C++與Python的比較

物件

- Python中，任何變數、方法皆為物件。
- C++中，物件為以類別為藍圖的變數與方法所成的集合。

- >>> s = ‘swing’ # ‘swing’ is an object of type ‘str’
- >>>type(s)
  - <type 'str'>
- >>> def hello(): pass # hello() is an object of a function
- >>>type(hello)
  - <type 'function'>

資料型態

- C++: 使用前需要宣告。

datatype.cpp

```
#include<string>
#include<iostream>
using namespace std;

int main(){
    int a = 5;
    string s("ababc abcdefg");
    cout << s << endl;
    cout << a << endl;
    return 0;
}
```

- Python: 使用前無須宣告。

datatype.py

```
a = 5
s = 'ababc abcdefg'
print s
print a
```

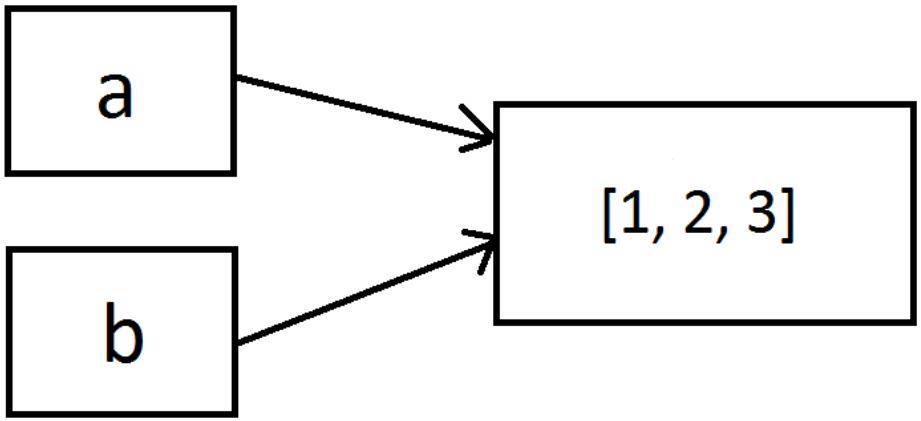
動態資料型態

- Python interpreter於程式執行時(runtime)才決定變數的資料型態。
- C++ compiler於程式編譯時(compilation time)即決定變數的資料型態。

- >>> s = 'swing'
- >>> type(s)
  - <type 'str'>
- >>> s = 2
- >>> type(s)
  - <type 'int'>

動態資料型態

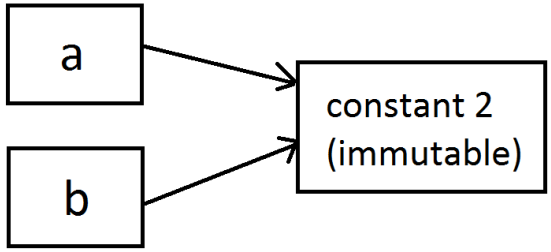
- Python的變數型態由該物件所對應的變數型態決定，此稱為「參考」。
- C++中對應的概念為「指標」。
- 用C++的語言來說，Python中所有的變數呼叫皆使用指標參照的方式達成。
- >>> a = [1,2,3]
- >>> b = a
- >>> a
  - [1, 2, 3]
- >>> b
  - [1, 2, 3]
- >>> b[0] = 4
- >>> a
  - [4, 2, 3]



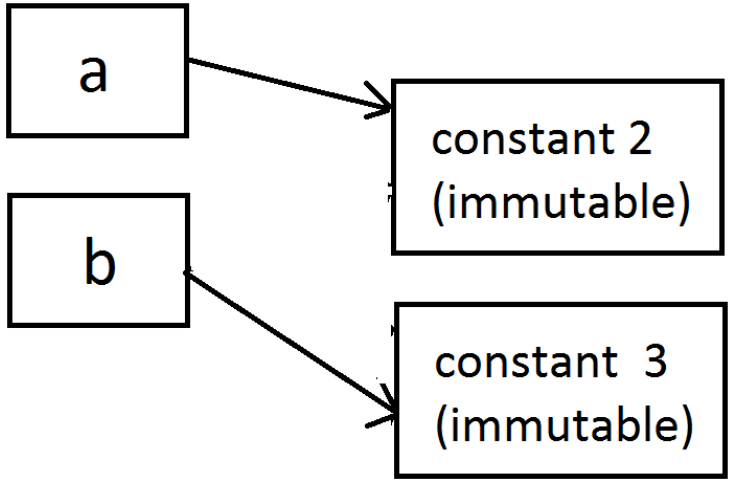
動態資料型態

- 當不可改變的物件被試圖修改，新的物件會自動被產生。

```
>>> a = 2
>>> b = a
>>> b
    – 2
>>> id(a)
    – 10834240
>>> id(b)
    – 10834240
>>> b = 3
>>> a
    – 2
>>> id(b)
    – 10834216
```



id returns the address of the object in memory.



程式碼區塊

- C++以大括號{}區分程式碼區塊，程式碼的縮排與否不會影響執行結果。

indent.cpp

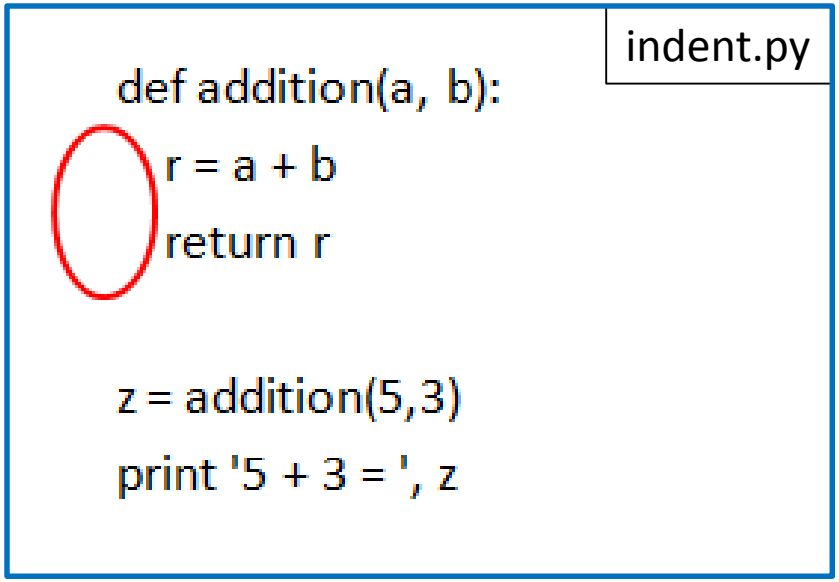
```
#include <iostream>
using namespace std;

int addition(int a, int b){
   int r;
   r = a + b;
   return r;
}

int main(){
   int z;
   z = addition(5,3);
   cout << "5 + 3 = " << z << endl;
   return 0;
}
```

程式碼區塊

- Python以縮排深度區分程式碼區塊，無需加上{}，但縮排錯誤會影響程式正確性。



高維函數

- Python中，一個函數可以直接做為另一個函數的引數。

higherorder.py

```
def summation(n, term):
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

def cube(x):
    return x * x * x

def sum_cubes(n):
    return summation(n, cube)

result = sum_cubes(4)
print 'result = ', result
```

高維函數

- C++中，欲將一個函數做為另一個函數的引數時，必須指定作為引數的函數的引數與回傳值的資料型態。

```
#include <iostream>
#include <cmath>
using namespace std;

int cube(int x){
   return pow(x,3);
}
int summation(int n, int term(int fv) ){
   int total = 0;
   int k = 1;
   while(k <= n){
      total = total + term(k);
      k++;
   }
   return total;
}
int sum_cubes(int n){
   return summation(n, cube);
}

int main(){
   int result = sum_cubes(4);
   cout << "result = " << result << endl;
   return 0;
}
```

higherorder.cpp

模組載入視野(Python)

- >>> import random
- >>> random.random()
  - 0.9609252011074219
- >>> random.randint(1,10)
  - 4
- >>> dir()
  - ['__builtins__', '__doc__', '__name__', '__package__', 'random']

載入一個模組，模組內的變數與方法不複製至目前物件中。

- >>> from random import *
- >>> random()
  - 0.5035802953796077
- >>> randint(1,10)
  - 9
- >>> dir()
  - ['Random', 'SystemRandom', 'WichmannHill', '__builtins__', '__doc__', '__name__', '__package__', 'betavariate', 'choice', 'expovariate', 'gammavariate', 'gauss', 'getrandbits', 'getstate', 'jumpahead', 'lognormvariate', 'normalvariate', 'paretovariate', 'randint', 'random', 'randrange', 'sample', 'seed', 'setstate', 'shuffle', 'triangular', 'uniform', 'vonmisesvariate', 'weibullvariate']

將所有的變數與方法複製至目前物件中。

變數視野

- C++與Python在變數視野的概念幾乎相同，只不過Python沒有關鍵字*static*，需用關鍵字global來取得被保護的全域變數。

變數視野 – C++

varscope.cpp

```
#include <iostream>
using namespace std;
void uselocal();
void usestaticlocal();
void useglobal();

int x = 1;
int main(){
   int x = 5;
   cout << "local x in main's outer scope is " << x << endl;
   {
      int x = 7;
      cout << "local x in main's inner scope is " << x << endl;
   }
   cout << "local x in main's outer scope is " << x << endl;

   uselocal();
   usestaticlocal();
   useglobal();
   uselocal();
   usestaticlocal();
   useglobal();

   cout << "\nlocal x in main is " << x << endl;
   return 0;
}
```

變數視野 – C++

varscope.cpp

```
void uselocal(){
   int x = 25;
   cout << endl << "local x is " << x
        << " on entering uselocal" << endl;

   ++x;
   cout << "local x is " << x
        << " on exiting uselocal" << endl;
}
void usestaticlocal(){
   static int x = 50;
   cout << endl << "local static x is " << x
        << " on entering usestaticlocal" << endl;
   ++x;
   cout << "local static x is " << x
        << " on exiting usestaticlocal" << endl;
}
void useglobal(){
   cout << endl << "global x is " << x
        << " on entering useglobal" << endl;
   x *= 10;
   cout << "global x is " << x
        << " on exiting useglobal" << endl;
}
```

變數視野 – C++ (執行結果)

```
local x in main's outer scope is 5
local x in main's inner scope is 7
local x in main's outer scope is 5

local x is 25 on entering uselocal
local x is 26 on exiting uselocal

local static x is 50 on entering usestaticlocal
local static x is 51 on exiting usestaticlocal

global x is 1 on entering useglobal
global x is 10 on exiting useglobal

local x is 25 on entering uselocal
local x is 26 on exiting uselocal

local static x is 51 on entering usestaticlocal
local static x is 52 on exiting usestaticlocal

global x is 10 on entering useglobal
global x is 100 on exiting useglobal

local x in main is 5
```

變數視野 – Python

varscope.py

```
x = 1
_x = 50

def uselocal():
    x = 25
    print 'local x is ', x, ' on entering uselocal'
    x += 1
    print 'local x is ', x, ' on exiting uselocal', '\n'

def usestaticlocal():
    global _x
    print 'local static x is ', _x, ' on entering usestaticlocal'
    _x += 1
    print 'local static x is ', _x, ' on exiting usestaticlocal', '\n'

def useglobal():
    global x
    print 'global x is ', x, ' on entering useglobal'
    x *= 10
    print 'global x is ', x, 'on exiting useglobal', '\n'
```

變數視野 – Python

varscope.py

```
def main():
   x = 5
   print "local x in main's outer scope is ", x

   def inner():
      x = 7
      print "local x in main's inner scope is ", x
   inner()

   print "local x in main's outer scope is ", x, '\n'

   uselocal()
   usestaticlocal()
   useglobal()
   uselocal()
   usestaticlocal()
   useglobal()

   print '\nlocal x in main is ', x

main()
```

變數視野 – Python (執行結果)

```
local x in main's outer scope is  5
local x in main's inner scope is  7
local x in main's outer scope is  5

local x is  25  on entering uselocal
local x is  26  on exiting uselocal

local static x is  50  on entering usestaticlocal
local static x is  51  on exiting usestaticlocal

global x is  1  on entering useglobal
global x is  10 on exiting useglobal

local x is  25  on entering uselocal
local x is  26  on exiting uselocal

local static x is  51  on entering usestaticlocal
local static x is  52  on exiting usestaticlocal

global x is  10  on entering useglobal
global x is  100 on exiting useglobal


local x in main is  5
```

變數指派

- C++中，多重變數指派必須分成多步驟完成。

```
#include <iostream>
using namespace std;

int fibonacci(int x){
   int a = 1;
   int b = 1;
   for(int i = 0; i < x; i++){
      int c = a;
      a = b;
      b = c + b;
   }
   return a;
}

int main(){
   for(int i = 1; i <= 20; i++){
      cout << fibonacci(i) << endl;
   }
   return 0;
}
```

fibo.cpp

變數指派

- Python中，多重變數可以使用一行程式碼同時指派。

fibo.py

```
def fibonacci(x):
    a,b = 1,1
    for i in range(x-1):
        a,b = b, a+b
    return a

fib = []

for i in range(1,21):
    fib.append(fibonacci(i))

print fib
```

Python 裝飾物

- C++沒有類似的概念。
- Python中，裝飾物提供一個非常有用的方法來增加函數與類別的功能性。
- 裝飾物是包裝其他函數與類別的函數。

Python 裝飾物 – 例 1

- 回傳值的裝飾

Deco1.py

```
def D1(fn):
   def newFn():
         return "<head>" + fn() + "<tail>"
   return newFn

@D1
def f1():
   return "a sentence"

print f1()
```

- 輸出:
  - <head>a sentence<tail>

Python 裝飾物 – 例 2

- 執行函數前後做些事情

Deco2.py

```
def D2(fn):
    def newFn():
        print "---before executing the function---"
        fn()
        print "---after executing the function---"
    return newFn

@D2
def f2():
    print "execute f2()"

f2()
```

- 輸出:
  - ---before executing the function---
  - execute f2()
  - ---after executing the function---

Python 裝飾物 – 例 3

- 變數檢查

Deco3.py

```
def D3(fn):
    def newFn(arg):
        if arg <= 0:
            print "error!!"
            return
        fn(arg)
    return newFn

@D3
def f3(n):
    print "%s is a positive number" % n

f3(-1)
f3(5)
```

- 輸出:
  - error!!
  - 5 is a positive number

列表理解

- Python 在變數指派時接受表達式, C++ 則否。

```
#include <iostream>
using namespace std;

int main(){
    int x[10] = {0};
    for(int i = 0; i < 10; i++){
        x[i] = i * i;
    }
    for(int j = 0; j < 10; j++){
        cout << x[j] << endl;
    }
    return 0;
}
```

listcompre.cpp

```
y = [x * x for x in range(10)]
print y
```

listcompre.py

語法比較

Syntax	Python	C++
包含一個模組/圖書館	import *	#include *
for 敘述	for i in range(10):	for(i = 0; i < 10; i++)
while 敘述	while x != 3:	while (x != 3)
if 敘述	if x != 3:	if (x != 3)
函數定義	def myfunction():	int myfunction()
and / or / not 操作器	and / or / not	&& / || / !
單行註解	#	//
程式碼區塊	縮排	{ }
敘述分隔器	EOL	;
常數	沒有	const int a = 2.71828
輸入輸出	raw_input, input, print	cin, cout, etc.

二維陣列 – 定義與型態

- C++: 固定大小，變數型態需要宣告。
  - int a[3][3] = {{1,2,3},{4,5,6},{7,8,9}};

- Python: 大小可變，不需宣告變數型態。
  - a = [[1,2,3],[4,5,6],[7,8,9]]
  - a += [[9,10,11]]
  - a
    - [[1, 2, 3], [4, 5, 6], [7, 8, 9], [9, 10, 11]]

二維陣列 – 變數指派

- C++: 一個一個變數指派(通常使用迴圈)。

```
#include<iostream>
using namespace std;

int main(){

   int a[3][3]={};
   int x = 1;

   for(int i = 0; i < 3; i++){

      for(int j = 0; j < 3; j++){

         a[i][j] = x + j;
      }

      x++;
   }
   for(int k = 0; k < 3; k++){
      for(int l = 0; l < 3; l++){
         cout << a[k][l] << '\t';
      }
      cout << endl;
   }

   return 0;
}
```

2dvarassign.cpp

二維陣列 – 變數指派

- Python: 可使用列表理解指派變數。

```
a = [[i+1+j for i in range(3)] for j in range(3)]
print a
```

2dvarassign. py

函數中的靜態變數

- C++: 使用關鍵字*static*來定義。
- Python: 使用關鍵字*global*來取得被保護的變數。

```
#include<iostream>
using namespace std;

int generate(){
    static int sid = 0;
    return ++sid;
}

int main(){
    cout << generate() << endl;
    cout << generate() << endl;
    cout << generate() << endl;
    return 0;
}
```

staticvar.cpp

```
_sid = 0

def generate():
    global _sid
    _sid += 1
    return _sid

print generate()
print generate()
print generate()
```

staticvar.py

變數空間大小

- C++: 固定大小。
- Python: 記憶體有多大就能夠儲存多大。

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	Range	0 to 65,535
signed short int	Range	-32768 to 32767
long int	4bytes	-2,147,483,647 to 2,147,483,647
signed long int	4bytes	same as long int
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)

變數空間大小

C++

```
#include<iostream>
#include<cmath>
using namespace std;

int main(){
    int a = 2;
            int b = pow(a,10);
           long c = pow(a,20);
    unsigned long d = pow(a,200);
    cout << b << endl;
    cout << c << endl;
    cout << d << endl;
    return 0;
}
```

largevar.cpp

```
1024
1048576
0
```

Python

```
b = 2 ** 10
c = 2 ** 20
d = 2 ** 200

print b
print c
print d
```

largevar.py

```
1024
1048576
1606938044258990275541962092341162602522202993782792835301376
```

輸入與輸出 – C++

- 使用iostream標頭中定義的cin與cout，變數型態需指定；若輸入的不是所指定的變數型態則會出現錯誤。

ioex.cpp

```
#include<iostream>
using namespace std;

int main(){
   int x = 0;
   cout << "enter: " << endl;
   cin >> x;
   cout << "you entered: " << x << endl;
   return 0;
}
```

```
enter:
1234
you entered: 1234

enter:
xxxx
you entered: 0
```

輸入與輸出 – Python

- 使用 __builtins__ 模組所定義的input, raw_input, 與print。
  - input將輸入視為**數字**。
  - raw_input將輸入視為**文字**。

```
x = raw_input("enter: ")
print 'you entered: ', x


執行結果
enter: 222
you entered:  222

enter: rrrr
you entered:  rrrr
```

```
x = input("enter: ")
print 'you entered: ', x


執行結果
enter: 222
you entered:  222

enter: rrrr
Error!
```

浮點數精準度問題

- 浮點數儲存在記憶體裡以二進位表示，但小數部分有些無法已有限位數的二進位表示出來。
  - $0.5_{(10)} = 0.1_{(2)}$
  - $10.8125_{(10)} = 1010.1101_{(2)}$
  - $0.1_{(10)} = 0.0001100110011...{}_{(2)}$
- 所以，無法表式成2的乘冪的和的小數在浮點數運算之後於小數點後多位之處便會產生誤差。
- C++顯示時使用cout會自動修正。
- Python使用print時不會自動修正，必許指定有效位數或使用四捨五入後的值顯示。

浮點數精準度問題(C++)

precision.cpp

```
#include <iostream>
using namespace std;

int main(){
   float a = .1;
   float b = .1;
   float c = .2;
   float d = .4;
   float f = .3;

   float e = a + b + c - d;
   float g = a + c - f;

   cout << a << " + " << b << " + " << c << " - " << d << " = " << e << endl;
   cout << a << " + " << c << " - " << f << " = " << g << endl;

   return 0;
}
```

```
0.1 + 0.1 + 0.2 - 0.4 = 0
0.1 + 0.2 - 0.3 = 0
```

浮點數精準度問題(Python)

precision.py

```
a = .1
b = .1
c = .2
d = .4
f = .3

e = a + b + c - d
g = a + c - f

print a, '+', b, '+', c, '-', d, '=',e
print a, '+', c, '-', f, '=', g, ' ~= ', '%.2f' % g
```

```
0.1 + 0.1 + 0.2 - 0.4 = 0.0
0.1 + 0.2 - 0.3 = 5.55111512313e-17  ~=  0.00
```