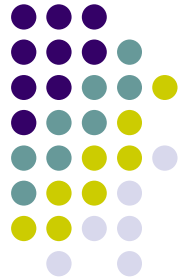


大綱



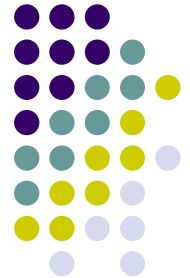
- Sequence（序列）抽象型別
- list（串列）， tuple（元組）， str（字串）
- 迭代（iteration）協定、可迭代者（Iterable）、迭代器（Iterator）
- 串列生成式（list comprehension）
- 產生器運算式（generator expression）

容器



- 資料結構
- 形式：序列（線性、一個接一個）、映射（對應關係）、樹狀
- 讀（取用、拿出來）、寫（存入，放進去）
- 尋找、排序、分類、過濾等動作

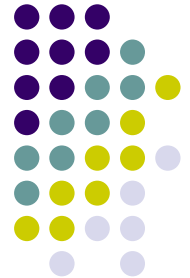
物件、型別、方法



- 物件皆有型別，由型別定義其能力、性質
- 3.x版整合型別（type）與類別（class）
- 例如：tuple是不可變的，可使用[]索引語法讀取
- 例如：list是可變的，擁有方法append
- 各型別、物件擁有不同的屬性項（attribute）

```
>>> li = li[0, 1 2]; type(li)
<class 'list'>
>>> dir(li)
['__add__', '__class__', '__contains__',
  ['__add__', '__class__', ...省略...]
```

在Python裡，什麼都是物件



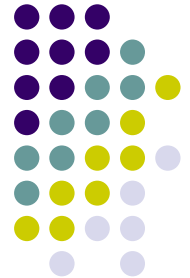
- 其實，本來應使用「物件.屬性項」的語法
- 屬性項（attribute）：就是物件裡頭的名稱（指向物件），而該物件可以是函式（方法）

```
>>> li = [0, 1, 2]; li.__len__()
3
```

- 前後兩個底線的特殊方法（special method），Python會特別對待

```
>>> len(li)
3
```

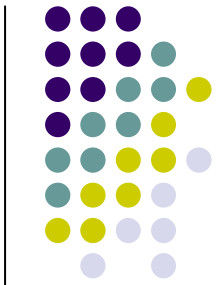
特殊方法：對應到運算子、內建函式



- 若只有「物件.屬性項」，程式會變得複雜難懂；所以提供各種簡便語法，稱爲語法糖（syntax sugar）

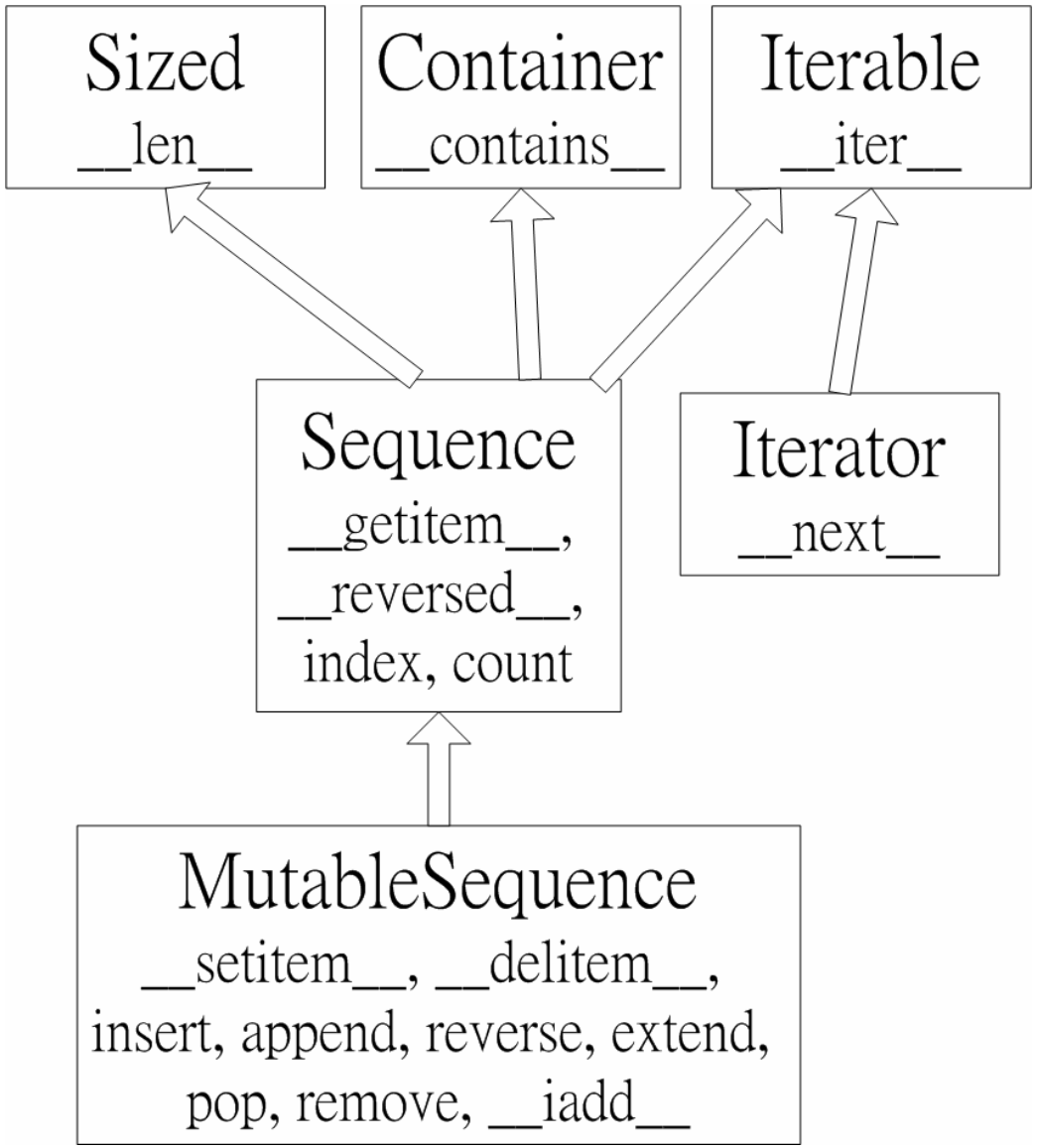
```
>>> a = 3; b = 4; c = 5
>>> a - b, a.__sub__(b)
(-1, -1)
>>> a + b * c, a.__add__(b.__mul__(c))
(23, 23)
>>> li = [0, 1, 2]; li[1] == a
False
>>> li.__getitem__(1).__eq__(a)
False
```

抽象型別

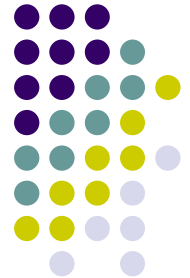


- 把共同的行爲、介面，放在上層的型別
- 如：取出元素、長度（大小、個數）的概念、容器是否含某元素、迭代、等
- 不能從這些型別建立實體物件，所以稱爲抽象
- 如：Sequence（序列）、Mapping（映射）
- Sequence：list、tuple、str
- Mapping：dict

序列抽象型別


Sized
__len__
Container
__contains__
Iterable
__iter__
Sequence
__getitem__,
__reversed__,
index, count
Iterator
__next__
MutableSequence
__setitem__, __delitem__,
insert, append, reverse, extend,
pop, remove, __iadd__

元素存取：索引（index）



- 以方括號「[、]」包住索引值，從0起跳，超過容器的界限則會出錯

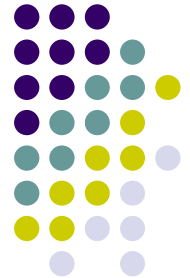
```
>>> li = [3, 4, 5, 6, 7, 8]        # 串列
>>> li[3]
5
>>> s = 'abcdef'; s[len(s) - 1]   # 字串
'f'
>>> t = (1, 3, 5, 7, 9)             # tuple
>>> t[99]
IndexError: tuple index out of range
```

索引值可以是負數

- 假定i是正數，索引值「-i」會被當做「長度-i」，所以s[-i]等同於s[len(s)-i]（最後一個）

```
>>> li = [3, 4, 5, 6, 7, 8]
>>> li[-1], li[len(li) - 1]
(8, 8)
>>> li[0], li[-len(li)]
(3, 3)
>>> li[-2], li[-3]
(7, 6)
```

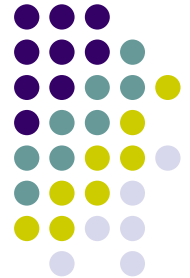
可變的序列型別，使用指派述句修改容器元素



- 正確地說，是讓該索引值的位置指向別的物件

```
>>> li = [3, 4, 5, 6, 7, 8]
>>> li[0] = 30; li[-1] = li[-1] * 10
>>> li
[30, 4, 5, 6, 7, 80]
>>> del li[-1]; li          # del述句，刪除
[30, 4, 5, 6, 7]
>>> del li[2]; li
[30, 4, 6, 7]
```

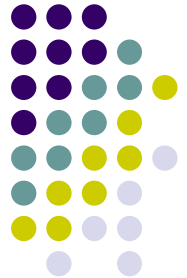
元素存取：切片（slice）



- 基本語法s[i:j]
- i代表起始索引值，j代表結束索引值（不含）

```
>>> li = [0, 1, 2, 3, 4, 5]
>>> li[0:3]                    # 不包含索引值3
[0, 1, 2]
>>> li[0:-1]
[0, 1, 2, 3, 4]                # 注意，沒有5
>>> li[99:3], li[3:99]          # 超過界限，會被當做
([], [3, 4, 5])                 # 長度
>>> li[-3:-1] = 33, 44; li     # 指派
[0, 1, 2, 33, 44, 5]
```

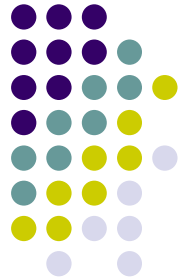
元素存取：切片（slice）



- 省略i，預設爲0。省略j，預設爲長度

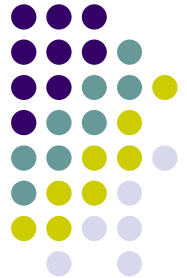
```
>>> li = [0, 1, 2, 3, 4, 5]
>>> li[:2]          # 從頭拿，到某處爲止
[0, 1]
>>> li[2:]          # 從某處開始拿，直到最後
[2, 3, 4, 5]
>>> li[:-3]
[0, 1, 2]
>>> li[:]           # 複製出新的list
[0, 1, 2, 3, 4, 5]
```

s[i:j:k]
k代表每次跳幾個元素



```
>>> li = [0, 1, 2, 3, 4, 5]
>>> li[1:5:2]
[1, 3]
>>> li[-1:1:-2]          # k是負數，從後面往前
[5, 3]
>>> li[::-1]             # 反轉
[5, 4, 3, 2, 1, 0]
>>> a, b, c = li[1:4]; a, b, c
(1, 2, 3)
>>> x, *y = li[1::2]; x, y
(1, [3, 5])
```

問題：



① `li[::2]`、`li[-2::-2]`，什麼意思？

② `del li[:]`，什麼意思？

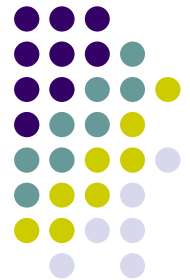
③ 若想刪除最後三個元素，怎麼寫？

④ `del[0:2] = []`，什麼意思？

⑤ `li[1:3] = 'x', 'y', 'z'`，右邊數量比較多，會如何？

⑥ `[::-1]`可反轉順序，請問若明確指定i、j、k的值，辦得到嗎？

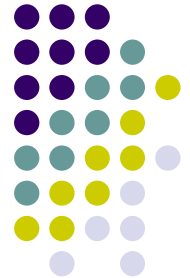
比較運算子：is、is not、==、!=、in、not in



- is：判斷爲同一物件
- ==：比較兩物件內容是否相同
- in，檢查是否「包含」元素，要注意效能速度

```
>>> li = [0, 1, 2, 3, 4]
>>> li2 = li[:]                  # 複製
>>> li == li2, li is li2         # 內容相同，
(True, False)                    # 但非同一物件
>>> 3 in li, [3] in li
(True, False)
>>> t = (0, 1, 2); t2 = t[:]; t3 = tuple(t)
>>> t2 is t, t3 is t             # 注意，因tuple
(True, True)                     # 是不可變物件
```

序列型別和運算子「+」



- 「+」代表連接

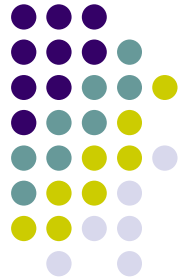
```
>>> li = [0, 1, 2]; s = 'Hi'
>>> li + [3, 4, 5]          # 建立新物件
[0, 1, 2, 3, 4, 5]
>>> s + ', how are you, ' 'John?'   # 注意
'Hi, how are you, John?'
>>> li + list( (3, 4, 5) )        # 轉型
[0, 1, 2, 3, 4, 5]
>>> li + 'abc'
TypeError: can only concatenate list (not
  "str") to list
```

str的「+」，效率低

- 需要連接大量字串時，可使用方法join
- 看起來有點怪，但很多人用；效率高

```
>>> 'a' + 'b' + 'c' + 'd' # 因str不可變，
'abcd'                    # 會產生大量暫時性物件
>>> ''.join(['a', 'b', 'c', 'd'])
'abcd'                  # 空字串，作爲分隔符
>>> t = ('Hi,', 'how', 'are', 'you?')
>>> ' '.join(t)         # 空格，作爲分隔符
'Hi, how are you?'
```

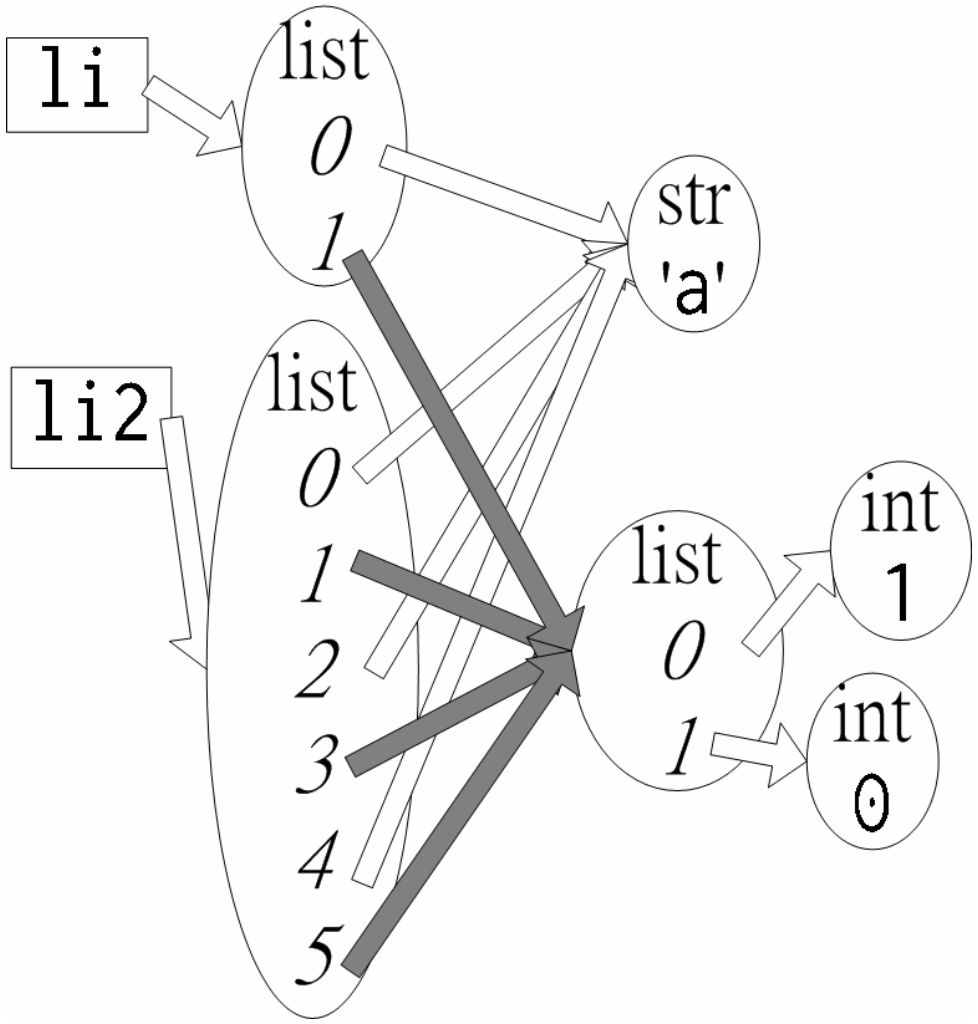
序列型別的運算子「*」



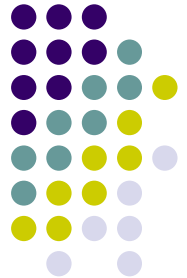
- 「*」是「淺」複製，小心別名現象

```
>>> li = ['a', [0, 1]]
>>> li2 = li * 3; li2
['a', [0, 1], 'a', [0, 1], 'a', [0, 1]]
>>> # 請看下張示意圖
>>> li[1][1] = 9
>>> li2
['a', [0, 9], 'a', [0, 9], 'a', [0, 9]]
```

淺複製示意圖



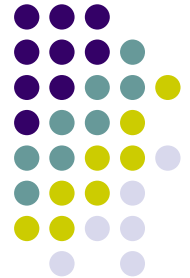
淺複製、深複製



- li2 = li[::]，li2 = list(li)，淺複製
- 模組copy的copy：淺複製
- 模組copy的deepcopy：深複製

```
>> from copy import deepcopy
>>> li = ['a', [0, 1]]
>>> li2 = deepcopy(li)
>>> li[1][1] = 9
>>> li2
['a', [0, 1]]
```

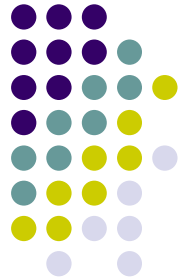
注意：「不可變」的意義



- tuple不可變，僅針對tuple物件本身內容而言
- 不規範tuple元素指向的物件

```
>>> scores = [60, 70, 80]          # list可變
>>> data = ('Amy', 23, scores)
>>> data[1] = 33                   # tuple本身內容不可變
TypeError: 'tuple' object does not support item assignment
>>> data[2][0] = 99                # 但元素（list）可變
>>> data
('Amy', 23, [99, 70, 80])
```

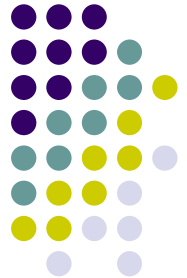
增強型指派運算子「+=」、「*=」



- 不可變：建立新物件
- 可變：原地修改（in-place change）

```
>>> t = t2 = (0, 1, 2)          # tuple
>>> t += (3, 4); t is t2
False
>>> li = li2 = [0, 1, 2]        # list
>>> li += [3, 4]; li is li2
True
>>> li = li2 = [0, 1, 2]        # list
>>> li *= 3; li2
[0, 1, 2, 0, 1, 2, 0, 1, 2]
```

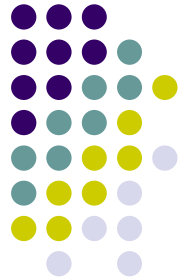
內建函式



- len、min、max、sum、slice

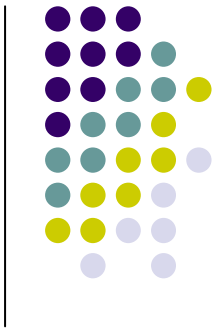
```
>>> li = [3, 9, 8, 2, 5, 6, 1]
>>> min(li), max(li), sum(li)
(1, 9, 34)

>>> s0 = slice(0, 2); s1 = slice(1, -1, 2)
>>> li[s0], li[s1]
([3, 9], [9, 2, 6])
```

方法index、count



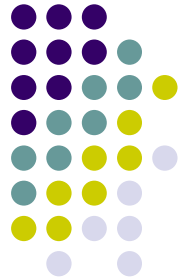
```
>>> li = [3, 'a', 7, 3, 'b', (1, 2), 'c']
>>> li.index(3), li.index(3, 1)  # 求索引值
(0, 3)
>>> li.count(3), li.count(4)     # 個數
(2, 0)
>>> li.index(4)                  # 無此元素，出錯
ValueError: 4 is not in list
>>> 4 in li                      # 使用in來判斷
False
```

方法append、extend、insert、remove、pop



```
>>> li = [0, 1, 2]
>>> li.append(3); li            # 附加元素到末尾
[0, 1, 2, 3]
>>> li.extend( [4, 5] ); li # 加入裡頭的東西
[0, 1, 2, 3, 4, 5]
>>> li.insert(3, 'a'); li      # 插入
[0, 1, 2, 'a', 3, 4, 5]
>>> li.remove('a'); li         # 移除某元素
[0, 1, 2, 3, 4, 5]
>>> li.pop()                   # 拿出最後一個元素
5
>>> li
[0, 1, 2, 3, 4]
```

再次提醒：可變物件

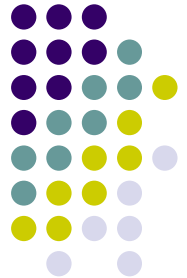


- 注意：建立新物件，原地修改
- 注意：方法的回傳值

```
>>> li = [0, 1, 2]
>>> li = li.append(3)
>>> li
```

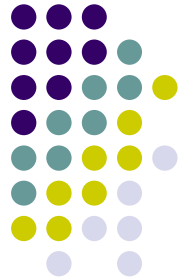
結果是什麼？

內建函式



- range、zip、enumerate、sorted、reversed

```
>>> list(range(1, 10+1, 2))
[1, 3, 5, 7, 9]
>>> list(zip(('a', 'b', 'c'), (30, 41, 52)))
[('a', 30), ('b', 41), ('c', 52)]
>>> list(enumerate(['a', 'b', 'c']))
[(0, 'a'), (1, 'b'), (2, 'c')]
>>> sorted([3, 2, 9, 5])
[2, 3, 5, 9]
>>> list(reversed([3, 2, 9, 5]))
[5, 9, 2, 3]
```



問題

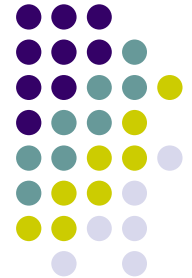
- 你手中有下列資料

```
names = ['Amy', 'Bob', 'Cathy']
scores = [70, 92, 85]
```

- 請寫程式印出如下的樣子

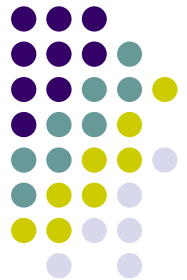
```
0 Amy 70
1 Bob 92
2 Cathy 85
```

迭代、可迭代者、迭代器



- 協定：溝通雙方共同遵守的規則
- 迭代協定：提供方根據協定規則提供一連串的東西，接收方經由協定的介面拿到那些東西
- 可迭代者（Iterable）：能夠提供符合迭代器介面的物件；內建函式iter
- 迭代器（Iterator）：能夠逐一給出一連串東西的介面；內建函式next
- 有些物件既是Iterable、也是Iterator；有些則否

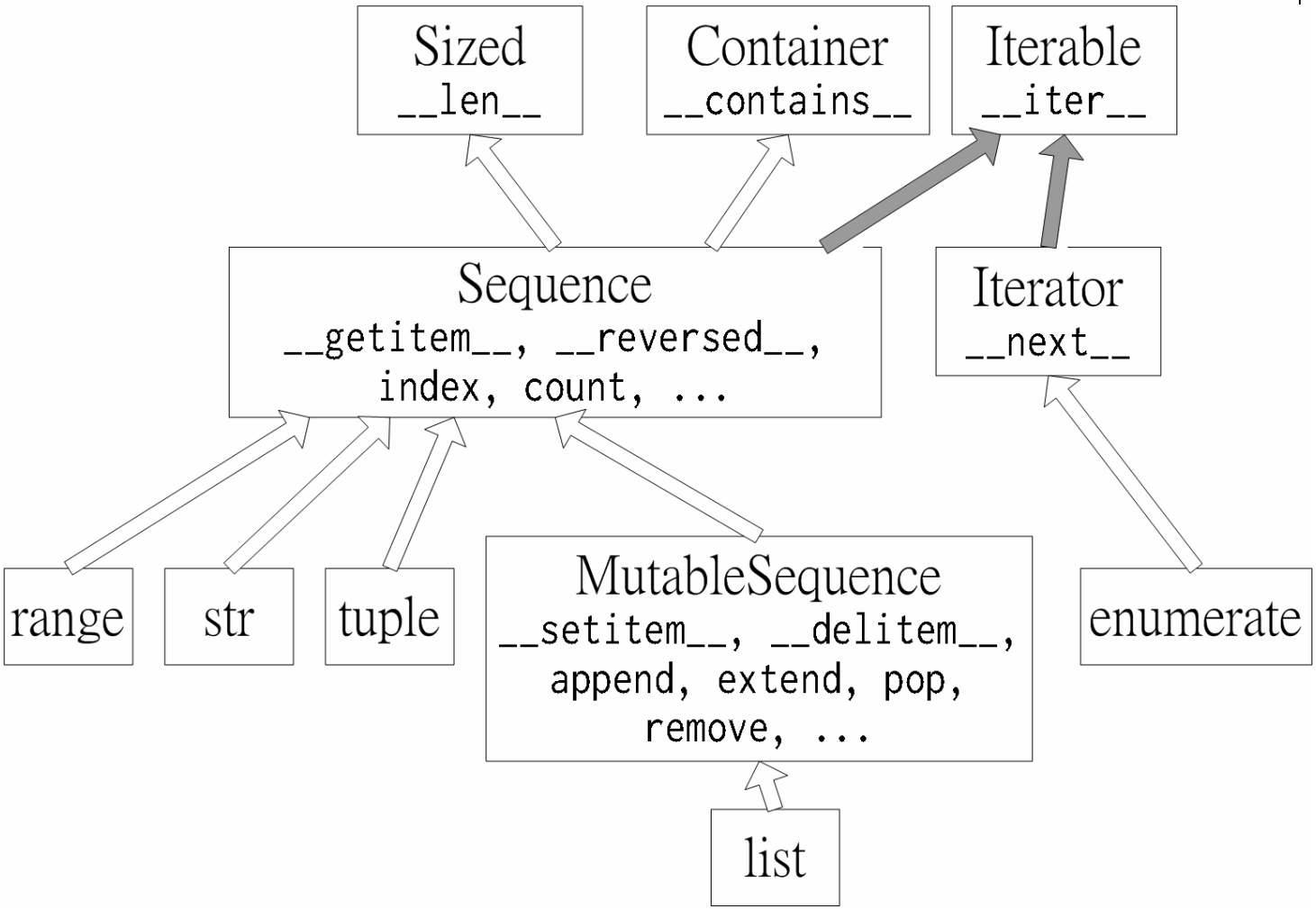
提供方：串列、**tuple**或**range**
接收方：**for**迴圈



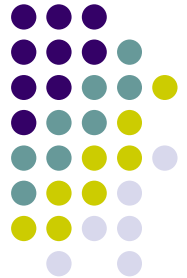
- 以前說：「能提供一串東西的物件」，現在知道是符合「迭代協定」的物件

```
li = [0, 1, 2]          # 串列
for x in li:
    print(x)

for x in (0, 1, 2):     # tuple
    print(x)

for x in range(3):      # range
    print(x)
```

迭代相關抽象型別


Sized
__len__
Container
__contains__
Iterable
__iter__
Sequence
__getitem__, __reversed__,
index, count, ...
Iterator
__next__
range
str
tuple
MutableSequence
__setitem__, __delitem__,
append, extend, pop,
remove, ...
enumerate
list

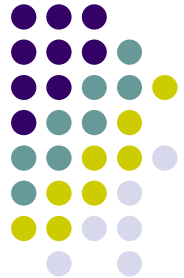
list符合Iterable介面



- 使用內建函式iter，可得到符合Iterator介面的物件
- 使用內建函式next，可從Iterator物件拿出元素

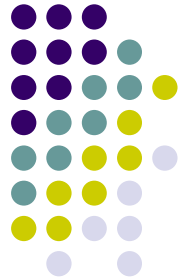
```
>>> li = [0, 1, 2]      # 串列
>>> itb = iter(li)      # 拿到Iterator物件
>>> next(itb)           # 拿出下一個
0
>>> next(itb)           # 拿出下一個
1
>>> next(itb)           # 拿出下一個
2
>>> next(itb)           # 沒了，引發異常
StopIteration
```

不用for述句，自己手動處理



```
li = [0, 1, 2, 3, 4, 5]

itr = iter(li)                  # 取得Iterator
while True:
    try:
        x = next(itr)           # 下一個
        print(x)
    except StopIteration: # 耗盡
        break
```

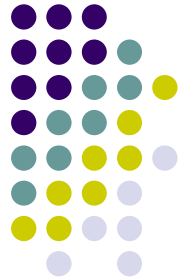
2.x版的差異



- 2.x版以list爲主
- 3.x版改以Iterable和Iterator爲主
- 2.x版的函式與方法，原本使用list，3.x版都改成Iterable或Iterator
- 例如：range

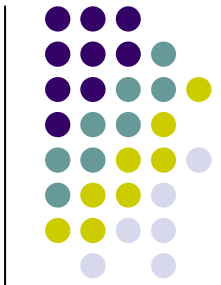
```
>>> range(3)
[0, 1, 2]          # 2.x版，list
>>> range(3)
range(0, 3)        # 3.x版，符合Iterable的物件
```

range、zip、enumerate



```
>>> range(3)              # range符合Sequence
range(0, 3)               # 也符合Iterable介面
>>> list(range(3))
[0, 1, 2]
>>> enumerate(range(3))
<enumerate object at 0x0049C760>
>>> reversed(range(3))
<range_iterator object at 0x00C4F878>
>>> list(zip(reversed(range(3)), range(3)))
[(2, 0), (1, 1), (0, 2)]
```

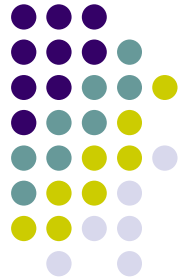
sorted、reversed



- sorted的結果是list
- reversed的參數必須是Sequence

```
>>> sorted(range(9, 1, -2))
[3, 5, 7, 9]
>>> reversed(range(3))
<range_iterator object at 0x00C4F878>
>>> reversed(enumerate(range(3)))
TypeError: argument to reversed() must be a
  sequence
```

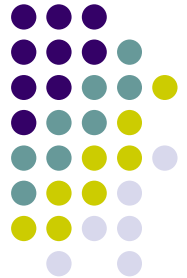
序列指派與迭代協定



- 「等號」右邊可以是Iterable或Iterator

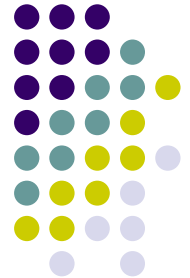
```
>>> n0, n1, n2 = range(3)
>>> n0, n1, n2
(0, 1, 2)
>>> head, *m, rest = range(10)
>>> m
[1, 2, 3, 4, 5, 6, 7, 8]
```

範例：itb_examples.py



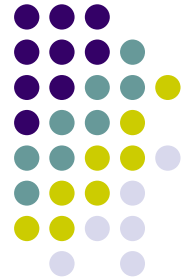
- product函式，傳入含數字的iterable，計算所有元素的乘積
- 使用product函式來計算階乘factorial
- cumulative_sum函式，傳入含數字的iterable，回傳累和串列
- unique函式，拿掉重複的元素
- duplicate函式，找出重複的元素
- group函式，根據參數指定的長度做切割
- flatten，壓平二維陣列

範例：Deck.py（撲克牌）



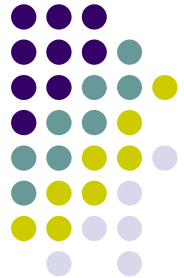
- 建立Deck撲克牌物件後，含52張牌，可行動作是隨機抽牌（draw），抽完後就沒牌了
- 可呼叫len得知還剩多少張牌
- Duck typing（鴨子型別）、繼承抽象型別
- `__len__`、`__iter__`、`__next__`
- Deck物件，既是Iterable、也是Iterator

範例：**MyDeck.py**（撲克牌）



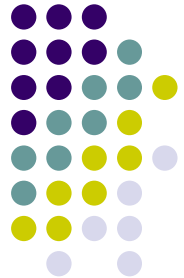
- 建立MyDeck撲克牌物件後，含52張牌，建立時就已亂數排好順序，不可變
- 可使用索引任意存取裡頭的牌
- `__len__`、`__getitem__`、`__iter__`、
- 沒有`__next__`，由另一個類別負責
- MyDeck物件是Iterable，不是Iterator

問題：迭代器只能迭代一次



```
>>> li = [5, 3, 7, 1]
>>> sorted(li) == sorted(li)
True

>>> li_r = reversed(li)
>>> li_r
<list_reverseiterator object at 0x00C6FD10>
>>> list(li_r) == list(li_r)
False
```

問題：複製迭代器



```
>>> r = range(5)
>>> itrs = [iter(r)] * 3      # 淺複製
>>> itrs[0] is itrs[1] is itrs[2]
True                          # 都是同一個

>>> next(itrs[0])
0
>>> next(itrs[1]), next(itrs[2])
(1, 2)
```

Q&A