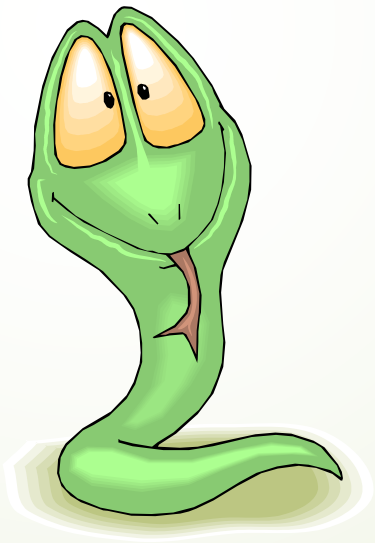
Python程式設計入門
-more data type



基本型態 – 串列(Lists)

- 任意物件的串列

```
a = [2, 3, 4]                # A list of integer
b = [2, 7, 3.5, "Hello"]     # A mixed list
c = []                       # An empty list
d = [2, [a, b]]              # A list containing a list
e = a + b                    # Join two lists
Print €
```

- 串列的操作

```
X = a[1]          # Get 2nd element (0 is first)
y = b[1:3]        # Return a sub-list
z = d[1][0][2]    # Nested lists
b[0] = 42         # Change an element
```

example

```
a = [2, 3, 4]                # A list of integer
b = [2, 7, 3.5, "Hello"]     # A mixed list
c = []                       # An empty list
d = [2, [a, b]]              # A list containing a list
print (d)
z = d[1][0][2]
print(z)
```

Negative Indexing

- Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

Ex.

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
print(len(thelist))
```

Range of Indexes

- Ex1:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

- Ex2:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

- Ex3:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:2])
```

Change/add Item Value

- Ex.

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

- Ex.

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

- Ex.

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Remove Item

- Ex.

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

- Ex.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

- Ex.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

Remove Item (2)

- Ex.

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

- Ex.

```
thislist = ["apple", "banana", "cherry"]
del thislist[1:]
print(thislist)
```

- Ex.

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Copy a List

- Ex..

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

- Ex..

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

-

- Ex..

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
mylist.pop(1)
print(mylist)
print(thislist)
```

- Ex..

```
thislist = ["apple", "banana", "cherry"]
mylist =  thislist
mylist.pop(1)
print(mylist)
print(thislist)
```

Set

- A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

- Ex

```
set1 = {5, 2, 1, "c", 1}
print (set1)
set2 = {1,2,3}
set3 = set1.union(set2)
print (set3)
```

基本型態 – 固定有序列(Tuples)

- Tuples

```
f = (2,3,4,5)                    # A tuple of integers
g = (,)                          # An empty tuple
h = (2, [3,4], (10,11,12))       # A tuple containing mixed objects
```

- Tuples的操作

```
x = f[1]        # Element access. x = 3
y = f[1:3]      # Slices. y = (3,4)
z = h[1][1]     # Nesting. z = 4
a = f[-1]       # a=5
```

- 特色
  - 與list類似，最大的不同tuple是一種唯讀且不可變更的資料結構
  - 不可取代tuple中的任意一個元素，因為它是唯讀不可變更的

基本型態 – 字典 (Dictionaries)

- Dictionaries (關聯陣列)

```
a = { }                    # An empty dictionary
b = { 'x': 3, 'y': 4 }     key : value
c = { 'uid': 105,
      'login': 'beazley',
      'name' : 'David Beazley'
    }
```

- Dictionaries的存取　　每一筆都是一item
-

```
phonebook = {
        "John" : 938477566,
        "Jack" : 938377264,
        "Jill" : 947662781
    }
    phonebook.pop("John")
    print(phonebook)
    for key, val in phonebook.items():
        print (key, "->",val)
    print (phonebook["Jill"])
    for value in phonebook.values():
        print(value)
```

檔案處理

- open()函式

```
f = open("foo","w")        # Open a file for writing
g = open("bar","r") # Open a file for reading
```

- 檔案的讀取/寫入

```
f.write("Hello World")
data = g.read()            # Read all data
line = g.readline()    # Read a single line
lines = g.readlines()      # Read data as a list of lines
```

- 格式化的輸入輸出
  - 使用%來格式化字串

```
for i in range(0,10):
    f.write("2 times %d = %d\n" % (i, 2*i))
```

First example

```
# Open a file
fo = open("foo.txt", "w")
fo.write( "Python is a great language.\nYeah its great!!\n")

# Close opend file
fo.close()
-------------------------------------
# r+ Opens a file for both reading and writing.
fo = open("foo.txt", "r+")
str = fo.read(10)
print ("Read String is : ", str)

# Close opened file
fo.close()
```

example

```
# import systems module
#import sys
marker = '::::::'
#for name in sys.argv[1:]:
input = open("1.py", 'r')
print (marker + "1.py")
print (input.read())
```

讀整個資料夾內檔案

```
import os
def read_path(path_name):
    for dir_item in os.listdir(path_name):
        #絕對路徑表示， 可識別的操作路徑
        full_path = os.path.abspath(os.path.join(path_name, dir_item))
        if os.path.isdir(full_path):   #如果是資料夾，繼續遞迴呼叫
            read_path(full_path)
        else:                          #檔案
            if full_path.endswith('.txt'):
                input = open(full_path, 'r')
                print (input.read())

if __name__ == '__main__':
        read_path("textdata")       #輸入資料夾路徑
```

Class in pythom

```
class ClassName:
   'Optional class documentation string'
   class_suite
```

The class has a documentation string, which can be accessed via ClassName.__doc__.

The class_suite consists of all the component statements defining class members, data attributes and functions.

ex

```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print ("Total Employee %d" % Employee.empCount)

   def displayEmployee(self):
      print ("Name : ", self.name,  ", Salary: ", self.salary)
```

```
#This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
#This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

Class Inheritance

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
    'Optional class documentation string'
    class_suite
```

ex

```
class Parent:        # define parent class
   parentAttr = 100
   def __init__(self):
      print ("Calling parent constructor")

   def parentMethod(self):
      print ('Calling parent method')

   def setAttr(self, attr):
      Parent.parentAttr = attr

   def getAttr(self):
      print ("Parent attribute :", Parent.parentAttr)
```

```
class Child(Parent): # define child class
   def __init__(self):
      print ("Calling child constructor")

   def childMethod(self):
      print ('Calling child method')

c = Child()          # instance of child
c.childMethod()      # child calls its method
c.parentMethod()     # calls parent's method
c.setAttr(200)       # again call parent's method
c.getAttr()          # again call parent's method
```

Overloading Operators

```
class Vector:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return 'Vector (%d, %d)' % (self.a, self.b)

    def __add__(self,other):
        return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print (v1 + v2)
```

Data Hiding

- An object's attributes may or may not be visible outside the class definition.
- You need to name attributes with a double underscore prefix,
- and those attributes then will not be directly visible to outsiders.

ex

```
class JustCounter:
    __secretCount = 0

    def count(self):
        self.__secretCount += 1
        print (self.__secretCount)

counter = JustCounter()
counter.count()
counter.count()
# print (counter.__secretCount)
print (counter._JustCounter__secretCount)
```