Python
Container Data Types

馬誠佑

2025/03/21

Collections - List & Array & Dictionary

Array	List	Dictionary
• Order guaranteed • Numerical data types • Must all be the same type	• Order guaranteed • Store anything, any type • Zero-based index	• Order Not Guaranteed • Store anything, any type • Key:value pairs
• from array import array • scores = array(‘d’) • scores.append(97) • scores.append(98) • print(score[1])	• A = [‘John’, 120, ’13.6’]	• D = {‘John’:20, ‘David’:12.6, ‘Tom’:’n/a’} • print(D[‘David’])

List

- a = [1, 2, 34, 5, 67]
- a[index]
- b = [
  [1,’b’,4,d,t,4],
  [2, g, ‘d’ ,e ,’h’, h],
  [d,g,hh,j,3]
  ]
- b[row][col]
- Empty list:
- a = []

Python 表达式	结果	描述
len([1, 2, 3])	3	长度
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	组合
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	重复
3 in [1, 2, 3]	True	元素是否存在于列表中
for x in [1, 2, 3]: print x,	1 2 3	迭代

Python列表截取

Python 的列表截取实例如下：

```
>>>L = ['Google', 'Runoob', 'Taobao']
>>> L[2]
'Taobao'
>>> L[-2]
'Runoob'
>>> L[1:]
['Runoob', 'Taobao']
>>>
```

描述：

Python 表达式	结果	描述
L[2]	'Taobao'	读取列表中第三个元素
L[-2]	'Runoob'	读取列表中倒数第二个元素
L[1:]	['Runoob', 'Taobao']	从第二个元素开始截取列表

List function

```
a = ['a','b',' c','d','e']
a.append('f')
print(a)
```
✓ 0.0s
```
['a', 'b', ' c', 'd', 'e', 'f']
```

```
a = ['a','b','c','d','e']
a.remove('c')
print(a)
```
✓ 0.0s
```
['a', 'b', 'd', 'e']
```

```
b = [1,2,3,4,5]
a.extend(b)
print(a)
```
✓ 0.0s
```
['a', 'b', ' c', 'd', 'e', 'f', 1, 2, 3, 4, 5]
```

```
a = ['a','b','c','d','e']
a.reverse()
print(a)
a = a[::-1]
print(a)
```
✓ 0.0s
```
['e', 'd', 'c', 'b', 'a']
['a', 'b', 'c', 'd', 'e']
```

```
a = ['a','b','c','d','e']
print(a.index('c'))
```
✓ 0.0s
```
2
```

```
a = ['c','a','e','d','b']
a.sort()
print(a)
```
✓ 0.0s
```
['a', 'b', 'c', 'd', 'e']
```

```
a.insert(0,'x')
print(a)
```
✓ 0.0s
```
['x', 'a', 'b', 'c', 'd', 'e']
```

```
a = ['a','b','c','d','e']
tmp = a.pop()
print(a)
print(tmp)
tmp = a.pop(2)
print(a)
print(tmp)
```
✓ 0.0s
```
['a', 'b', 'c', 'd']
e
['a', 'b', 'd']
c
```

Python包含以下方法:

序号	方法
1	list.append(obj) 在列表末尾添加新的对象
2	list.count(obj) 统计某个元素在列表中出现的次数
3	list.extend(seq) 在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
4	list.index(obj) 从列表中找出某个值第一个匹配项的索引位置
5	list.insert(index, obj) 将对象插入列表
6	list.pop([index=-1]) 移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
7	list.remove(obj) 移除列表中某个值的第一个匹配项
8	list.reverse() 反向列表中元素
9	list.sort(cmp=None, key=None, reverse=False) 对原列表进行排序

Tuple

```
tup1 = ('physics','maths','chemistry',1997,2000)
tup3 = 'a','b','c'
print(tup1)
print(tup3[1])
```

✓ 0.0s

```
('physics', 'maths', 'chemistry', 1997, 2000)
b
```

Dictionary

- {key1:value1, key2:value2, key3:value3, ...}
- v = d[‘key’]
- empty dictionary:

  d = {}

Dictionary function

```
D = {'a':1,'b':2,'c':3}
D2 = D
DC = D.copy()
D['a'] = 10
print(D2)
print(DC)
D = {'a':1,'b':2,'c':3,'d':{'e':5,'f':6}}
D['d']['e']=6
DC = D.copy() #淺拷貝
print(DC)

import copy

D = {'a':1,'b':2,'c':3,'d':{'e':5,'f':6}}
DC = copy.deepcopy(D) #深拷貝
D['d']['e'] = 6
print(DC)
```

✓ 0.0s

```
{'a': 10, 'b': 2, 'c': 3}
{'a': 1, 'b': 2, 'c': 3}
{'a': 1, 'b': 2, 'c': 3, 'd': {'e': 6, 'f': 6}}
{'a': 1, 'b': 2, 'c': 3, 'd': {'e': 5, 'f': 6}}
```

```
ks = D.keys()
print(ks)
print(type(ks))
```

✓ 0.0s

```
dict_keys(['a', 'b', 'c', 'd'])
<class 'dict_keys'>
```

```
D = {'a':1,'b':2,'c':3,'d':{'e':5,'f':6}}
ks = D.keys()
print(ks)
print(type(ks))
print(D.values())
v = D.pop('a')
print(D)
print(v)
pair = D.popitem()
print(D)
print(pair)
D2 = {'a':10,'b':4,'d':{'e':7,'f':6}}
D.update(D2)
print(D)
```

✓ 0.0s

```
dict_keys(['a', 'b', 'c', 'd'])
<class 'dict_keys'>
dict_values([1, 2, 3, {'e': 5, 'f': 6}])
{'b': 2, 'c': 3, 'd': {'e': 5, 'f': 6}}
1
{'b': 2, 'c': 3}
('d', {'e': 5, 'f': 6})
{'b': 4, 'c': 3, 'a': 10, 'd': {'e': 7, 'f': 6}}
```

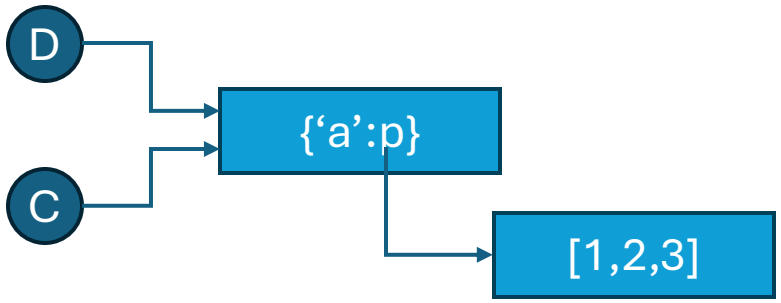
Python字典包含了以下内置方法：

序号	函数及描述
1	dict.clear() 删除字典内所有元素
2	dict.copy() 返回一个字典的浅复制
3	dict.fromkeys(seq[, val]) 创建一个新字典，以序列 seq 中元素做字典的键，val 为字典所有键对应的初始值
4	dict.get(key, default=None) 返回指定键的值，如果值不在字典中返回default值
5	dict.has_key(key) 如果键在字典dict里返回true，否则返回false
6	dict.items() 以列表返回可遍历的(键, 值) 元组数组
7	dict.keys() 以列表返回一个字典所有的键
8	dict.setdefault(key, default=None) 和get()类似, 但如果键不存在于字典中，将会添加键并将值设为default
9	dict.update(dict2) 把字典dict2的键/值对更新到dict里
10	dict.values() 以列表返回字典中的所有值
11	pop(key[,default]) 删除字典给定键 key 所对应的值，返回值为被删除的值。key值必须给出。 否则，返回default值。
12	popitem() 随机返回并删除字典中的一对键和值。

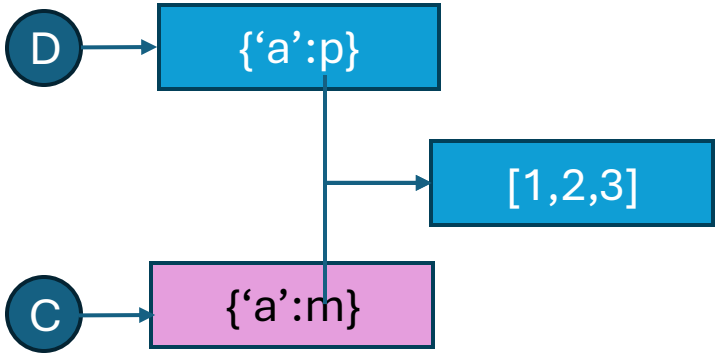
Shallow vs. Deep Copy

D = {‘a’:[1,2,3]}

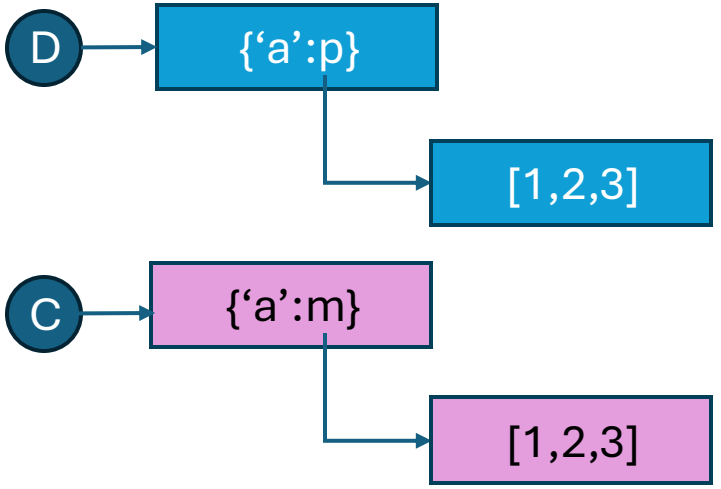
C = D



C = D.copy()



C = copy.deepcopy(D)



Set

```
set1 = {1, 3, 4, 5, 2, 6}
set2 = {2, 4, 5, 7}
print(set1)
print(set1 - set2) # same as set1.difference(set2)
print(set1^set2) # same as set1.symmetric_difference(set2)
print(set2.issubset(set1))
set2 = {3,4,5}
print(set2.issubset(set1))
print(set1.issuperset(set2))
```

✓ 0.0s

```
{1, 2, 3, 4, 5, 6}
{1, 3, 6}
{1, 3, 6, 7}
False
True
True
```

```
set1 = {1, 2, 3, 4, 5, 6}
set2 = {2, 4, 5, 7}

print(set1 - set2) # same as set1.difference(set2)
print(set1^set2) # same as set1.symmetric_difference(set2)
```

✓ 0.0s

```
{1, 3, 6}
{1, 3, 6, 7}
```

{}	create a new set
set()	
.add()	add a single element
.update()	add multiple elements
.remove()	remove an element; error if not in set
.discard()	remove an element; no error if not in set
.pop()	remove any element from set
.clear()	clear set
in	check if element in set
len(set)	number of elements in set
max(set)	largest element in set
min(set)	smallest element in set
sorted(set)	sorted list of elements in set
sum(set)	sum of all elements in set

union	|	.union()
intersection	&	.intersection()
difference	-	.difference()
symmetric difference	^	.symmetric_difference()

Comparison of set ==, >, <, <=, >=

ChainMap

- from collections import ChainMap
- Combine multiple dictionary

```
from collections import ChainMap
from collections import OrderedDict, defaultdict

numbers = {"one": 1, "two": 2}
letters = {"a": "A", "b": "B"}
CM = ChainMap(numbers, letters)
print(CM)
CM = ChainMap(numbers, {"a": "A", "b": "B"})
print(CM)
numbers = OrderedDict(one=1, two=2)
letters = defaultdict(str, {"a": "A", "b": "B"})
CM = ChainMap(numbers, letters)
CM = ChainMap.fromkeys(["one", "two","three"], 0)
```

✓ 0.0s

```
ChainMap({'one': 1, 'two': 2}, {'a': 'A', 'b': 'B'})
ChainMap({'one': 1, 'two': 2}, {'a': 'A', 'b': 'B'})
```

```
numbers = {"one": 1, "two": 2}
letters = {"a": "A", "b": "B"}
CM = ChainMap(numbers, letters)
print(CM['a'])\
```

✓ 0.0s

```
A
```

ChainMap2

```
for_adoption = {"dogs": 10, "cats": 7, "pythons": 3}
vet_treatment = {"dogs": 4, "cats": 3, "turtles": 1}
pets = ChainMap(for_adoption, vet_treatment)
print(pets['dogs'])
print(pets['cats'])
print(pets['turtles'])
print(pets['pythons'])
```

✓ 0.0s

```
10
7
1
3
```

```
for key, value in pets.items():
    print(key, "->", value)

for key in pets: # for key in pets.keys():
    print(key, "->", pets[key])

for value in pets.values():
    print(value)
```

- Key 有overlap時會return先找到的（先放入ChainMap的）
- 存取方式同dictionary

ChainMap3

- pop, del, clear 皆僅對第一個dictionary做操作

```
numbers = {"one": 1, "two": 2}
letters = {"a": "A", "b": "B"}
alpha_num = ChainMap(numbers, letters)
print(alpha_num)

# Add a new key-value pair
alpha_num["c"] = "C"
print(alpha_num)

# Update an existing key
alpha_num["b"] = "b"
print(alpha_num)
# Pop keys
p = alpha_num.pop("two")
print(p)
print(alpha_num)
p = alpha_num.pop("a")
```

0.1s

```
ChainMap({'one': 1, 'two': 2}, {'a': 'A', 'b': 'B'})
ChainMap({'one': 1, 'two': 2, 'c': 'C'}, {'a': 'A', 'b': 'B'})
ChainMap({'one': 1, 'two': 2, 'c': 'C', 'b': 'b'}, {'a': 'A', 'b': 'B'})
2
ChainMap({'one': 1, 'c': 'C', 'b': 'b'}, {'a': 'A', 'b': 'B'})

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call la
File ~/anaconda3/lib/python3.11/collections/__init__.py:1074, in ChainMa
   1073 try:
-> 1074     return self.maps[0].pop(key, *args)
   1075 except KeyError:

KeyError: 'a'
```

ChainMap4

- 可利用此規則來創建不修改原始字典的可更新字典
- 在此使用情境下可使用空字典作為ChainMap的第一個參數

```
numbers = {"one": 1, "two": 2}
letters = {"a": "A", "b": "B"}
alpha_num = ChainMap({}, numbers, letters)
print(alpha_num)
alpha_num["comma"] = ","
alpha_num["period"] = "."
alpha_num["a"] = "a"
print(alpha_num)
```

✓ 0.0s

```
ChainMap({}, {'one': 1, 'two': 2}, {'a': 'A', 'b': 'B'})
ChainMap({'comma': ',', 'period': '.', 'a': 'a'}, {'one': 1, 'two': 2}, {'a': 'A', 'b': 'B'})
```

ChainMap5

```
for_adoption = {"dogs": 10, "cats": 7, "pythons": 3}
vet_treatment = {"dogs": 4, "cats": 3, "turtles": 1}
pets = ChainMap(for_adoption, vet_treatment)
print(pets)
pets.maps.append({'hamsters':2})
print(pets)
print(pets.maps)
for mapping in pets.maps:
    print(mapping)
pets.maps.reverse()
print(pets)
```

✓ 0.0s

```
ChainMap({'dogs': 10, 'cats': 7, 'pythons': 3}, {'dogs': 4, 'cats': 3, 'turtles': 1})
ChainMap({'dogs': 10, 'cats': 7, 'pythons': 3}, {'dogs': 4, 'cats': 3, 'turtles': 1}, {'hamsters': 2})
[{'dogs': 10, 'cats': 7, 'pythons': 3}, {'dogs': 4, 'cats': 3, 'turtles': 1}, {'hamsters': 2}]
{'dogs': 10, 'cats': 7, 'pythons': 3}
{'dogs': 4, 'cats': 3, 'turtles': 1}
{'hamsters': 2}
ChainMap({'hamsters': 2}, {'dogs': 4, 'cats': 3, 'turtles': 1}, {'dogs': 10, 'cats': 7, 'pythons': 3})
```

ChainMap6

```
mom = {"name": "Jane", "age": 31}
dad = {"name": "John", "age": 35}
family = ChainMap(mom, dad)
print(family)
son = {"name": "Tim", "age": 5}
family = family.new_child(son)
for person in family.maps:
    print(person)

print(family.parents)
print(family)
```

✓ 0.0s

```
ChainMap({'name': 'Jane', 'age': 31}, {'name': 'John', 'age': 35})
{'name': 'Tim', 'age': 5}
{'name': 'Jane', 'age': 31}
{'name': 'John', 'age': 35}
ChainMap({'name': 'Jane', 'age': 31}, {'name': 'John', 'age': 35})
ChainMap({'name': 'Tim', 'age': 5}, {'name': 'Jane', 'age': 31}, {'name': 'John', 'age': 35})
```

Python
Flow Control: Conditional Statements and Loops

馬誠佑

2025/03/21

if, elif, and else

- if 判斷式：
- if 判斷式：

  elif 判斷式：

  elif 判斷式：

  ...

  else:

判斷式

- Logical operator: ==, >, <, >=, <=, and, or, not, !=
- in
- True, False, None, [], ‘’, (), {}, set()

if … else … in one line

- beta = 99 if alpha >=100 else 80 if alpha == 7 else 70

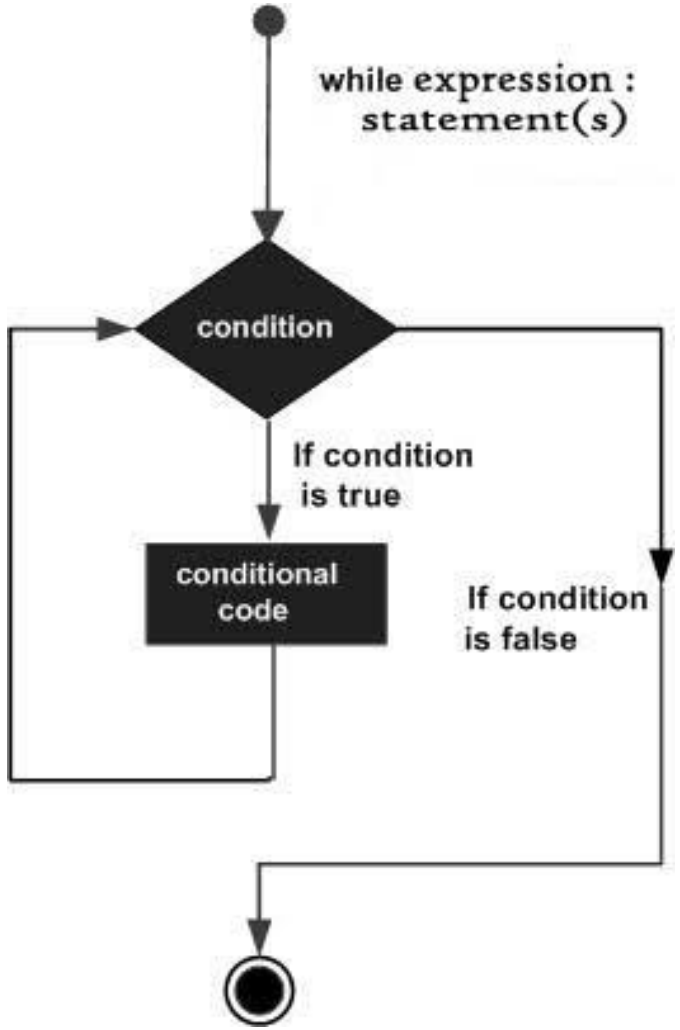
Equals to:

```
if alpha >= 100:
        beta = 99
else:
        if alpha == 7:
                beta = 80
        else:
                beta = 70
```

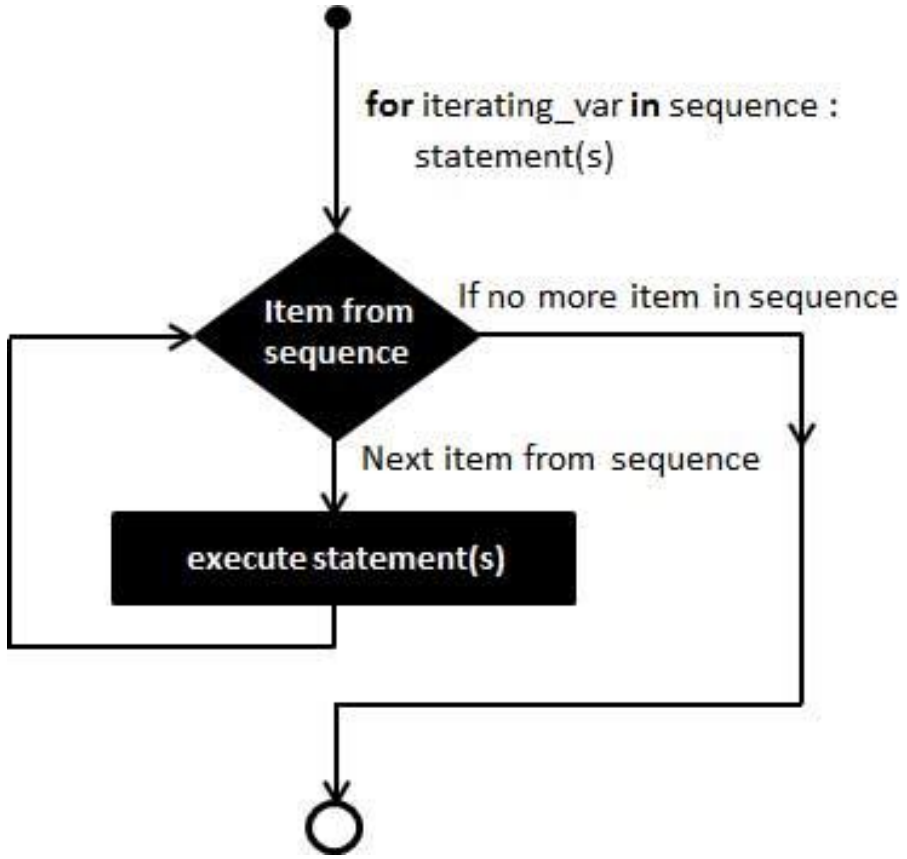
Loop

- while 條件判斷 :
- for i in range(0,10): .....
- A= [‘a’, ’b’, ‘c’, ‘d’, ‘e’]
- for a in A: ...

```python
numbers = [12, 37, 5, 42, 8, 3]
even = []
odd = []
while len(numbers) > 0 :
    number = numbers.pop()
    if(number % 2 == 0):
        even.append(number)
    else:
        odd.append(number)
```

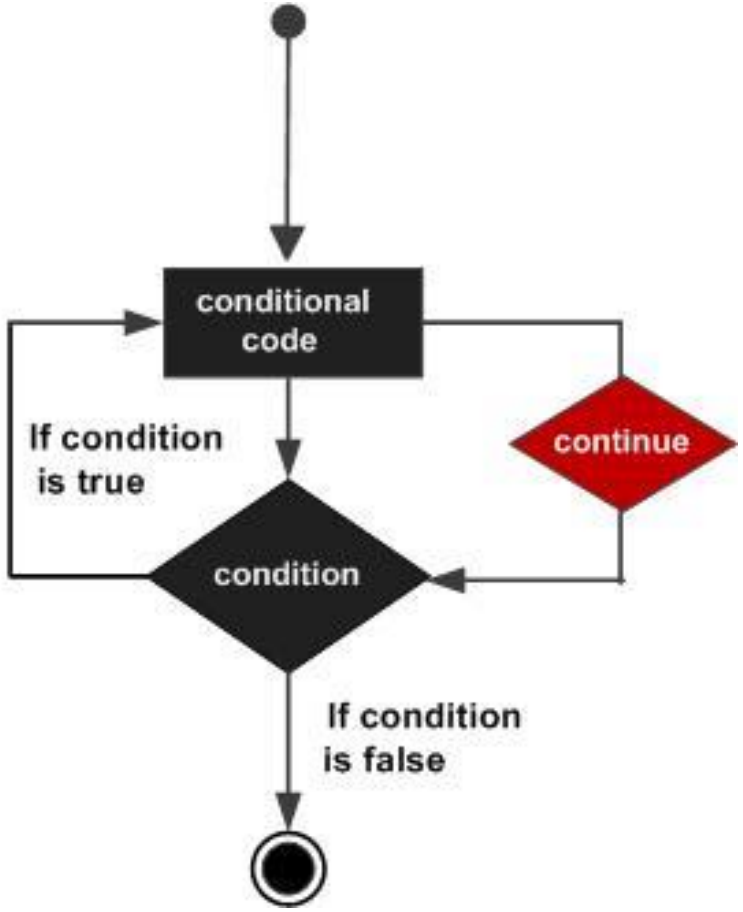


```
for letter in ‘Python’:     # 1st example
    print (‘Current letter: ‘ + letter)

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # 2nd example
    print (‘Current fruit: ‘ + fruit)

print "Good bye!"
```



```
for num in range(10,20):
    for i in range(2,num):
        if num%i == 0:
            j = num/i
            print (‘%d = %d * %d’ % (num, i, j) )
            break
        else:
            print (str(num) + ‘is a prime’)
```

```
for letter in 'Python':     # 1st example
  if letter == 'h':
    continue
  print (‘Current letter: ‘ + letter)

var = 10                    # 2nd example
while var > 0:
  var = var -1
  if var == 5:
    continue
  print (‘Current value: ‘ + str(var))
print (“Good bye!”)
```



break, continue

- break：跳出上”一層”的loop
- continue：直接進入下一輪迴圈

用else檢查break

```
numbers = [1,3,5]
position = 0
while position < len(numbers):
    number = numbers[position]
    if number % 2 == 0:
        print("Found even number", number)
        break
    position += 1
else:
    print("No even number found")
```

```
word = 'thud'
for letter in word:
    if letter == 'x':
        print("There is an 'x'!")
        break
    print(letter)
else:
    print("There is no 'x'!")
```

enumerate

- a = [‘ha’, ’hi’, ‘hh’, ‘kk’, ‘dd’]

- for i, v in enumerate(a):

  print(‘index: ’ + i + ’ v: ’ + v)

range(), zip(), zip(*)

- Range(start, stop, step)
- zip(iterable, ...)
- zip(*iterable)

```
a = [1,2,3,4,5]
b = ['a','b','c','d','e']
c = [(1,'a'),(2,'b'),(3,'c'),(4,'d'),(5,'e')]
zipped = zip(a,b)
for i in zipped:
    print(i)
c1, c2 = zip(*c)
print(c1)
print(c2)
zipped = zip(a,b)
c3, c4 = zip(*zipped)
print(c3)
print(c4)
```

✓ 0.0s

```
(1, 'a')
(2, 'b')
(3, 'c')
(4, 'd')
(5, 'e')
(1, 2, 3, 4, 5)
('a', 'b', 'c', 'd', 'e')
(1, 2, 3, 4, 5)
('a', 'b', 'c', 'd', 'e')
```

Homework3 -container data types 1/3

1.

(列印不同的數值) 請撰寫一程式，讀取一行中以空白間隔的數值，並顯示不同的數值(若一數值出現多次時，則只顯示出一次)。(提示：讀取所有的數字，並將它儲存於 list1 串列中，建立一新的串列 list2，將串列 list1 的數字加到 list2，若數字已存存於串列中，則加以忽略之)。以下為程式範例的執行結果：

2.

**10.3 (計算數值的出現) 請撰寫一程式，讀取一些介於 1 到 100 的整數，並計算每個數值出現的個數。以下是程式執行的樣本。

```
Enter integers between 1 and 100: 2 5 6 5 4 3 23 43 2 2 5 6 5 4 3 23 43 2
2 occurs 2 times
3 occurs 1 time
4 occurs 1 time
5 occurs 2 times
6 occurs 1 time
23 occurs 1 time
43 occurs 1 time
```

注意，若數字出現超過一次，則以複數 times 輸出。

Homework3 -container data types 2/3

3.

***10.1** (給予等級) 請撰寫一程式，讀取學生分數串列，接著根據以下規則給予成績等級：

The grade is A if score is >= best – 10.
The grade is B if score is >= best – 20.
The grade is C if score is >= best – 30.
The grade is D if score is >= best – 40.
The grade is F otherwise.

以下為範例執行結果：

```
Enter scores: 40 55 70 58 c
Student 0 score is 40 and grade is C
Student 1 score is 55 and grade is B
Student 2 score is 70 and grade is A
Student 3 score is 58 and grade is B
```

Homework3 -container data types 3/3

4. Please process the long string in the file, ‘3_4_input.txt’ and calculate:

(1)Total # of words:

(2)Total # of word’s type:

(3)The first three words that most frequently appear in the article and their #:

(4)List the first three words that only appear once in the article in lexical order.

(5)Count the # of each character’s (a-z, 0-9) appearance in the article and print it out.

Ex: a->100

b->50

c->88…

5. In a grocery store, the price lists are as below:

fruits_prices = {"apple": 0.80, "grape": 0.60, "orange": 0.40}

veggies_prices = {"tomato": 1.80, "pepper": 1.40, "onion": 1.23}

The order list is : order = {”grape": 4, "tomato": 10, "orange": 4, “pepper”:1}.

Please write a program with only one “for loop” to calculate the total price of each item and the total amount of the bill for the order list.

Ex:

```
apple : $0.80 × 4 = $3.20
tomato: $1.20 × 8 = $9.60
orange: $0.50 × 4 = $2.00
Total: $14.80
```

Homework4 - Flow Control: Conditional Statements and Loops 1/3

1. 可使用巢狀 if 敘述，撰寫計算稅款的程式。

根據報稅身分及可徵稅的所得，計算美國聯邦個人所得稅。報稅身分有四種：單身納稅人、已婚共同報稅或符合資格的鰥寡納稅人，以及已婚分開報稅，還有戶長納稅人。每年的稅率都不太一樣。表 4.2 為 2009 年的稅率。假設您是單身，且可徵稅之所得為 $10,000，裡頭的 $8,350 會被徵收 10% 的稅，其餘所得 $1,650 則會被徵收 15% 的稅，因此，須繳納的總稅金為 $1,082.50。

表 4.2 2009 年美國聯邦個人稅率

邊際稅率 (Marginal Tax Rate)	單身	已婚共同報稅或符合資格的鰥寡納稅人	已婚分開報稅	戶長
10%	$0–$8,350	$0–$16,700	$0–$8,350	$0–$11,950
15%	$8,351–$33,950	$16,701–$67,900	$8,351–$33,950	$11,951–$45,500
25%	$33,951–$82,250	$67,901–$137,050	$33,951–$68,525	$45,501–$117,450
28%	$82,251–$171,550	$137,051–$208,850	$68,526–$104,425	$117,451–$190,200
33%	$171,551–$372,950	$208,851–$372,950	$104,426–$186,475	$190,201–$372,950
35%	$372,951+	$372,951+	$186,476+	$372,951+

我們將撰寫一個計算個人所得稅的程式。此程式需提示使用者輸入報稅身分及可被徵稅之所得，接著計算應繳稅款。輸入 0 代表單身納稅人，1 代表已婚共同報稅或符合資格的鰥寡納稅人，2 代表已婚分開報稅，3 則代表戶長。

此程式會根據報稅身分，為可徵稅之所得計算應繳稅款。可使用如下的 if 敘述判斷報稅身分：

```
if status == 0:
    # Compute tax for single filers
elif status == 1:
    # Compute tax for married filing jointly
elif status == 2:
    # Compute tax for married filing separately
elif status == 3:
    # Compute tax for head of household
else:
    # Display wrong status
```

對於每一種報稅身分，都有六種不同的稅率。每種稅率會對可徵稅之收入的特定部分做計算。比方說，假設單身報稅人的可徵稅之收入為 $400,000，其中的$8,350 會被徵收 10% 的稅，(33,950 – 8,350)會被徵收 15% 的稅，(82,250 – 33,950)會被徵收 25% 的稅，(171,550 – 82,250)會被徵收 28% 的稅，(372,950 – 171,550)會被徵收 33% 的稅，而 (400,000 – 372,950)會被徵收 35% 的稅。

Homework4 - Flow Control: Conditional Statements and Loops 2/3

2.

這個樂透程式產生隨機數值，比較數位，並使用毒株運算子。

假設我們想開發一個樂透程式。此程式會隨機產生一個二位數的樂透數字，並提示使用者輸入一個二位數數值，接著會根據以下規則，判斷使用者是否贏得樂透彩：

1. 使用者所輸入的數字，如果與樂透號碼完全符合，獎金為$10,000。
2. 使用者所輸入的數字，如果與樂透號碼出現的數字相符，但順序不同，獎金為$3,000。
3. 使用者所輸入的數字，如果只有一個數字與樂透號碼的數字相符，獎金為 $1,000。

3.

這裡所要解決的問題為猜測電腦所選的數字。我們將撰寫一程式，隨機產生 0 到 100 之間(包含 100)的整數。程式接下來會持續提示使用者輸入一個數字，直到該數字與電腦隨機產生的數字相符。對於每一次的使用者輸入，程式都會告訴使用者所猜測的數字過小或過大，讓使用者更有概念地做進一步猜測。以下為範例執行結果：

Homework4 - Flow Control: Conditional Statements and Loops 2/3

4. 整數 4 與 2 的最大公因數(greatest common divisor, GCD)為 2。整數 16 與 24 的最大公因數則為 8。要如何找尋最大公因數呢？假設兩個輸入的整數分別為 n1 與 n2。我們知道數字 1 為一個公因數，但不見得會是最大公因數。我們可檢查 k (k = 2、3、4，依此類推) 是否為 n1 與 n2 的公因數，直到 k 大於 n1 或 n2。將公因數儲存於名為 gcd 的變數。gcd 的起始值為 1。當找到新的公因數，其也將變成新的 gcd。檢查從 2 到 n1 或 n2 所有可能的公因數後，這時候變數 gcd 的值即是最大公因數。這個概念可轉換成以下迴圈內容：

5. ***5.19 (顯示金字塔) 請撰寫一程式，提示使用者輸入 1 到 15 間的整數，並以金字塔的形式顯示，如以下範例執行的結果所示：**

```
Enter the number of lines: 7 Enter
          1
        2 1 2
      3 2 1 2 3
    4 3 2 1 2 3 4
  5 4 3 2 1 2 3 4 5
6 5 4 3 2 1 2 3 4 5 6
7 6 5 4 3 2 1 2 3 4 5 6 7
```