Python課程資訊

課程內容

1. Introduction to Python
2. Variables and Scope
3. Selection Control Statements
4. Collections
5. Loop Control Statements
6. Errors and Exceptions
7. Functions
8. Classes and Object-Oriented Programming
9. 期中考
10. Useful Modules in the Standard Library
11. Introduction to GUI programming
12. Scientific Computing with Numpy and SciPy
13. Plotting Using Matplotlib
14. Concurrent and Parallel Programming
15. 期末考

評分

1. 期中考:25%
2. 期末考:25%
3. 作業:50%

課本

1. Brett Slatkin, Object-Oriented Programming in Python

參考資料

1. Brett Slatkin, Effective Python: 59 Specific Ways to Write Better Python, Pearson, 2015.
2. David I. Schneider, An Introduction to Programming Using Python, Pearson, 2016
3. 林信良, Python3.5技術手冊, 碁峯, 2016.
4. 何敏煌,Python程式設計實務,博碩, 2016.

安裝

- Anaconda: https://www.anaconda.com/download/ (https://www.anaconda.com/download/)
- Python official web site:https://www.python.org (https://www.python.org)

Python IDEs

- Spyder
- Jupyter Notebook
- Visual Studio Code: https://code.visualstudio.com/docs/languages/python (https://code.visualstudio.com/docs/languages/python)

A Taxonomy of Programming Languages

程式設計範式(programming paradigm)

宣告式語言 (The declarative language)

- Functional e.g., Lisp/Scheme, ML, …
- Dataflow e.g., Id, Val, …
- Logic e.g., Prolog, …
- Template-based e.g., XSLT, …

命令式語言 (The imperative language)

- Procedural e.g., C, Fortran, …
- Object-oriented e.g., Smalltalk, Java, C++, …

一個程式語言可以同時擁有多種程式設計範式特徵

Python的程式設計範式特徵包含

- Procedural
- Object-oriented
- Functional

編譯式程式語言(the compiled language)與解譯式程式語言(the interpreted language)

- A compiled language, such as C, C++, comes with a compiler, which translates the programs into executable binary files.
- An interpreted language, such as Python, comes with an interpreter, which interprets and executes programs lines-by-line.
- Interpreted languages are usually slower than compiled languages.

The Python Programming Language

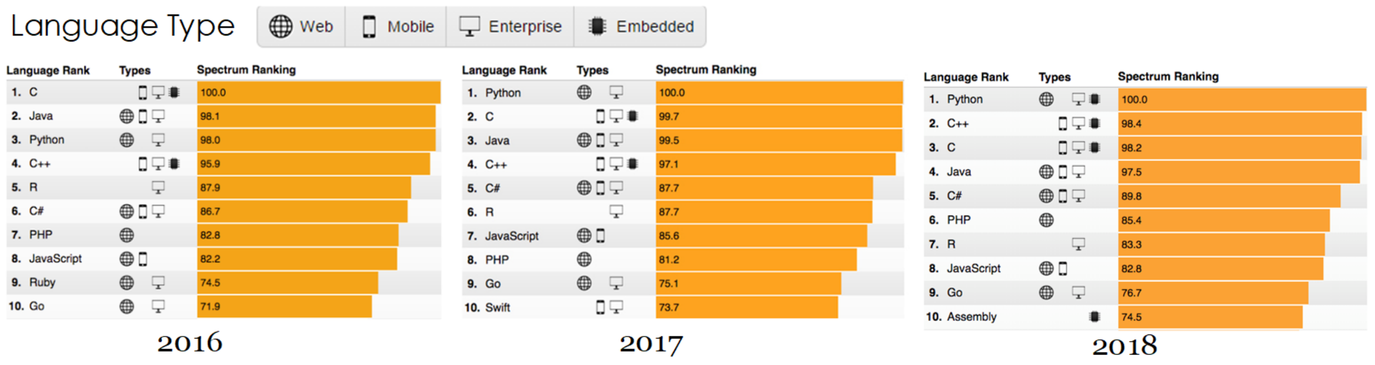
Features

- Python is a general-purpose interpreted language.
- Python has features of procedural, object-oriented, and functional programming paradigms.
- Python was originally created in the late 1980s and became popular in the 2000s after the release of version 2.0.
- Python is known for its clear, simple syntax and its dynamic typing – the same variables in Python can be reused to store values of different types. This is not allowed in a statically-typed language like C.

Python 2 vs Python 3

Python 2.7 will not be supported in 2020.

IEEE Spectrum’s Top 10 Languages


Language Type
🌐 Web
📱 Mobile
🖥 Enterprise
▪ Embedded

Language Rank	Types	Spectrum Ranking
1. C	📱🖥▪	100.0
2. Java	🌐📱🖥	98.1
3. Python	🌐🖥	98.0
4. C++	📱🖥▪	95.9
5. R	🖥	87.9
6. C#	🌐📱🖥	86.7
7. PHP	🌐	82.8
8. JavaScript	🌐📱	82.2
9. Ruby	🌐🖥	74.5
10. Go	🌐🖥	71.9

2016

Language Rank	Types	Spectrum Ranking
1. Python	🌐🖥	100.0
2. C	📱🖥▪	99.7
3. Java	🌐📱🖥	99.5
4. C++	📱🖥▪	97.1
5. C#	🌐📱🖥	87.7
6. R	🖥	87.7
7. JavaScript	🌐📱	85.6
8. PHP	🌐	81.2
9. Go	🌐🖥	75.1
10. Swift	📱🖥	73.7

2017

Language Rank	Types	Spectrum Ranking
1. Python	🌐🖥▪	100.0
2. C++	📱🖥▪	98.4
3. C	📱🖥▪	98.2
4. Java	🌐📱🖥	97.5
5. C#	🌐📱🖥	89.8
6. PHP	🌐	85.4
7. R	🖥	83.3
8. JavaScript	🌐📱	82.8
9. Go	🌐🖥	76.7
10. Assembly	▪	74.5

2018

Python安裝

- 由https://www.anaconda.com/download/#download (https://www.anaconda.com/download/#download) 下載安裝程式，按照步驟安裝。
- 檢查Python版本

In [1]:

```python
import sys
print(sys.version_info)
print(sys.version)
```

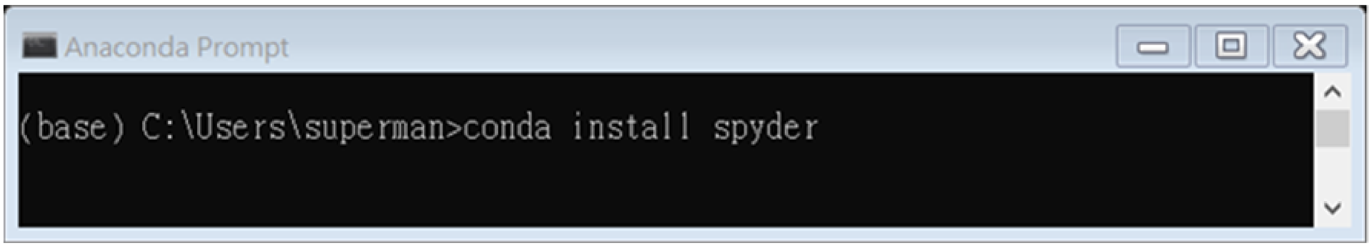
```
sys.version_info(major=3, minor=5, micro=5, releaselevel='final', serial=0)
3.5.5 |Anaconda, Inc.| (default, Apr  7 2018, 04:52:34) [MSC v.1900 64 bit
(AMD64)]
```

安裝其他Python程式模組。

Install a package

- conda install -c menpo opencv
- conda install -n myenv spyder
- pip install packagename

https://conda.io/docs/commands/conda-install.html (https://conda.io/docs/commands/conda-install.html)



```
(base) C:\Users\superman>conda install spyder
```

建立虛擬環境

針對不同的Python模組或專案，可建立各自虛擬環境，以免因版本問題互相衝突。

Create a virtual environment

- conda create -n env
- conda create python=3 --name env

Activate a virtual environment

- activate env

Deactivate a virtual environment

- deactivate env

Rename a virtual environment

以下面兩個指令完成改虛擬環境名稱

- conda create --name new_name --clone old_name
- conda remove –name old_name --all

Create a virtual environment in a specific directory

- conda create --prefix /new_directory/env_name python=3
- This command will create the environment named /new_directory/env_name which resides in /new_directory instead of the default e.g., E:\ProgramData\Anaconda3\envs

---

Hello Python

學一個新的程式語言，第一步先試著輸出/入訊息，確認撰寫程式環境一切無誤。

- Python使用input函式輸入資料
- Python使用print函式輸出資料
  下面Python範例輸出一個訊息。

In [2]:

```python
print('Hello Python')
```

```
Hello Python
```

下面Python範例輸入與輸出一個訊息。注意input可以顯示提示訊息。

In [3]:

```
# this is a comment
name = input("name:")
age  = int(input("age:"))
if age >= 18:
    print("Hello ", name)
```

```
name:david
age:20
Hello  david
```

In [4]:

```
# this is a comment
name = input("name:"); age = int(input("age:"))
if age >= 18:
    print("Hello ", name)
```

```
name:david
age:20
Hello  david
```

根據此程式語言程式設計範式，學習如何以此程式語言表達那些概念，如此就可以很快的上手此程式語言。但是還是要花時間才能精熟。在這裡提醒同學:

1. 在Jupyter notebook裡，可以方便呈現想法、公式、簡單程式碼、及結果。若是要撰寫複雜一點的程式，好一點的程式除錯環境是必要的，這時還是用spyder這種IDEs來開發程式比較恰當。
2. 撰寫Python要有良好的程式撰寫紀律(discipline)。由於Python有很多模組，使得複雜工作只要幾行程式就完成，這時拙劣的程式與資料結構看不出來有嚴重影響。不過當程式規模變大，良好的Python程式撰寫紀律是必要的，不然會造成程式除錯及維護極大的困擾。參考PEP8 http://www.python.org/dev/peps/pep-0008/ (http://www.python.org/dev/peps/pep-0008/)

Python變數名稱命名慣例

- constants: I_AM_A_CONSTANT
- variables: i_am_a_variable
- Class names: ClassName
- Private variables: _private_variable

區分大小寫

Python Keywords

False class finally is return None continue for lambda try True def from nonlocal while and del global not with as elif if or yield assert else import pass break except in raise

Python裡keyword不可做為變數名稱

The Procedural Feature of Python

Python變數

1. Dynamically typed: We can use the same variable to store values of different types
2. Scope rules
3. Lifetime
4. Binding rules

- 不用宣告變數，但在使用前必須先建構或給初值;
- 變數名稱可以是a-z,A-Z,0-9,_的組合,但是第一個字元不可以是0-9字元

Data Types

A type consists of two parts: (1) a domain of possible values; (2) a set of operations on the values

In [5]:

```
# The type of an object can be shown by the type function.
print(type(1),type(complex(1,3)),type(dict()))
```

```
<class 'int'> <class 'complex'> <class 'dict'>
```

In [6]:

```python
# The operations on the type can be shown by the dir function.
print(dir(1))
print(dir(complex(1,3)))
print(dir(dict()))
```

```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__de
lattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floo
r__', '__floordiv__', '__format__', '__ge__', '__getattribute__', '__getnewa
rgs__', '__gt__', '__hash__', '__index__', '__init__', '__int__', '__invert_
_', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg
__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__r
divmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rl
shift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrs
hift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr_
_', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__',
'__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator', 'from_byte
s', 'imag', 'numerator', 'real', 'to_bytes']
['__abs__', '__add__', '__bool__', '__class__', '__delattr__', '__dir__', '_
_divmod__', '__doc__', '__eq__', '__float__', '__floordiv__', '__format__',
'__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__hash__', '__ini
t__', '__int__', '__le__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg_
_', '__new__', '__pos__', '__pow__', '__radd__', '__rdivmod__', '__reduce_
_', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rmod__', '__rmul__', '_
_rpow__', '__rsub__', '__rtruediv__', '__setattr__', '__sizeof__', '__str_
_', '__sub__', '__subclasshook__', '__truediv__', 'conjugate', 'imag', 'rea
l']
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__do
c__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr_
_', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'co
py', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'up
date', 'values']
```

Built-in Data Types of Python:

- Simple data types
  - int
  - float
  - bool
  - complex
  - str
- Collections
  - range
  - list
  - tuple
  - set
  - dict

改變變數內容

- 使用assignment operator =

- 使用compound assignment operator +=, -=, *=, /=, **=

Python有兩種數值型態: 整數(int,沒有小數點)與浮點數(float)，有下面運算

1. 加 +
2. 減 -
3. 乘 *
4. 除(浮點數) / ; 除(整數) //
5. 次方 **
6. 求餘數 % (整數獨有)

Python 整數沒有限制整數大小

Operation	**Symbol**	**Example**	**Result**
Addition	+	28 + 10	38
Subtraction	-	28 - 10	18
Multiplication	*	28 * 10	280
Division	//	28 // 10	2
Modulus (remainder)	%	28 % 10	8
Exponent (power)	**	28**10	296196766695424

In [7]:

```python
print(2**100)
```

```
1267650600228229401496703205376
```

常數寫法 **(#後面是註解)**

- 0b1010 # binary number
- 0xABCD # hexadecimal number
- 0o7777 # octonary number

In [8]:

```python
print(3+2, 3//2, 3/2, 3**2, 3-2, 3 % 2)
```

```
5 1 1.5 9 1 1
```

In [9]:

```python
print(3+2,',', 3//2, ',',3/2, ',',3**2, ',',3-2, ',',3 % 2)
```

```
5 , 1 , 1.5 , 9 , 1 , 1
```

In [10]:

```
print(3+2, 3//2, 3/2, 3**2, 3-2, 3 % 2,sep=',')
```

```
5,1,1.5,9,1,1
```

In [11]:

```
print('{},{},{},{},{},{}'.format(3+2, 3//2, 3/2, 3**2, 3-2, 3 % 2))
```

```
5,1,1.5,9,1,1
```

In [12]:

```
x = 1
x = x + 1
print(x)
x += 1
print(x)
x = x ** 2
print(x)
x **= 2
print(x)
```

```
2
3
9
81
```

Floating-point numbers

- Arithmetic operations (*, , /, %, +, -) for floating-point numbers are the same as those for integers.
  - Use float(x) to convert another object to a floating-point number.
  - Use the string formatting to display a floating-point number.
- Python also provides the complex number type.
  1+2j represents a complex number with real part 1 and imaginary part 2.
- The fractions and decimal modules provide rational numbers and decimal floating-point arithmetic.

In [13]:

```
# This will print 12.35
print("%.2f" % 12.3456)
# This will print 1.234560e+01
print("%e" % 12.3456)
```

```
12.35
1.234560e+01
```

In [14]:

```
# This will print 12.35
print("{:.2f}".format(12.3456))
# This will print 1.234560e+01
print("{:e}".format(12.3456))
```

```
12.35
1.234560e+01
```

Boolean

- A bool variable has two possible values: True or False.
- The Boolean values for None, False, 0, 0.0, 0j, '' (empty string), (), [], {} are False, and the Boolean values for the other values are True.

In [15]:

```
a=[]
print(bool(a))
```

```
False
```

In [16]:

```
a=''
print(bool(a))
```

```
False
```

In [17]:

```
a=0.0
print(bool(a))
```

```
False
```

In [18]:

```
a=None
print(bool(a))
```

```
False
```

In [19]:

```
a=[1,2]
print(bool(a))
```

```
True
```

In [20]:

```
a=1
print(bool(a))
```

```
True
```

Python字串

- 使用單引號'字串'
- 使用雙引號"字串"
- 使用3單引號可以跨行 """hello world"""
- 單一字元視為長度為1的字串
  Use ord(x) and chr(49) to get the code of x and the character of 49.
- By adding an r before the opening quote of the string, the contents of the string are not translated and the backslashes have no special meaning.

Sequence	Meaning	Sequence	Meaning
\	literal backslash	\ooo	Character with oct value oo
\'	single quote	\xhh	Character with hex value hh
\"	double quote	\uhhhh	Character with 16-bit hex value hhhh
\n	newline	\uhhhhhhhh	Character with 32-bit hex value hhhhhhhh
\t	tab		

In [21]:

```
msg1 = 'hello'
print(msg1)
msg2 = "hello world"
print(msg2)
msg3='''hello   hello hello "hello" 'hello'
hello world
     hello world'''
print(msg3)
```

```
hello
hello world
hello   hello hello "hello" 'hello'
hello world
     hello world
```

字串串接與重複

- 字串串接使用+
- 字串重複使用*

In [22]:

```
msg = 'hello'+' world '
print(msg)
msg1 = msg*3
print(msg1)
```

```
hello world
hello world hello world hello world
```

字串足標 ¶

足標由0開始算，負足標由字串尾部開始算

In [23]:

```
print(msg3[2:10]) #顯示msg3前2~9字元的字串
```

```
llo   he
```

In [24]:

```
print(msg3[-10:]) # 顯示msg3後10~最後一個字元的字串
```

```
ello world
```

In [25]:

```
print(msg1.find('h')) #由左至右找第1個h
print(msg1.find('l')) #由左至右找第1個l
print(msg1.rfind('l'))#由右至左找第1個l
```

```
0
2
33
```

字串常用函式

In [26]:

```
msg = '''introduction to Python
'''
print(len(msg)) #長度
print('[',msg.upper(),']',sep='') #變大寫
print('[',msg.lower(),']',sep='') #變小寫
print('[',msg.capitalize(),']',sep='') #首字大寫，其他小寫
print('[',msg.title(),']',sep='') #每個子首字大寫，其他小寫
print('[',msg.rstrip(),']',sep='') #移除字串尾空白與換行字元
```

```
26
[INTRODUCTION TO PYTHON
]
[introduction to python
]
[Introduction to python
]
[Introduction To Python
]
[introduction to Python]
```

In [27]:

```
print("123456".startswith("12")) #是否12開始
print("123456".endswith("12")) #是否12結束
```

```
True
False
```

字串轉數值

In [28]:

```
print(int("123")+int("456"))
print(float("100.0")+20)
print(eval("123"))
print(eval("123/456"))
x = 5
print(eval("2*x+x-1"))
```

```
579
120.0
123
0.26973684210526316
14
```

數值轉字串

In [29]:

```
msg = str(123)
print(msg*3)
```

```
123123123
```

格式化

- Look at here https://pyformat.info (https://pyformat.info) for more new features of string formatters

In [30]:

```
name = "Jane"
age = 23

# old style
print('-'*10+ ' old style '+'-'*10)
print("Hello! My name is %s." % name)
print("Hello! My name is %s and I am %d years old." % (name, age))

# new style
print('-'*10+' new style '+'-'*10)
print("Hello! My name is {} and I am {} years old.".format(name, age))
print("Hello! My name is {0} and I am {1} years old.".format(name, age))
print("Hello! My name is {0:2} and I am {1:5d} years old.".format(name, age))
print("Hello! My name is {n} and I am {a} years old.".format(n=name, a=age))
```

```
---------- old style ----------
Hello! My name is Jane.
Hello! My name is Jane and I am 23 years old.
---------- new style ----------
Hello! My name is Jane and I am 23 years old.
Hello! My name is Jane and I am 23 years old.
Hello! My name is Jane and I am    23 years old.
Hello! My name is Jane and I am 23 years old.
```

---

list與tuple

- 使用list,tuple來儲存資料與物件。
- list用[]表示，tuple用()表示，但是tuple不可以改變其內容。下面a為list，b為tuple
  a=[1,2,3,4]
  b=(1,2,3,4)
- list,tuple裡的物件可以是不相同的型態。
- list,tuple可以用範圍m:n的方式處理。
  - list1[m:n]指list1[m],...,list[n-1]。
  - list1[:m]指list1[0],...,list[m-1]。
  - list1[m:]指list1[m],...,最後一個。
  - list1[:]指list1[0],...,最後一個。

In [31]:

```
list1=[1,0.2,3,'world',5,'hello']
list2=[1,'hello',1.3,list1]
print(list1)
print(list1[0])
print(list1[1:3])
print(list1[0::2])
print(list1[-1::-1]) # 反轉
print(list2) # nested list
print(list1+list2) # list串接
print([1,2,3]*3) # list重複
```

```
[1, 0.2, 3, 'world', 5, 'hello']
1
[0.2, 3]
[1, 3, 5]
['hello', 5, 'world', 3, 0.2, 1]
[1, 'hello', 1.3, [1, 0.2, 3, 'world', 5, 'hello']]
[1, 0.2, 3, 'world', 5, 'hello', 1, 'hello', 1.3, [1, 0.2, 3, 'world', 5, 'h
ello']]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

list常用函式

- len
- max/min/sum
- count
- index
- clear
- append 注意與extend的差異
- extend 注意與append的差異
- insert

In [32]:

```
numbers=[1,2,3,4,5,1,0,-1,-2,-3]
print(len(list1)) # list元素個數
print(len(list2))
print(len(numbers))
print(max(numbers)) #最大的
print(min(numbers)) #最小的
print(sum(numbers)) #和
print(numbers.count(1)) # 1有幾個
print(numbers.index(1)) # 第一個1出現在第幾號
```

```
6
4
10
5
-3
10
2
0
```

In [33]:

```
numbers.clear()
print(numbers)   # 清除所有元素
numbers.append(1)# 從尾端加入一個元素
print(numbers)
numbers.append([7,9,10])# 從尾端加入一個元素[7,9,10]
print(numbers)
numbers.extend([7,9,10]) # 從尾端加入一list個元素
print(numbers)
numbers.insert(1,12) #1號位置插入12
print(numbers)
del numbers[2] #刪除第2號元素
print(numbers)
print([1,2,3]+[4,5,6,7]) # list串接
ones = [1]*5 # list重複
print(ones)
```

```
[]
[1]
[1, [7, 9, 10]]
[1, [7, 9, 10], 7, 9, 10]
[1, 12, [7, 9, 10], 7, 9, 10]
[1, 12, 7, 9, 10]
[1, 2, 3, 4, 5, 6, 7]
[1, 1, 1, 1, 1]
```

字串與list

- str1.split(x): 將字串str1以x字元拆解成list
- ','.join(list1):將list1以','串接

In [34]:

```
str1='123,456,789'
list1=str1.split(',')
print(list1)
str2 =','.join(list1)
print(str2)
```

```
['123', '456', '789']
123,456,789
```

Immutable and Mutable Objects

- 字串，數值，tuple是immutable object
- list等其他物件為mutable object

下面例子，list2不是list1的複製，他們指向同一物件，所以更改list2如同改list1

In [35]:

```
list1 = [1,2,3,4]
list2 = list1
list2[1] = 5
print(list1)
print(list2)
```

```
[1, 5, 3, 4]
[1, 5, 3, 4]
```

下面例子，list2是list1的複製，他們指向不同一物件，所以更改list2不會改到list1

In [36]:

```
list1 = [1,2,3,4]
list2 = list1.copy() # 或是 list(list1)
list2[1] = 5
print(list1)
print(list2)
```

```
[1, 2, 3, 4]
[1, 5, 3, 4]
```

關係運算子

- 相等==
- 不相等!=
- 小於<
- 大於>
- 小於等於<=
- 大於等於>=
- in
- not in

In [37]:

```
print(3 < 4)
x = 3 < 4
print(x)
x = (3 < 4) and (2 < 1)
print(x)
print(1 in [1,2,3,4])
```

```
True
True
False
True
```

關係運算子可用於list與tuple

In [38]:

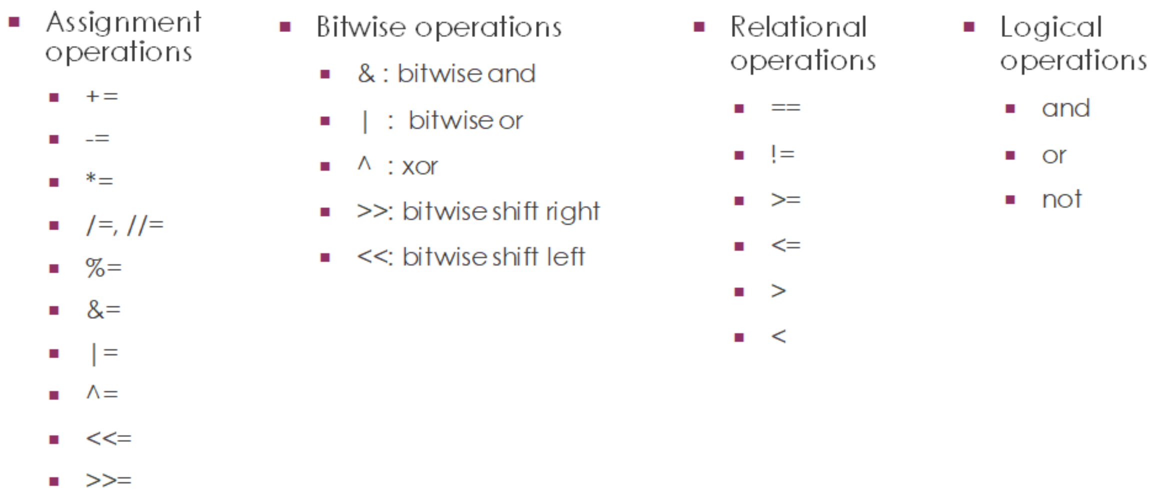
```
print([4,2,6,7]<[4,5,6])
print(4 in [1,2,3,4])
print(5 in [1,2,3,4])
print([1,2] in [1,2,3,4])
print([1,2] in [[1,2],3,4])
a=[1,2,3,4]
print(a in [1,2,3,4])
print(a in [[1,2,3,4]])
```

```
True
True
False
False
True
False
True
```

邏輯運算子

- and
- or
- not

整理: Bitwise, assignment, relational, logical operators



- Assignment operations
  - +=
  - -=
  - *=
  - /=, //=
  - %=
  - &=
  - |=
  - ^=
  - <<=
  - >>=
- Bitwise operations
  - & : bitwise and
  - | : bitwise or
  - ^ : xor
  - >>: bitwise shift right
  - <<: bitwise shift left
- Relational operations
  - ==
  - !=
  - >=
  - <=
  - >
  - <
- Logical operations
  - and
  - or
  - not

Flow-control statements

Selection statement

- if-else statement
- if-elif statement

Loop control

- while-loop
- for-loop
- pass; continue; break;

Procedure

```
def function_name(parameters):
    function body
    return value
```

---

if敘述

語法:注意:與對齊

> if condition:
>
> > indented block of statements

> else:
>
> > indented block of statements

if elif敘述

語法:注意:與對齊

> if condition1:
>
> > indented block of statements

> elif condition2:
>
> > indented block of statements

> elif condition3:
>
> > indented block of statements

> else:
>
> > indented block of statements

---

while迴圈

語法:注意:與對齊

> while condition:
>
> > indented block of statements

In [39]:

```
i=0
while i < 5:
    print(i)
    i+=1
```

```
0
1
2
3
4
```

break 敘述

跳出迴圈

continue敘述

跳過後面指令，進行下一次迴圈

pass敘述

不做任何事

In [40]:

```
z = 0
while z < 10:
  if z<5:
    z += 1
    continue
  print(z)
  z+=1
```

```
5
6
7
8
9
```

---

for迴圈

語法(注意冒號與縮排):

> for var in sequence:
>
> > indented block of statement

sequence可以是

- string
- list
- generator
- file object

In [41]:

```
for i in range(3, 7):
    print(i)
```

```
3
4
5
6
```

In [42]:

```
for i in range(3,7,2):
    print(i)
```

```
3
5
```

In [43]:

```
for i in range(3,-1,-1):
    print(i)
```

```
3
2
1
0
```

In [44]:

```
for i in 'abcde':
    print(i)
```

```
a
b
c
d
e
```

In [45]:

```
for i in ['hi',1,'oop',2]:
    print(i)
```

```
hi
1
oop
2
```

In [46]:

```
list1=[1,3.,'oop',[1,2]]
for idx, val in enumerate(list1):
    print(idx,val)
```

```
0 1
1 3.0
2 oop
3 [1, 2]
```

In [47]:

```
list2=['a','b']
for a1,a2 in zip(list1,list2):
    print(a1,a2)
```

```
1 a
3.0 b
```

函式

語法:
def functionName(par1,par2,par3=defaultValue,par4=defaultValue2):

> indented block of statememts
> return expression

呼叫函式時par1,par2照順序傳遞(positional parameters)，par3與par4為keyword parameter，可以不照順序，但是要寫參數名稱。

In [48]:

```
def mystery(a,b,c=1):
    return a+b*c

print(mystery(10,2))
print(mystery(10,2,c=3))
```

```
12
16
```

若是函式欲使用全域變數，在函式裡用global宣告變數為全域變數。

In [49]:

```
x=5
y=1
def variableScope():
    print(x)
    # x=1 #若這行沒變成註解，x會被視為local variable，這個程式會產生UnboundLocalError
    z=y #可以讀global variable y
    print(z)
    return

def globalVariable():
    global x,y #直接宣告x,y為全域變數，清楚明確
    print(x)
    x=1
    z=y
    print(z)
    return

variableScope()
globalVariable()
```

```
5
1
5
1
```

檔案處理

檔案輸出

步驟

- 開啟輸出檔: open

- 輸出: print/write
- 關閉檔案: close

如C一般的寫法

In [50]:

```
out = open('myfile.txt','w')
for i in range(100):
    print(i,file=out)
out.close()
```

使用with敘述，在離開with的區塊會自動關閉檔案

In [51]:

```
with open('myfile.txt','w') as out:
    for i in range(100):
        print(i,file=out)
```

可以使用write instance method取代print

In [52]:

```
with open('myfile.txt','w') as out:
    for i in range(100):
        out.write(str(i)+'\n')
```

檔案輸入

In [53]:

```
infp = open('myfile.txt','r')
A = []
while True:
    a= infp.readline().strip()
    if a:
        A.append(a)
    else:
        break
infp.close()
print(len(A),A)
```

```
100 ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '1
3', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25',
'26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '37', '3
8', '39', '40', '41', '42', '43', '44', '45', '46', '47', '48', '49', '50',
'51', '52', '53', '54', '55', '56', '57', '58', '59', '60', '61', '62', '6
3', '64', '65', '66', '67', '68', '69', '70', '71', '72', '73', '74', '75',
'76', '77', '78', '79', '80', '81', '82', '83', '84', '85', '86', '87', '8
8', '89', '90', '91', '92', '93', '94', '95', '96', '97', '98', '99']
```

In [54]:

```
A=[]
with open('myfile.txt','r') as infp:
    for a in infp.readlines():
        A.append(a.strip())
print(len(A),A)
```

```
100 ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '1
3', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25',
'26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '37', '3
8', '39', '40', '41', '42', '43', '44', '45', '46', '47', '48', '49', '50',
'51', '52', '53', '54', '55', '56', '57', '58', '59', '60', '61', '62', '6
3', '64', '65', '66', '67', '68', '69', '70', '71', '72', '73', '74', '75',
'76', '77', '78', '79', '80', '81', '82', '83', '84', '85', '86', '87', '8
8', '89', '90', '91', '92', '93', '94', '95', '96', '97', '98', '99']
```

Modules

眾多模組是Python受歡迎的一個重要原因。在安裝完那些模組後，程式只要透過import指令便可以使用那些模組

- import module_name [as alias]
- from module_name import something in this module [as alias]
- from module_name import *
- from mymodule import MyClass as FirstClass
- from myothermodule import MyClass as OtherClass
- from mymodule import $*$

In [55]:

```
import datetime
today = datetime.date.today()

import datetime as dt
today = dt.date.today()

from datetime import date
today = date.today()
```

小結

掌握上述Python特性，大概可以用Python描述出可用 C 語言所能表達的程序性敘述。注意Python畢竟是解譯式程式語言，撰寫Python程式時，更需要講究執行效能的寫法。