Data Processing Using Python

Walk into Python

ZHANG Dazhuang
Department of Computer Science and Technology
Department of University Basic Computer Teaching

Data Processing Using Python

1

AN INTRODUCTION TO PYTHON

What is Python

Ease to use, similar to script languages and interpretive programming languages

It has all the powerful functions of a traditional compiler language

Python is an interpretative, object-oriented, high-level programming language with dynamic semantics.

Birth of Python

- The first Python compiler/interpreter was born in 1991
- The name of Python comes from Guido's beloved TV show Monty Python's Flying Circus
- Python is between C and Shell, comprehensive, easy to learn, and extensible

History of Python

- Glue Language

It is easy to connect to and integrate with other well-known program languages (like C/C + +)

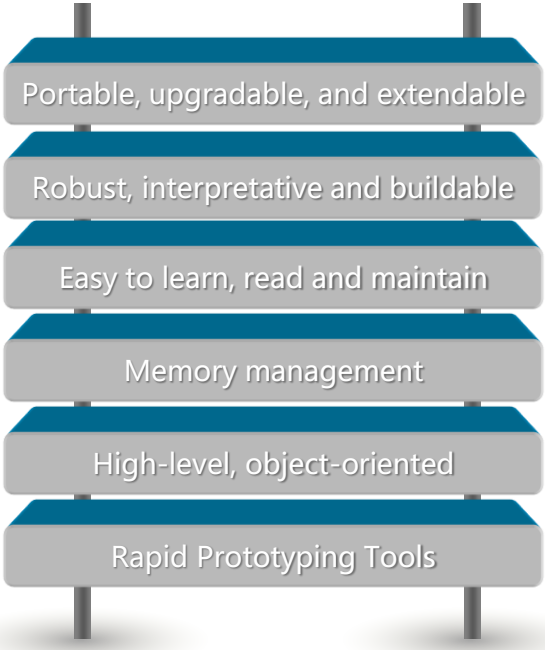
- Script Language

Advanced script language, which is more powerful than general script languages that can handle only simple tasks

- Object-Oriented Language

Full support to inheritance, overload, derivation, and multiple inheritance

Features of Python



Development of Python

Worldwide, Mar 2017 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Java	22.7 %	-1.4 %
2		Python	15.0 %	+3.0 %
3		PHP	9.3 %	-1.2 %
4		C#	8.3 %	-0.4 %
5	↑↑	Javascript	7.7 %	+0.4 %
6	↓	C++	6.9 %	-0.5 %
7	↓	C	6.9 %	-0.1 %
8		Objective-C	4.1 %	-0.6 %
9		R	3.5 %	+0.4 %
10		Swift	2.9 %	+0.0 %

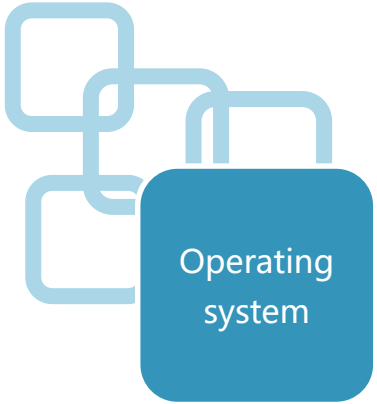
Popularity of programming language (PyPL)

Application of Python(1)

Python defines a WSGI standard application interface to coordinate communication between HTTP servers and Python-based Web applications

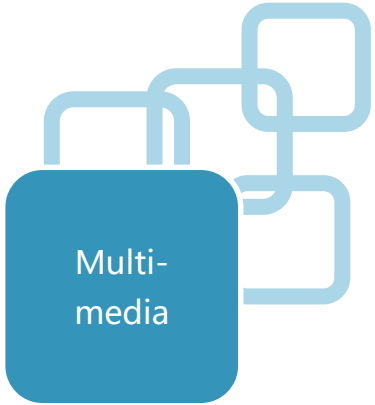
wxPython or PyQt can be used to develop cross-platform desktop software

Application of Python(2)



Most Linux distributions, as well as NetBSD, OpenBSD, and Mac OS X, have integrated Python, and the Python standard library includes multiple libraries that can call the functionalities of operating system.

It can be used in 3d scene production in computer games.



Application Examples of Python

Top Python programmers

Alex Martelli

Winner of 2002 Activators' Choice Award and 2006 Frank Willison award, developer of business intelligence software at Google

Daniel Greenfeld

Previously worked at NASA, currently in charge of the Cartwheel Web

Miguel Grinberg

He produces video software for Harmonic. C + + is the primary language he uses, but the test framework for automated unit written in Python seems more interesting to him.

Python mottos

The Zen of Python

>>> import this

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

by Tim Peters

Data processing Using Python

2

THE FIRST PYTHON PROGRAM

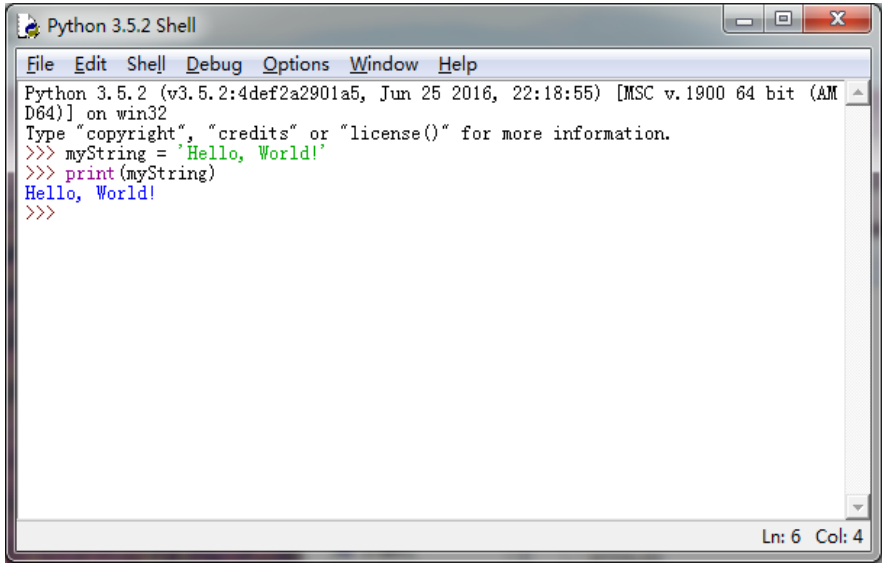
Classical Hello World

```
myString = 'Hello, World!'

print(myString)
```

How Python works (1)

Shell way



- Shell is an interactive interpreter
- When a line of command is input, the interpreter will interpret and run it to get the corresponding result.

How Python works (2)

Document way

- Create a file with extension name ***py*** in the IDE environment of Python.
- Run in the Shell using Python interpreter to get the result.

Classical Hello World

Source

```
>>> myString = 'Hello, World!'
>>> print(myString)
Hello, World!
>>> myString
'Hello, World!'
```

File

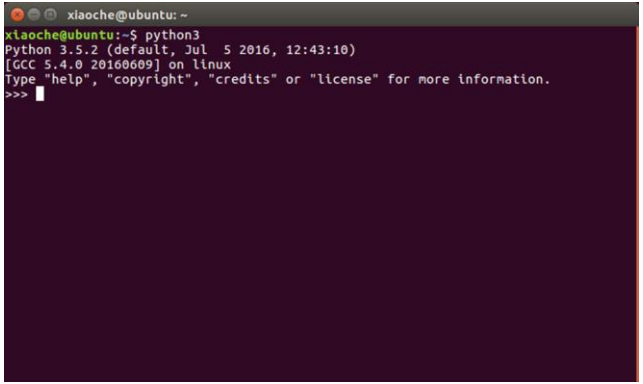
```
# Filename: helloworld.py
myString = 'Hello, World!'
print(myString)
```

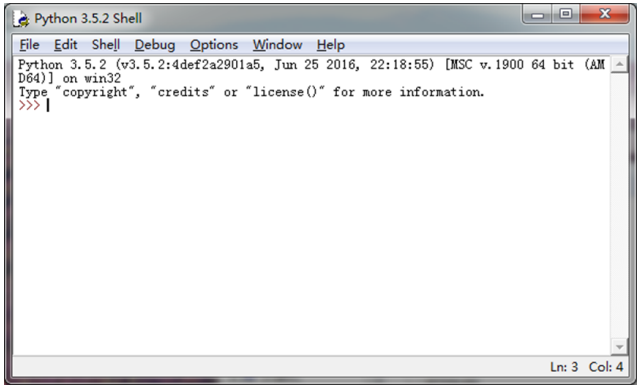
Python Integrated Development Environment (IDE)

Python IDE

- In Mac OS & Linux
  - $ python
  - $ python3
- Python built-in IDE
  - IDLE

  (https://www.python.org/ftp/python/3.5.2/python-3.5.2-amd64.exe)
- Other IDE
  - Ipython
  - PyCharm





Installation of the package(plug-in)

Installation of plug-ins

- Install the plug-in with the pip command (integrated in most Python IDEs and require no additional installation)

①Download get-pip.py (https://pip.pypa.io/en/latest/installing/)

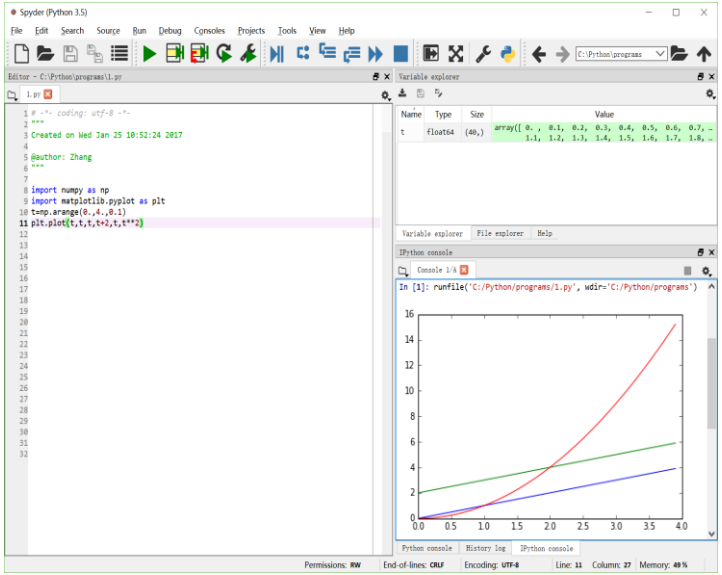
②Execute the following commands in sequence

> python get-pip.py

> pip install atx

Python development platform

- **Anaconda integrated development platform**
  - Download the installation package (https://www.continuum.io/downloads)
  - Install in Mac OS and Linux
    - **command line installer in MacOS:** bash Anaconda3-4.3.0-MacOSX-x86_64.sh
    - **graphical installer in MacOS:** download the graphical installer .pkg and follow the instrutions
    - **command line installer in Linux:** bash Anaconda3-4.3.0-Linux-x86_64.sh



Python output: *print* function

- Python uses the **print** function to output information
  - print(variables)
  - print(strings)

Source

```
>>> myString = 'Hello, World!'
>>> print(myString)
Hello, World!
```

Python input: the input () function

- The type returned by input () is a string type.

Source

```
>>> price = input('input the stock price of Apple: ')
input the stock price of Apple: 109
>>> price
'109'
>>> type(price)
<class 'str'>
>>> price = int(input('input the stock price of Apple: '))
>>> price = eval(input('input the stock price of Apple: '))
```

Python style (1)

Comment

Source

```
>>> # comment No.1
>>> print('Hello, World!')      # comment No.2
Hello, World!
```

Python style (2)

long sentence


```
>>> # long sentence
>>> if (signal == 'red') and\
      (car == 'moving'):
          car = 'stop'
      elif (signal == 'green') and\
      (car == 'stop'):
          car = 'moving'
```


```
>>> # long sentence
>>> if (signal == 'red') and (car == 'moving'):
          car = 'stop'
      elif (signal == 'green') and (car == 'stop'):
          car = 'moving'
```

Python style (2)

Long sentence

- There are two situations in which the line can be continued without the continuation markers:
  - Multiple lines can be written in parentheses, brackets, and curly braces
  - Strings included in triple quotes can also be written across lines.

Source

```
>>> # triple quotes
>>> print('''hi everybody,
welcome to python's MOOC course.
Here we can learn something about
python. Good lucky!''')
```

Python style (3)

Multiple statements in one line

Source

```
>>> x = 'Today' ; y = 'is' ; z = 'Thursday' ; print(x, y, z)
Today is  Thursday
```

Source

```
>>> x = 'Today'
>>> y = 'is'
>>> z = 'Thursday'
>>> print(x, y, z)
Today is  Thursday
```

Python style (4)

Indentation

01

Increasing indentation represents the beginning of a statement block.

02

Same indentation level means the same level of statement block in Python.

03

Reducing indentation represents the end of a statement block.

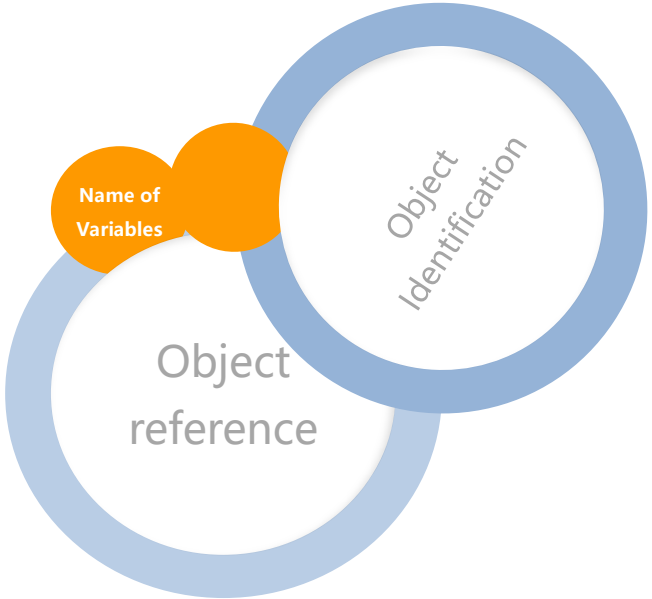
Source

```
>>> # Indentation
>>> if (signal == 'red') and (car == 'moving'):
        car = 'stop'
        signal = 'yellow'
    elif (signal == 'green') and (car == 'stop'):
        car = 'moving'
        signal = 'yellow'
```

Processing Data Using Python

3 PYTHON GRAMMAR FOUNDATION

Variables



Source

```
>>> # variable
>>> p = 3.14159
>>> myString = 'is a mathematic circular constant'
>>> print(p, myString)
3.14159 is a mathematic circular constant
```

Identifiers

- Identifiers are valid symbols in Python language that could be used as names of variables or other objects
  - The first character is a letter or an underline (_)
  - The rest can be letters, underlines, and numbers
  - Case sensitive (PI and pi are different identifiers)

Source

```
>>> # Identifier
>>> PI = 3.14159
>>> pi = 'one word'
>>> print(PI)
3.14159
>>> print(pi)
one word
```

Keyword

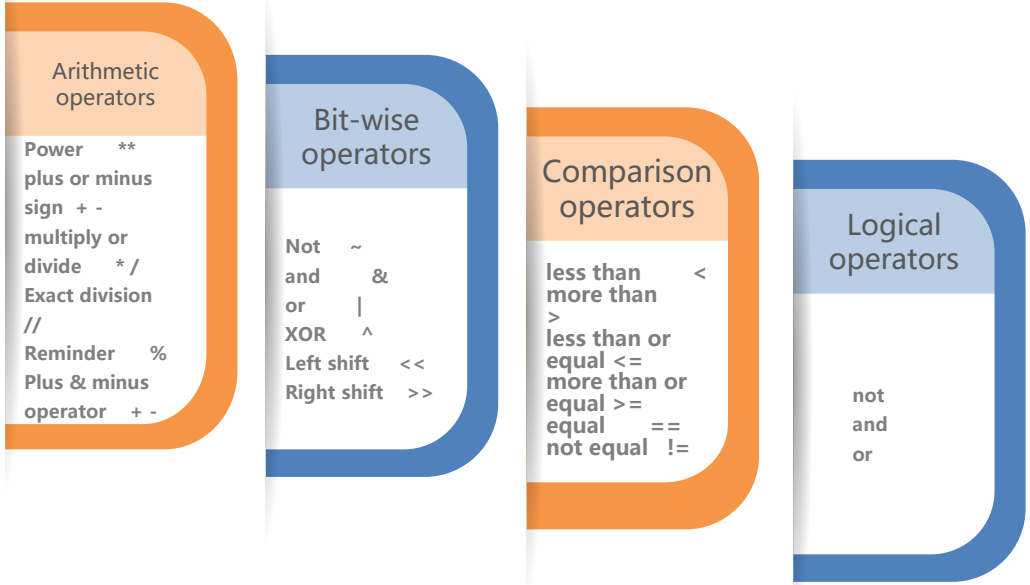
- Keywords are key components of Python language and cannot be used as identifier for other objects
  - Key word in a language is basically a fixed set of characters
  - Often appear with different color or fonts in IDE

```
>>> import keyword
>>> print(keyword.kwlist)
```

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

Expressions

- Expressions are combinations of various types of data and operators.



Expressions

- Operators have precedence order.
- The expression must have a result.

Source

```
>>> # expression
>>> PI = 3.14159
>>> r = 2
>>> c_circ = 2 * PI * r
>>> print("The circle's circum is", c_circ)
```

- 2*PI*r is an expression.
- The result is assigned to variable c_circ

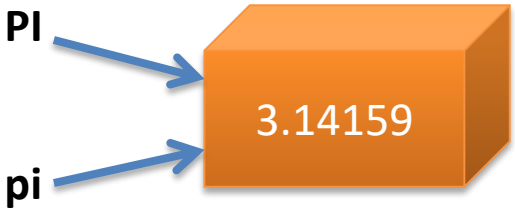
Assignment

- When variable is first assigned with a value, it gets both the type and the value.
  - Python is a dynamic, strongly-typed language
  - No explicit declaration. The type depends on the “value”.
  - Assignment is implemented in a "reference" way.

Source

```
>>> # Identifier
>>> PI = 3.14159
>>> pi = 'one word'
>>> print(PI)
3.14159
>>> print(pi)
one word
```

Assignment



Source

```
>>> # Identifier
>>> PI = 3.14159
>>> pi = PI
>>> print(PI)
3.14159
>>> print(pi)
3.14159
>>> p = 3
>>> q = 3
>>> p is q
True
```

Assignment -Augmented assignment

Augmented assignment operator

+= -= *= /= %= **= <<= >>= &= ^= |=

- **m %=5** equals to **m = m % 5**
- **m **=2** equals to **m = m ** 2**

Source

```
>>> # Augmented assignment
>>> m = 18
>>> m %= 5
>>> m
3
>>> m **= 2
>>> m
9
```

Assignment -Chained assignment

Source

```
>>> # Chained assignment
>>> PI = pi = 3.14159
>>> PI
3.14159
>>> pi
3.14159
```

Source

```
>>> # Chained assignment
>>> PI = 3.14159
>>> pi = PI = PI * 2
>>> pi
6.28318
```

Assignment- multiple assignments

- The forms of tuples appear in both sides of the equal sign.

Source

```
>>> # assignment
>>> x = 1
>>> y = 2
>>> x, y
(1, 2)
>>> x, y = y, x
>>> x, y
(2, 1)
```

Source

```
>>> # assignment
>>> temp = 3.14159, 3
>>> PI, r = temp
>>> PI
3.14159
>>> r
3
>>> (PI, r) = (3.14159, 3) # same as no  round brackets
```

Tuple packing
Sequence unpacking

Statement

- A line of logical codes that completely performs a task
  - The assignment statement performs the assignment operation.
  - The print () function calls statements and completes the output task.

Source

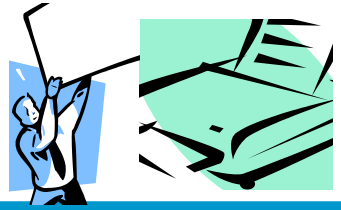
```
>>> # statement
>>> PI = 3.14159
>>> r = 2
>>> c_circ = 2 * PI * r
>>> print("The circle's circum is", c_circ)
```

Statements and expressions

Statements

Complete a task

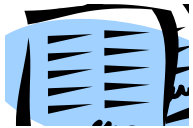
e.g. printing a document



Expressions

A specific component of the task

e.g. the specific content of the document



Processing Data Using Python

4 DATA TYPE IN PYTHON

Data type

- There must be clear data types for program to assign accurate storage sizes to constants and variables so as to perform precise or efficient operations.

1	0	0	1	0	0	1	1
1	0	0	1	0	0	1	1

+

0	0	0	0	1	0	0	0

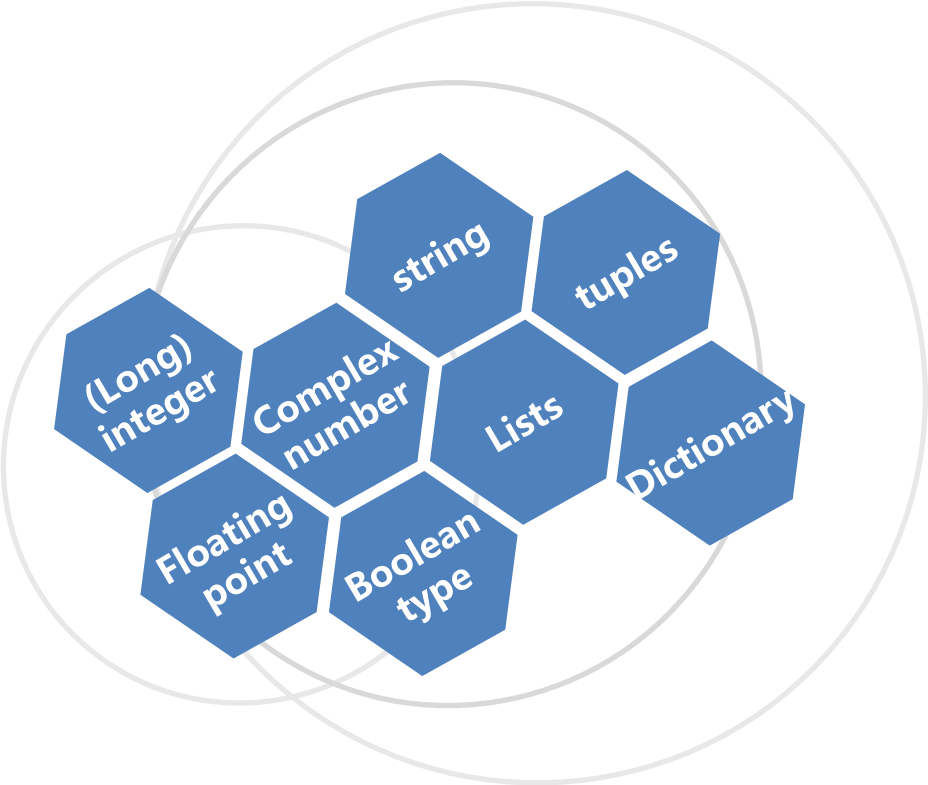
1	0	0	1	0	0	1	1

+

0	0	0	0	1	0	0	0

1	0	0	1	1	0	1	1

Python Standard data type



Integer

- The integer and long integer type are not strictly distinguished
- In Python 2 , integer value affixed with L is interpreted as long integer

Source

```
>>> # integer
>>> type(3)
<class 'int'>
```

Boolean type

- Subtype of integer
- Only two values: True and False
- In essence, they are stored as integer 1 and 0.

Source

```
>>> # boolean
>>> x = True
>>> int(x)
1
>>> y = False
>>> int(y)
0
```

Floating point type

- real number in mathematics
- can be expressed in the way of scientific notation

Source

```
>>> # float
>>> 3.22
3.22
>>> 9.8e3
9800.0
>>> -4.78e-2
-0.0478
>>> type(-4.78e-2)
<class 'float'>
```

Complex number

- j=$\sqrt{-1}$, then j is imaginary
- Real part+ imaginary part= complex number
- The imaginary part must have a j

Source

```
>>> # complex
>>> 2.4+5.6j
(2.4+5.6j)
>>> type(2.4+5.6j)
<class 'complex'>
>>> 3j
3j
>>> type(3j)
<class 'complex'>
>>> 5+0j
(5+0j)
>>> type(5+0j)
<class 'complex'>
```

Complex number

- The complex number can be separated into real part and imaginary part
  - complex.real
  - complex.imag
- Conjugate of complex Numbers
  - complex.conjugate()

Source

```
>>> # complex
>>> x = 2.4+5.6j
>>> x.imag
5.6
>>> x.real
2.4
>>> x.conjugate()
(2.4-5.6j)
```

Sequence types

01 Strings

Strings are contents inside single quote, double quotes, or triple quotes, which are immutable

02 List

A strong, mutable type, defined in the square brackets [].

03 Tuple

Similar to list，immutable type，defined in the parenthesis ()

Representation of String

- Single quotes
- Double quotes
- Triple quotes

Source

```
>>> myString = 'Hello World!'
>>> print(myString)
Hello World!
>>> myString = "Hello World!"
>>> print(myString)
Hello World!
>>> myString = '''Hello World!'''
>>> print(myString)
Hello World!
```

Mapping type-dictionary

- Defined by the curly braces {}
- Similar to the key-value pairs in hash table

Source

```
>>> # dictionary
>>> d ={'sine':'sin','cosine':'cos','PI':3.14159}
>>> d['sine']
'sin'
```

Data Processing Using Python

5 BASIC OPERATIONS IN PYTHON

Arithmetic operations

- The precedence of arithmetic operators
  - Power * *, positive and negative sign + -, multiply & divide by * /, exact division / /, remainder %, add and subtract + -

Source

```
>>> # arithmetic
>>> pi = 3.14159
>>> r = 3
>>> circum = 2 * pi * r
>>> x = 1
>>> y = 2
>>> z = 3
>>> result1 = x + 3/y -z % 2
>>> result2 = (x + y**z*4)//5
>>> print(circum, result1, result2)
18.84954 1.5 6
```

Comparison operations

- Numerical comparison: by value
- String comparison: the value of ASCII code

Source

```
>>> # compare
>>> 3 < 4 < 7   # same as ( 3 < 4 ) and ( 4 < 7 )
True
>>> 4 > 3 == 3  # same as ( 4 > 3 ) and ( 3 == 3 )
True
>>> 4 < 3 < 5 != 2 < 7
False
```

Source

```
>>> # compare
>>> 2 == 2
True
>>> 2.46 <= 8.33
True
>>> 'abc' == 'xyz'
False
>>> 'abc' > 'xyz'
False
>>> 'abc' < 'xyz'
True
```

Logical operations

- Logical operator precedence:
  - Not, and, or

Source

```
>>> # logical
>>> x, y = 3.1415926536, -1024
>>> x < 5.0
True
>>> not (x < 5.0)
False
>>> (x < 5.0) or (y > 2.718281828)
True
>>> (x < 5.0) and (y > 2.718281828)
False
>>> not (x is y)
True
>>> 3 < 4 < 7    # same as "( 3 < 4 ) and ( 4 < 7 )"
True
```

Character operator

- Raw string operator (r/ R) :
  - For places where you don't want the escape character to work
- All strings are Unicode strings:
  - In Python 2.x, need to be converted to a Unicode string

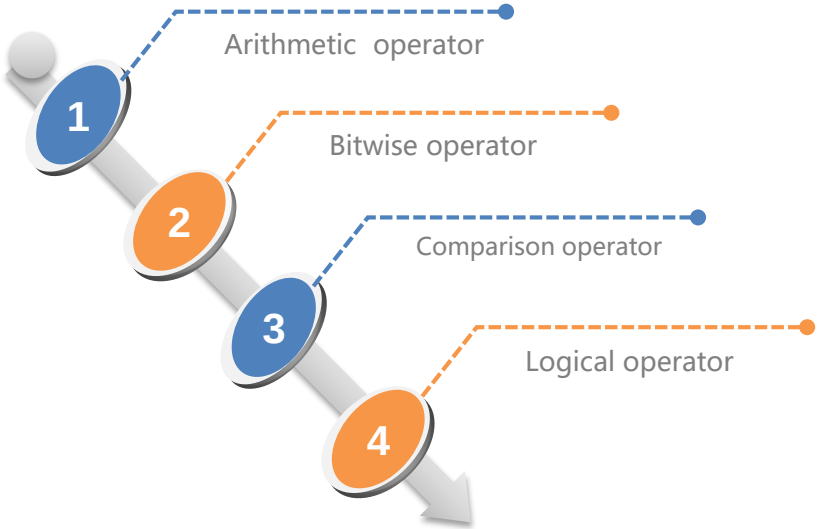
Source

```
>>> # u in Python 2.x
>>> print u'Hello\nWorld'
hello
World
```

Source

```
>>> # r
>>> f = open('c:\python\test.py','w')
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    f = open('c:\python\test.py','w')
IOError: [Errno 22] invalid mode ('w') or filename: 'c:\\python\test.py'
>>> f = open(r'c:\python\test.py','w')
>>> f = open('c:\\python\\test.py','w')
```

Mixed operation



Source

```
>>> # mix
>>> 3 < 2 and 2 < 1 or 5 > 4
True
>>> x + 3/y -z % 2 > 2
False
>>> 3-2 << 1
2
>>> 3-2 << 1 < 3
True
```

Data Processing Using python

6 FUNCTIONS, MODULES AND PACKAGES OF PYTHON

Functions(1)

- A function can be regarded as a mathematic function
- A piece of code that completes a specific task
  - Absolute function abs(x)
  - Type function type(x)
  - Round-off function round(x)

Functions(2)

- Built-in functions
  - **str**() and **type**() are applicable to all standard types

Numerical built-in functions

abs()	bool()	oct()
round()	int()	hex()
divmod()	ord()	pow()
float()	chr()	complex()

Useful functions

dir()	input()
help()	open()
len()	range()

Built-in functions

```
>>> dir(__builtins__)
```


		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Functions(3)

Source

```
>>> # round-off int
>>> int(35.4)
35
>>> int(35.5)
35
>>> int(35.8)
35
>>> type(int(35.8))
<class 'int'>
```

Source

```
>>> # ord
>>> ord('3')
51
>>> ord('a')
97
>>> ord('\n')
10
>>> type(ord('A'))
<class 'int'>
```

Module(1)

- How to use non-built-in functions?

Source

```
>>> # round-off floor
>>> floor(5.4)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    floor(5.4)
NameError: name 'floor' is not defined
```

Source

```
>>> # round-off floor
>>> from math import *
>>> floor(-35.4)
-36
>>> floor(-35.5)
-36
>>> floor(-35.8)
-36
```

Module(2)

- A complete Python file is a **module**
  - File: physical organization math.py
  - Module: logical organization math
- Python usually uses "**import module**" to apply functions, types in a given module to other code blocks.
  - The value of math.pi can be used directly without defining by yourself.

Source

```
>>> # module
>>> import math
>>> math.pi
3.141592653589793
```

Module(3)

- Import multiple modules
- To import a specified module element to current module is to import the specified name to the current scope

```
>>>import ModuleName
>>>import ModuleName1, ModuleName2, …
>>>from Module1 import ModuleElement
```

package

- A hierarchical file directory structure
- Defines the execution environment for Python application consisting of modules and sub-packages

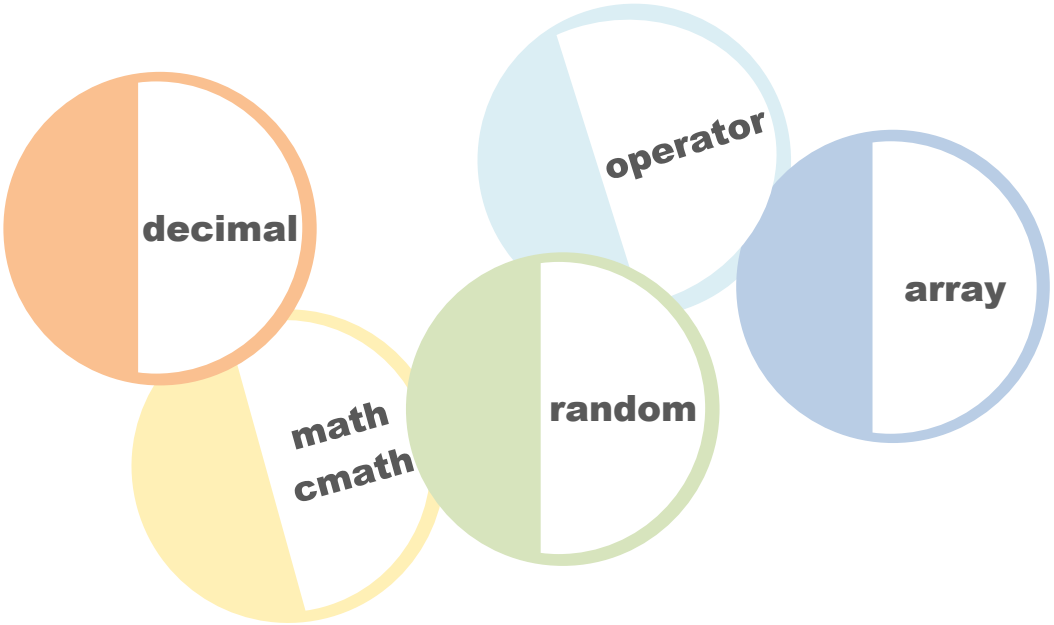
```
>>> import AAA.CCC.c1
>>> AAA.CCC.c1.func1(123)
```

```
>>> from AAA.CCC.c1 import func1
>>> func1(123)
```

```
AAA/
    __init__.py
    bbb.py
    CCC/
        __init__.py
        c1.py
        c2.py
    DDD/
        __init__.py
        d1.py
    EEE/
        …
```

library

- A library is a collection of modules with related functions
- One feature of Python is that it has a powerful standard library, as well as third-party libraries and custom modules



Numeric-related standard libraries