# Python Programming Basics

www.huawei.com



HUAWEI

# Foreword

- Python is an easy-to-learn and functional programming language.
- Python has an effective advanced data structure and enables simple and rapid object-oriented programming.
- With its beautiful syntax, dynamic types and good interpretability, Python has become an ideal language in terms of scripting and developing apps on most platforms.

# Objectives

- After completing this course, you will be able to:
  - Understand the compilation environment and installation process of Python.
  - Master the data structure, data types, conditional statements, loop statements, functions, and modules of Python.
  - Apply the Python programming basics to actual scenarios.

# Contents

## History of Python

- Python is one of the achievements of free software.
- Python is purely free software and both its source code and interpreter comply with the GNU General Public License (GPL) protocol.

| | |
|---|---|
| Founder | Guido van Rossum |
| When and Where | Created in Amsterdam during Christmas in 1989. |
| Meaning of Name | A big fan of Monty Python's Flying Circus |
| Origin | Influenced by Modula-3, Python is a descendant of ABC that would appeal to Unix/C hackers. |





In December 1989, Guido Van Rossum had been looking for a '"hobby' programming project that would keep him occupied during the week around Christmas" as his office was closed when he decided to write an interpreter for a "new scripting language he had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers". He attributes choosing the name "Python" to "being in a slightly irreverent mood and being a big fan of Monty Python's Flying Circus".

Guido believes that the ABC programming language, specially designed for programmers, was elegant and powerful, but it did not succeed because it is not open. Guido decided to avoid this issue. Even so, Python is well integrated with other programming languages like C, C++, and Java. In this same time, he wanted to implement the flashing ideas in ABC. Then Guido created Python. Actually, Python was firstly run on a MAC computer. In a conclusion, Python, under the influence of Modula-3 (another beautiful and powerful language, designed by a small community), was a descendant of ABC that would appeal to Unix/C hackers.

According to The Free Software Foundation Europe (FSFE), the freedoms to use, copy, modify and redistribute software, as described in The Free

Software Definition, are necessary for equal participation in the Information Age. Software is tradable, which is the most important essence of free software.

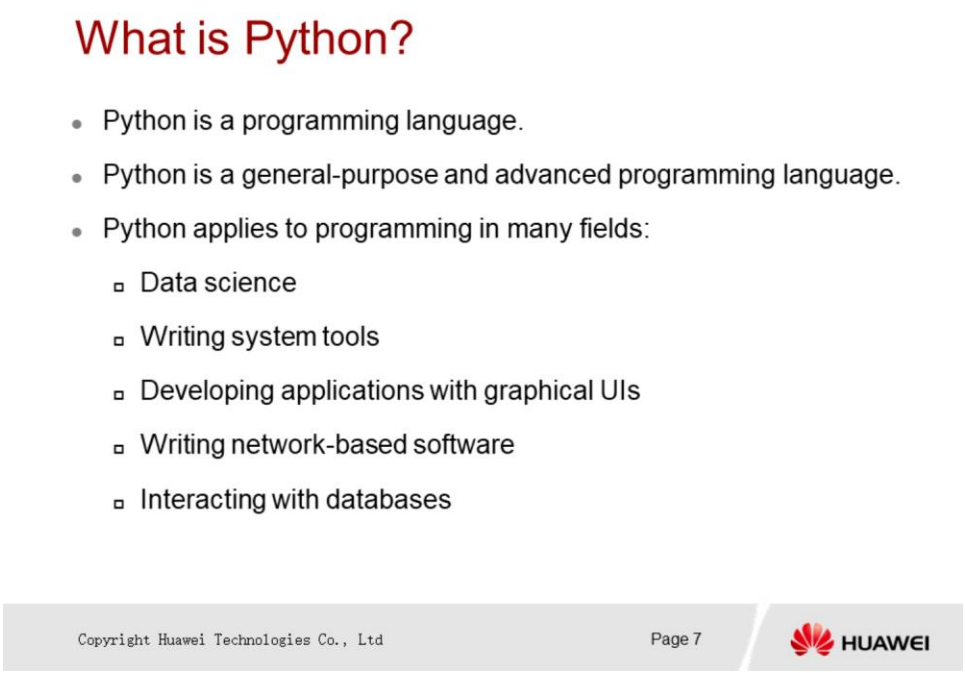
## Origin of Python

- Guido van Rossum:
  - Master of Mathematics
  - Master of Computer Science
- Philosophy of Python:
  - Python is engineering but not art.
  - There should be one, and preferably only one, obvious way to do it.
  - Simple is better than complex, and explicit is better than implicit.

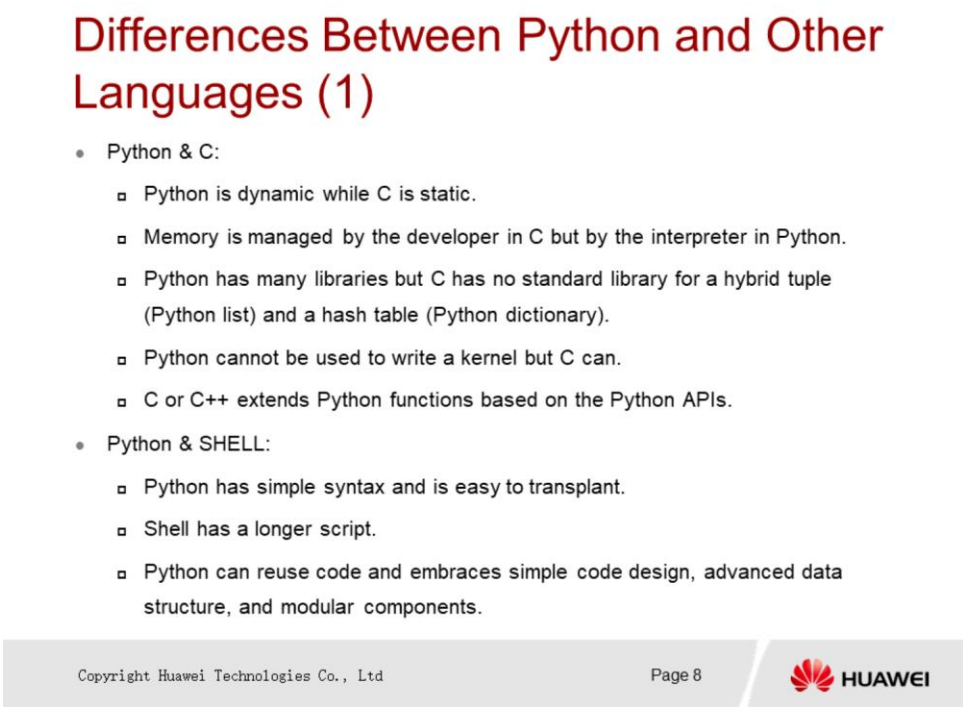




- Guido Van Rossum was born and raised in the Netherlands, where he received a master's degree in mathematics and computer science from the University of Amsterdam in 1982. In the same year, he joined Centrum Wiskunde & Informatica (CWI) as a researcher. In 1989, he created the Python programming language while working at CWI. In 1991, the first public issue of Python was released.




- Python has rich and powerful databases. It glues together various software components made by other languages, especially C and C++. That is why it is called a glue language. In a typical application, a software prototype (sometimes even the final UIs of software) is quickly created using Python, and the components with special requirements are created using other more suitable languages. For example, the graphics rendering module in a 3D game, which requires high performance, can be rewritten by C or C++ and then packaged as an extension class that can be called by Python. It should be noted that extension classes may not span multiple platforms.
- Data science includes data analysis, data mining, machine learning, natural language processing, and artificial intelligence.




- A dynamically compiled executable file has to be attached with a dynamic link library so that the commands in the dynamic link libraries can be called during the file execution.
  - Advantages: It squeezes size of the executable file and also speeds up compilation while saving system resources.
  - Disadvantages: A large link library has to be attached even if one or two commands from the library are used for a simple program. If the matched run-time library is not installed on the computer, the dynamically compiled executable file may not be able to executed.
- In static compilation, the compiler extracts a part of the dynamic link library (.so) and linked it to the executable file so that execution of the file does not depend on the dynamic link library.
  - Therefore, static compilation reversely turns the advantages and disadvantages of dynamic compilation.

# Differences Between Python and Other Languages (2)

- Python & Java:
  - Python is dynamic while Java is static.
  - Python supports object-oriented and function-based programming while Java supports object-oriented programming only.
  - Python is simpler than Java, and typically applies to quick prototyping.
  - Python and Java enable multiple programmers to develop a large project together step by step.

# Development Environment of Python

- VIM: mainly for Linux
- IDLE: integrated development environment
- Sublime Text: a lightweight editing tool
- Eclipse: chargeable
- Eric4: powerful, based on PyQT4
- Boa: similar to IDE (wxPython) of Delphi
- WingIDE: shared software
- Other editors: notepad++, editplus…

# Advantages of Python (1)

- Simple: Python is an ideal programming language of simplicity.
- Easy-to-learn: few key words, simple structure, and explicit syntax
- Open-source: Python is one of Free/Libre and Open-Source Software (FLOSS).
- Interpretability: Python program does not need to be compiled into binary code, and can be directly run from source code.
- Transplantability: Python has been transplanted to multiple platforms thanks to its open-source nature.
  - The platforms include Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS,Psion, Acom RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE, and even Symbian and Android developed by Google based on Linux.
- Scalability: If you want to speed up running of some key code or keep some algorithms not open, you can write some programs using C or C++ and use them in Python.





- Coding-type languages: C and C++
- Source code > compilation > target code > execution > output

# Advantages of Python (2)

- Advanced language: When you write a program in Python, you do not need to consider the underlying-layer details, like how to manage the memory that your program uses.
- Embedded: Python can be embedded in a C or C++ program to provide scripting to program users.
- Object-oriented: Python supports both process-oriented programming and object-oriented programming. In a process-oriented language, a program is built from a process or a function that is merely reusable code. In an object-oriented language, programs are built from the combination of data and functionality.
- Rich libraries: The Python standard library is huge. It can help you with all kinds of work, including regular expressions, document generation, unit tests, threads, databases, Web browsers, CGI, FTP, e-mail, XML-RPC, HTML, WAV files, password systems, GUI, TK, and other system-related operations.

## Features of Python Statements

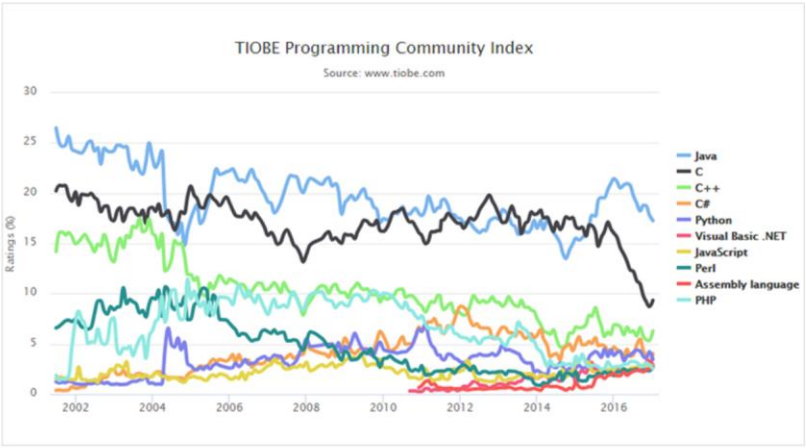
- Dynamic: Objects (like attributes and methods) can be changed during execution.
- Python uses indentation instead of a pair of curly braces {} to divide a block of statements.
- Multiple statements on one line are separated by ";".
- The symbol used for commenting out a line is #, and doc string ("... ") is used to comment out multiple lines.
- Variables do not need type definitions.
- Functional Programming (FP) is available.





One idea in functional programming (FP) is that calling a function should have no "side effects". That is, it cannot depend on the state or modify the state. One advantage of doing this is that each function is easy to understand and readable.

# Popularity Ranking








- The TIOBE Programming Community rankings are an indicator of the trends in programming languages that are updated every month, based on the number of experienced programmers, courses and third-party vendors on the Internet. Rankings are calculated using well-known search engines such as Google, MSN, Yahoo!, Wikipedia, YouTube, and Baidu.
- Note that this ranking only reflects the popularity of a programming language and does not explain how good a programming language is, or how much code is written in a single language.

# Differences Between Python 2 and Python 3 (1)

- Python 3 cannot be backwards compatible with Python 2, which requires people to decide which version of the language is to be used.
- Many libraries are only for Python 2, but the development team behind Python 3 has reaffirmed the end of support for Python 2, prompting more libraries to be ported to Python 3.
- Judging from the number of Python packages that are supported by Python 3, Python 3 has become increasingly popular.





- For differences between Python2 and Python3, access https://www.cnblogs.com/codingmylife/archive/2010/06/06/1752807.html.

## Differences Between Python 2 and Python 3 (2)

- print function
- Unicode
- Integer division
- Exceptions
- Xrange
- Data type
- Inequality operator





- Python 2 uses ASCII strings. Unicode strings are more versatile than ASCII strings, and are not of the byte type. Python 3 uses Unicode (utf-8) strings and a byte class (byte and bytearrays).
- In Python 2.x, integer division is similar to most of the languages that we are familiar with, such as Java and C. The result of integer division is an integer, and the fractional part of the totally ignored. For floating point division, the decimal part will be retained to get the result of a floating-point number. In Python 3.x, integer division no longer does so, which means that the result will also be floating-point numbers for dividing the integers.
- In Python 3, handling exceptions has also changed slightly, and we now use as as a keyword in Python 3. The syntax for catching exceptions is changed from except exc, var to except exc as Var. In the Python 2.x era, exceptions in the code not only express program errors, but also often do something as a common control structure should do. In Python 3.x, the designer make exceptions single-purpose and use them to capture Statements only in case of errors.
- In Python 3, range () is implemented as xrange () so that a dedicated xrange () function no longer exists (xrange () returns a naming exception in

Python 3).

- Python 3.x removes the long type and now has only one integer--int, but it behaves like long type in Python 2.x. Python 3.x has a new bytes type.

# Installing Python (1)

- For Linux users:
  - Download the Python package and install it.

```
$tar -zxf python3.6.4.tar.gz
$cd Python3.6.4
$./configure
$make && make install
```

  - Create soft connections.

```
$mv /usr/bin/python  /usr/bin/python.bak
$ln -s /usr/local/bin/python3.6.4
/usr/bin/python
```

  - Verify installation.

```
 $python -V
```

# Installing Python (2)

- For Windows users:
  - Download the official Python setup program.
  - Select the latest Python Windows installer to download the .exe installation file.
  - Double-click the setup program, Python-3.x.exe.
  - Add environment variables: Choose **My Computer** > **Properties** > **Advanced** > **Environment Variables**, and enter your Python installation location in path.
  - Verify installation: **Start** > **Program** > **Python 3.x** > **Start Python command line**, and then enter: print("Hello World"). If the output is "Hello World", it indicates that the installation was successful.

# Starting Python

- Start Python on Linux:


- Start Python on Windows:





- Linux: ctrl+d to exit Python
- Windows: Ctrl+z to exit Python

# Executing a Python Program

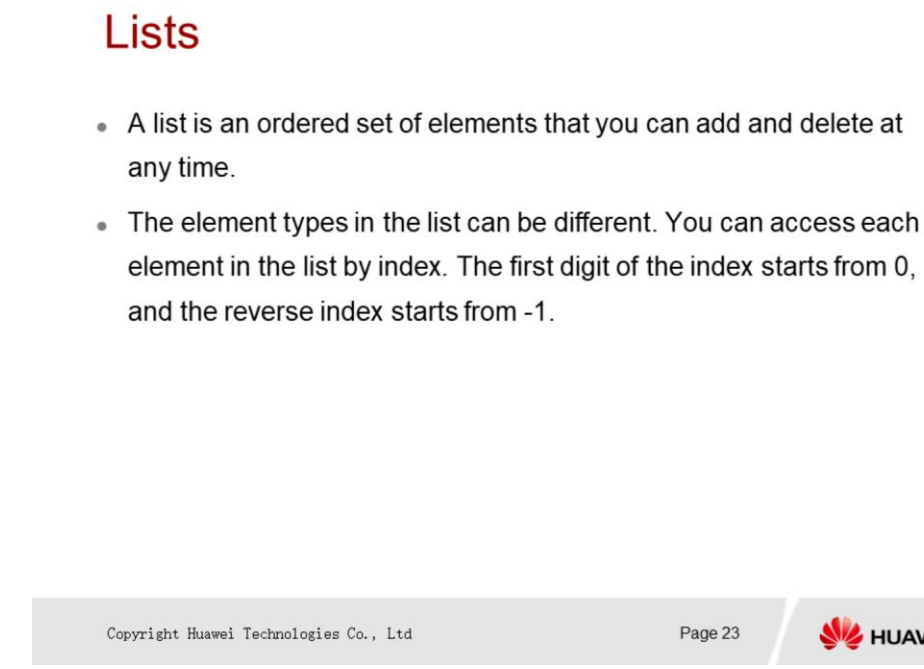
- Using command lines:
  - On Linux:
    - Enter a Python command in the Linux command line.
  - On Windows:
    - Enter a Python command on the DOS prompt.
- Using scripts:
  - Store Python Statements in a script file and execute it in the command line.

```
Input: python hello.py
Output: hello world !
```

# Vim

- Vim is a lightweight IDE.
- If you do not install too many plug-ins or plug-in performance is good, using VIM for development has a low requirement on hardware.
- Vim can achieve a consistent programming experience on local and remote servers.
- Vim has an editing speed of "what you think is what you have".

# Contents




- The list is the most frequently used data type in Python. The list can implement the data structures of most collection classes. It supports characters, numbers, and strings that can even contain lists (so-called nesting). The list is identified by [] and is the most common composite data type in Python.
- When you create a list in Python, the interpreter creates a data structure in memory that is similar to an array (but not an array) to store it. The numbering in the list starts with 0, then 1, and so on.

## Common Operations on Lists

- Access
- Update (append and insert)
- Delete elements (del)
- Operator on list script (+/*)
- List interception





- The list is the most frequently used data type in Python. The list can implement the data structures of most collection classes. It supports characters, numbers, and strings that can even contain lists (so-called nesting). The list is identified by [] and is the most common composite data type in Python.
- When you create a list in Python, the interpreter creates a data structure in memory that is similar to an array (but not an array) to store it. The numbering in the list starts with 0, then 1, and so on.

# Functions of Python Lists

- cmp(list1, list2),
- len(list)
- max(list)
- min(list)
- list(seq)





- cmp(list1, list2): compares elements between two lists
- len(list): number of list elements
- max(list): returns the maximum value of list elements
- min(list): returns the minimum value of list elements
- list(seq): converts tuple into list


- list.append(obj), appends a new object at the end of a list
- list.count(obj), counts the repetitions of a given element in a list
- list.extend(seq), adds multiple values of another sequence at the end of a list (using a new list to extend the original list)
- list.index(obj), finds the indexing location with the first match with a value in the list
- list.insert(index, obj), inserts an object in the list
- list.pop(obj=list[-1]), removes an element (last one by default) from the list and returns value of the element
- list.remove(obj), removes the first match with a given value from the list
- list.reverse(), reverse elements in a list
- list.sort([func]), sorting the original list

## Tuples

- Tuple is expressed using ().
- Like a list, a tuple cannot be modified once initialized, and the elements need to be identified when a tuple is defined.
- A tuple does not have the append () or insert () method, nor can it be assigned to another element. It has the same fetching methods as a list.
- Because a tuple is unchangeable, the code is more secure. Therefore, if possible, use a tuple instead of a list.
- A tuple is simple to create. You only need to add elements to parentheses and separate them with commas.





- In Python, tuples are similar to lists, except that the elements of a tuple cannot be modified, and tuples use parentheses, while lists use square brackets. A tuple is easy to create, that is, by adding elements in parentheses and separating them with commas.

## Common Operations on Tuples

- Access
- Modify (tuple calculation)
- Delete (del tuple)
- Tuple operators (+,*)
- Tuple index and interception
- No-close separator




- As with strings, + and * can be used for tuple calculation, which means that tuples can be combined or duplicated to create new tuples after the calculation.
- No close separator: Any unsigned objects, separated by commas, default to tuples,

# Embedded Functions of Tuples

- cmp(tuple1, tuple2)
- len(tuple)
- max(tuple)
- min(tuple)
- tuple(seq)




- cmp(tuple1, tuple2), compares elements between two tuples
- len(tuple), counts tuple elements
- max(tuple), returns the maximum value of elements in a tuple
- min(tuple), returns the minimum value of elements in a tuple
- tuple(seq), converts a list into a tuple

# Contents

## Definition of a String

- In Python, a string is a sequence of 0 or more characters, and a string is one of several sequences built in Python.
- In Python, strings are unchangeable, and are similar to string constants in the C and C++ languages.
- Python strings may be expressed using single quotes, double quotes and triple quotes, as well as escape characters, raw strings, and so on.
  - name='JohnSmith‘
  - name="Alice“
  - name="""Bob"""





- In the actual use of Python, you have to define an array, the contents of which are the strings of class names.
- When the corresponding class is called to generate an instance, the string has to be converted to a class, which then will be instantiated.

# String Formatting (1)

- Python supports the output of formatted strings. Although a complex expression may be used, the most basic use is to insert a value into a string with the string format character %s.
- String formatting in Python is accomplished by the string formatting operator (%), and its string conversion type table and its formatting operator auxiliary instructions are shown in the following tables.

```
Input: print("My name is %s and age is %d !" %('AI', 63))
Output: My name is AI and age is 63 !
```

# String Formatting (2)

- String format conversion types:

| Format | Description |
|---|---|
| %c | Character and its ASCII code |
| %s | String |
| %d | Signed integer (decimal) |
| %u | Unsigned integer (decimal) |
| %o | Unsigned integer (octal) |
| %x | Unsigned integer (hexadecimal) |
| %X | Unsigned integer (hexadecimal upper-case letters) |
| %e | Floating number (scientific counting method) |
| %E | Floating number (scientific counting method, E replacing e) |
| %f | Floating number (decimal point sign) |
| %g | Floating number (%e or %f, depending on a value) |
| %G | Floating number (similar to %g) |
| %p | Pointer (print memory address of a value using hexadecimal) |

# String Formatting (2)

- Auxiliary commands for formatting operators:

| Symbol | Function |
|---|---|
| * | Defines width or precision of the decimal point. |
| - | Aligns to the left. |
| + | Displays + before a positive value. |
| <sp> | Displays space before a positive value. |
| # | Displays 0 before an octal number or 0x or 0X before a hexadecimal value (depending on whether x or X is used). |
| 0 | Adds 0 instead of default space before numbers. |
| % | %% outputs a single %. |
| (var) | Maps variables (dictionary arguments). |
| m.n | m means the minimum width and n means the number of digits after the decimal point. |

## String Operators

- Python does not have dedicated Char type and one character is a string with a length of 1. Python strings are not changeable and do not end with '\0'. A string is the sequence of a character and is stored in memory as follows:

| | P | y | t | h | o | n |
|---|---|---|---|---|---|---|
| Superscript | 0 | 1 | 2 | 3 | 4 | 5 |
| Subscript | -6 | -5 | -4 | -3 | -2 | -1 |

- In Python, subscripts start from 0. -1 indicates the subscript of the last value in the sequence, 1 indicates the subscript of the second character, -2 indicates the subscript of the second to last character, and so on.





In a sequence, each element is assigned an ordinal number, which indicates the position of the element, and is also called index.

Ordinal numbers start forward from 0 to 1, 2, 3, … or backward from -1 to -2, -3…

Python contains 6 types of built-in sequences: lists, tuples, strings, Unicode strings, buffer objects, and xrange objects.

The main difference between a list and a tuple is that the list can be modified but the tuple cannot.

The individual elements in a list are separated by commas and are written in square brackets.

A sequence can also contain other sequences.

Sequences (lists and tuples) and dictionaries (mappings) are two main types of containers.

Each element in a sequence has its own ordinal number.

Each element in a dictionary has a name (also known as a key).

# String Methods

- You need to know the most useful built-in function of Python, dir, before learning more about functions. You can use the dir function to check the attributes and methods of Python strings.

```
>>> dir('')
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__',
'__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__str__', 'capitalize', 'center',
'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit',
'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
>>>
```

## String Modules

- In addition to the above mentioned attributes and methods, Python strings also have a string module. You can also use the dir function to check attributes and methods of the string module.

```
>>> import string
>>> dir(string)
['Template', '_TemplateMetaclass', '__builtins__', '__doc__', '__file__', '__name__', '_float',
'_idmap', '_idmapL', '_int', '_long', '_multimap', '_re', 'ascii_letters', 'ascii_lowercase',
'ascii_uppercase', 'atof', 'atof_error', 'atoi', 'atoi_error', 'atol', 'atol_error', 'capitalize', 'capwords',
'center', 'count', 'digits', 'expandtabs', 'find', 'hexdigits', 'index', 'index_error', 'join', 'joinfields',
'letters', 'ljust', 'lower', 'lowercase', 'lstrip', 'maketrans', 'octdigits', 'printable', 'punctuation',
replace', 'rfind', 'rindex', 'rjust', 'rsplit', 'rstrip', 'split', 'splitfields', 'strip', 'swapcase', 'translate', 'upp
er', 'uppercase', 'whitespace', 'zfill']
>>>
```





- From the output above, you can see that most of the methods provided by the string module are the same as those described earlier, but differ in how they are used.
- Note: You have to import string methods first to use them.

# Contents

## Dictionaries

- A dictionary is another variable container model and can store any type of object.
- Each key value of the dictionary is separated with a colon ":" key value pairs are separated by a comma "," and the entire dictionary is included in the curly braces "{}".
- The key is generally unique, and the type of the key is unchangeable. If the key repeats, the last key-value pair replaces the previous one. Key values do not need to be unique, and can take any data type.
- A dictionary has the following format:
  - d = {key1 : value1, key2 : value2 }





- Dictionaries are the only mapping types in Python.
- The hash value (key) and the pointed object (value) in the mapped type object are of one-to-many relationships and are often considered to be variable hash tables.
- A dictionary object is variable and is a container type that can store any number of Python objects. It can also include other types of containers.

# Python Dictionary Operations

- Access
- Modify
- Delete

# Built-in Functions of Dictionaries

| Function | Meaning |
| --- | --- |
| cmp(dict1,dict2) | Compares elements between dictionaries. |
| len(dict) | Counts elements in a dictionary, or total number of keys. |
| str(dict) | Outputs printable string expressions of a dictionary. |
| type(variable) | Returns the types of input variables, and returns dictionary types if the variables are dictionaries. |





- cmp(dict1, dict2): Compares elements between dictionaries.
- len(dict): Counts elements in a dictionary, or total number of keys.
- str(dict): Outputs printable string expressions of a dictionary.
- type(variable): Returns the types of input variables, and returns dictionary types if the variables are dictionaries.

# Built-in Methods of Dictionaries

- Built-in methods:

| Method | Description |
|---|---|
| has_key(x) | Returns the value if the dictionary has a key x. |
| keys() | Returns a key list of the dictionary. |
| values() | Returns a value list of the dictionary. |
| items() | Returns a tuple list. Each tuple includes a key and value pair of the dictionary. |
| clear() | Clears all items in the dictionary. |
| copy() | Returns a copy of the higher-layer structure of the dictionary. It does not copy the embedded structure but only reference to the structure. |
| update(x) | Updates the dictionary with key values in dictionary x. |
| get(x[,y]) | Returns key x. It returns none if key x is not found and returns y if y is provided and x is not found. |
| pop() | Deletes the value of a given key in the dictionary. It returns the deleted value and key value has to be provided; otherwise, it returns the default value. |
| popitem() | Randomly returns and deletes a key and value pair in the dictionary. |

# Contents

## if Statements

- Python supports three control structures: if, for, and while, but it does not support switch statements in the C language.
- In Python programming, if statements are used to control execution of control programs, and the basic form is:

```
if Judging condition 1:
    Statement 1...
elif Judging condition 2:
    Statement 2...
elif Judging condition 3:
    Statement 3...
else:
    Statement 4...
```





- The Python programming language specifies any non-0 and non-null values as true, and 0 and null as FALSE.
- In Python programming, if statements are used for execution of control program. The judgment condition of an if statement can be expressed as ">" (greater than), “<“ (less than), = (equal), >= (equal and greater), and <= (less and equal).
- Since Python does not support switch statements, multiple conditional judgments can only be implemented using elif. If multiple conditions need to be judged at the same time, and judgement is successful on either condition, or can be used; If multiple conditions need to be judged at the same time, and judgement is successful on both conditions, and can be used.

## while Statements

- The while statement in the Python language is used to execute a loop program that, under certain conditions, loops through a program to handle the same tasks that need to be repeated.
- When the condition of a while statement is never a Boolean false, the loop will never end, forming an infinite loop, also known as a dead loop. You can use the break statement in a loop to force a dead loop to end.
- How to use a while statement:

```
count = 0

while (count < 9):
   print("The  count is:", count)
   count = count + 1

print("Good  bye!")
```




- The while statement in the Python language is used to execute a loop program that, under certain conditions, loops through a program to handle the same tasks that need to be repeated.

## for Statements

- In the Python language, the for loop can traverse any items of a sequence, such as a list, a dictionary, or a string.
- The for statement is different from a traditional for statement. The former accepts an iterative object (such as a sequence or iterator) as its argument, and one element is iterated each time.

```
for num in nums:
    if num == 1:
        print(num+"---")
    elif num == 2:
        print(num+"///")
    else:
        print('break not triggered')
```





- The for statement is a loop control statement in Python. It can be used to traverse an object and also has an optional else block attached, which is used primarily to handle the break statements contained in a for statement.
- If the for loop is not terminated by a break, the statement in the else block is executed.
- A break statement terminates the for loop when needed.
- A continue statement skips the following statement and starts the next round of loops.

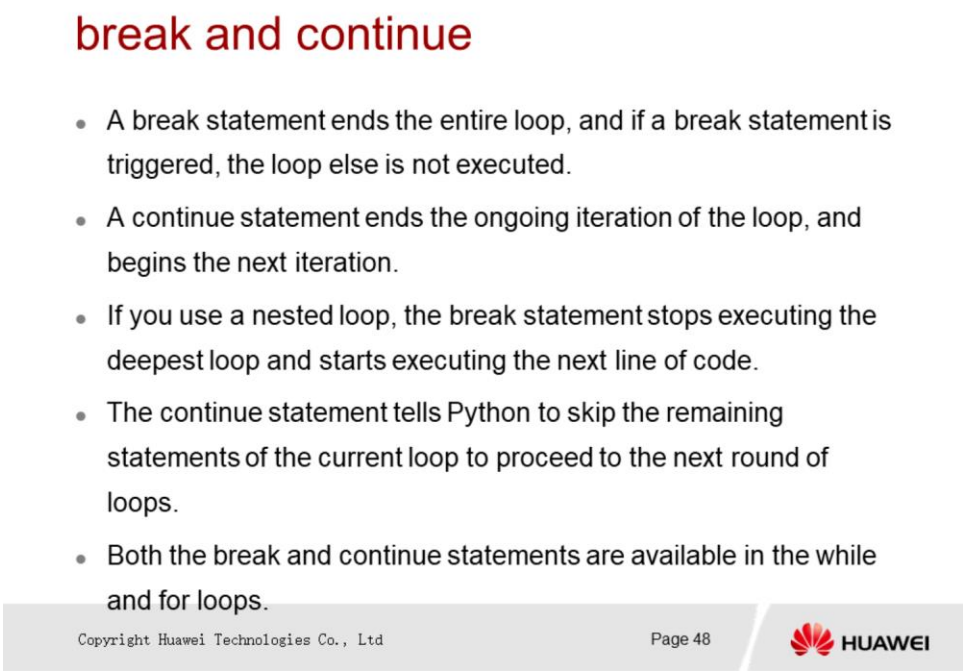
# Loop Nesting

- Python allows one loop to be nested in another loop.
- Syntax of for loop nesting:

```
for iterating_var in sequence:
    for iterating_var in sequence:
        statement(s)
    statement(s)
```

- Syntax of while loop nesting:

```
while expression:
    while expression:
        statement(s)
    statement(s)
```




- The while statements have other important commands, that is, continue and break, to skip loops. continue skips the loop, and break exits the loop. The "judging condition" can also be a constant, indicating that the loop must be true.

# Contents

# Python Functions

- A function is a code segment that is organized, reusable, and used to implement a single function or associated functions.
- Functions can improve the modularity of applications and reuse of code.
- Python provides a number of built-in functions, such as print(). You can also create your own functions, which are called user-defined functions.

# Defining a Function

- Define a function with the following rules:
  - The function code block begins with a def keyword, followed by the function name and parentheses ().
  - Any incoming arguments and independent variables must be placed in the middle of the parentheses. Parentheses can be used to define arguments.
  - The first line of the function can selectively use the document string to hold the description of the function.
  - The function content starts with a colon and indents.
  - *return[expression]* ends a function, and selectively returns a value to the caller. Returning without an expression is equivalent to returning none.

# Calling a Function

- Defining a function gives only the function a name, specifies the arguments contained in the function, and the code block structure.
- After the basic structure of this function is complete, you can execute it through another function call, or you can execute it directly from the Python prompt.

```
# Define a function
def test(str):
    "print any incoming string"
    return str;

# Call a function
test("I want to call a user-defined function!");
test("call the same function again");
```

# Transferring Arguments

- In Python, a type belongs to an object, and a variable is of no type.

```
a = [1,2,3]
a = "Huawei"
```

- In the above code, [1,2,3] is the list type, "Huawei" is a string type, and the variable a is of no type, which is only a reference (a pointer) to an object, and can be a list type object, or a string type object.

# Argument Types

- The following are the formal argument types you can use when calling a Python function:
  - Essential argument: The essential arguments must pass in the function in the correct order, and the number of arguments for calling should be the same as defined.
  - Keyword argument: Keyword arguments and functions are called closely, and function calling uses keyword arguments to determine the values of incoming arguments.
  - Default argument: When a function is called, if the value of the default argument is not transferred, it is considered that a default value is used.
  - Indefinite length argument: You may need a function to handle more arguments than those originally stated. These arguments are called indefinite arguments and are not named when they are stated.

# Anonymous Functions

- Python uses lambda to create anonymous functions:
  - lambda is only an expression, and its function body is much simpler than def.
  - The body of lambda is an expression, not a block of code. Only limited logic can be encapsulated in lambda expressions
  - A lambda function has its own namespace and cannot access arguments outside of its own argument list or in the global namespace.
  - Although a lambda function may seem to write only one line, it is not the same as the C or C ++ inline function, which is designed to call small functions without consuming stack memory and therefore increases operational efficiency.

# Global Variables and Local Variables

- A variable defined within a function has a local scope and is called a local variable, and a variable defined beyond a function has a global scope and is called a global variable.
- Local variables can only be accessed within the stated function, and global variables are accessible within the entire program.
- When a function is called, all variable names stated within the function are added to the scope.

# OS-Related Calling and Functions - sys

- System-related Information module sys.
- SYS.ARGV passes arguments from outside the program to the program.
- Sys.stdout, Sys.stdin, and Sys.stderr represent the standard input and output, the file object of error output, respectively.
- Sys.stdin.readline () reads a line from the standard input and sys.stdout.write ("a") outputs a on the screen.
- Sys.exit () exits the program.
- Sys.modules is a dictionary that holds all imported modules.
- Sys.platform Gets the running OS environment.
- Sys.path is a list that indicates the path to identify modules.

# OS-Related Calling and Functions - os

- os.environ contains the mapping between environment variables; os.environ["Home"] can get the value of the environment variable Home.
- os.chdir (dir) changes the current working directory; os.chdir('d:\\outlook').
- os.getcwd() gets the current directory.
- os.getegid () gets a valid group ID; Id;os.getgid () gets a valid group ID.
- os.getuid () gets a user ID; os.geteuid() gets a valid user ID.
- os.setegid () os.setegid () Os.seteuid () Os.setuid ().
- os.getgruops () gets a list of user group names.
- os.getlogin () gets the user login name.
- os.getenv () gets the environment variable.
- os.putenv () sets the environment variable.
- os.umask () sets the current permission mask and returns the previous permission mask. It is valid in Unix and windows.
- os.system (cmd) uses system calls to run the cmd command.

# Contents

# Object-Oriented Programming

- Object-oriented programming (OOP) is a program design philosophy. OOP takes objects as the basic units of a program, and an object contains data and functions that manipulate data.
- Process-oriented programming treats a computer program as a series of command sets, that is, the sequential execution of a set of functions. In order to simplify program design, process-oriented programming divides functions into sub functions, that is, to reduce the system complexity by cutting block functions into smaller functions.
- OOP treats computer programs as a collection of objects, and each object can receive messages from other objects and process them. The execution of a computer program is a series of messages passing between objects.
- In Python, all data types can be treated as objects, and objects can be customized. The custom object data type is the concept of class in OOP.

# Object-Oriented Design Philosophy

- The idea of object-oriented design (OOD) derives from nature, because the concepts of class and instance are natural.
- Class is an abstract concept. For example, our definition of class "student" refers to the concept of student, and instances refer to specific students such as Bart Simpson and Lisa Simpson. Therefore, the object-oriented design philosophy is to abstract classes and create instances according to classes.
- OOD has a higher abstraction than function because a class contains data and methods of manipulating data.

# Relationship Between OOD and OOP

- OOD does not specifically require OOP languages. Actually, OOD can be implemented by a purely structured language, such as C, but if you want to construct data types that have the nature and characteristics of objects, you need to do more work on the program. When OO features are built into a language, OOP will be more efficient.
- On the other hand, an OOP language does not necessarily force you to write OO-related programs. For example, C++ can be taken as "better C"; Java, in turn, requires that everything be classes, and that a source file corresponds to a class definition. In Python, however, classes and OOP are not necessary for daily programming. Although it was designed from the outset to be object-oriented and structured to support OOP, Python does not qualify or require you to write OO code in your application.
- OOP is a powerful tool, and no matter whether you are ready to enter, learn, transition, or turn, OOP can be arbitrarily controlled.

# Common Python OOP Terms

- Abstract/Implementation
- Encapsulation/Interface
- Composition
- Derivation/Inheritance/Inheritance Structure
- Generalization/Specialization
- Polymorphism
- Introspection/Reflection

# Classes

- A class is a data structure that can be used to define objects, which combine data values with behavioral characteristics. A class is a real-world abstract entity that appears in a programmatic fashion. Instances materialize these objects. As an analogy, a class is a blueprint or a model used to produce real objects (instances).
- In Python, a class statement is similar to a function statement, with a corresponding keyword in the first line, followed by a body of code as its definition, as follows:

```
def functionName(args):
    'function documentation string'
    function_suite

class ClassName(object):
    'Click class documentation string'
    class_suite
```

# Inheritance

- Inheritance is a way to create a class. In Python, a class can inherit from one or more parent classes. The original class is called a base class or a superclass.
- If there are several classes, and the classes have common variable attributes and function properties, then you can extract these variable properties and function properties as the base class properties. The special variable properties and function properties are defined in this class so that the base class's variable properties and function properties are accessible only if the base class is inherited. This increases the scalability of code.
- Abstraction is the extraction of similar parts.
- A base class is a class that abstracts the properties that are common to multiple classes.

## Composition and Derivation

- Once a class is defined, the goal is to use it as a module and embed the objects into your code, mixed with other data types and logic execution streams. There are two ways to use classes in your code.
- The first way is the composition, which allows different classes to be mixed and added to other classes to add functionality and code reusability. You can create an instance of your own class in a larger class and implement some other properties and methods to enhance the original class object.
- The other way is derivation, which means that a subclass derives a new property on the basis of inheriting the parent class. A subclass may have a unique object that its parent class does not have, or a subclass defines an object with a name repeated in the parent class. A subclass is also called a derivation class.
- When there are significant differences between classes, composition behaves well; but when you design the same class with different functions, derivation is a more reasonable choice.





- For example, let's imagine an enhanced design for this class of addresses created at the beginning of this chapter. If you create a separate class for names and addresses during the design process, then we might want to integrate the work into the Addrbookentry class instead of redesigning each of the required classes. This saves time and effort, and code will be easy to maintain. Correction of a bug in a code will be reflected throughout the application.

# Subclasses

- One of the more powerful aspects of OOP is the ability to use a well-defined class, extend it, or modify it without affecting other snippets of code that use existing classes in the system. OOD allows a class feature inheritance by a descendant class or subclass. These subclasses inherit their core properties from the base class (or ancestor class, superclass). Also, these derivations may be extended to multiple generations.
- Related classes in a hierarchical derivation (or vertically adjacent to a class tree diagram) are of parent and subclass relationships. These classes deriving from the same parent class (or horizontally adjacent to the class tree diagram) are sibling relationships. The parent class and all high-level classes are considered ancestors.

# Privatization

- By default, properties are "public" in Python and can be accessed by the module in which the class resides and by other modules that import the module in which the class resides. Many OO languages add some visibility to the data, providing only the function to access its values.
- Most OO languages provide access control characters to qualify the access of member functions.
- Double underline (__)
  - Python provides a preliminary form for the privacy of class elements (properties and methods). Properties that start with a double underline are "confusing" at run time, and therefore direct access is not allowed. Actually, it will precede the first name with an underscore and a class name.
- Single underline (_)
  - For simple module-level privatization, you only need to use a single underline character before the property name. This prevents the properties of the module from being loaded with the "from mymodule import *". This is strictly based on the scope, and therefore this is also true for functions.

# Contents

# Getting the Current Date and Time

- Let's see how to get the current date and time.

```
>>> from datetime import datetime
>>> now = datetime.now() # Get the current datetime
>>> print(now)
2015-05-18 16:28:07.198690
>>> print(type(now))
<class 'datetime.datetime'>
```

- Note that datetime is a module, and it also contains a datetime class. Only the class imported by from *datetime import datetime* is the datetime class. If you import only the import datetime, you need to refer to the full name datetime.datetime. DateTime.Now () returns the current date and time, with the type datetime.

# Converting datetime to timestamp

- In a computer, time is represented by numbers.
- We refer to the January 1, 1970 00:00:00 utc+00:00 time zone as epoch time, which is recorded as 0 (a negative timestamp for before 1970), and the current time is the number of seconds relative to epoch time, called timestamp. You can take that timestamp = 0 = 1970-1-1 00:00:00 utc+0:00. The corresponding Beijing time is: timestamp = 0 = 1970-1-1 08:00:00 utc+8:00. The value of the visible timestamp has nothing to do with the time zone, because once the timestamp is determined, the UTC time is determined, and time in any timezone after conversion is determined. That is why the current time that the computer stores is represented by timestamp. Because timestamps of the computers around the world are the same at any given time, only the timestamp () method needs to be called to convert a datetime type to timestamp.

```
>>> from datetime import datetime
>>> dt = datetime(2015,4,19,12,20)  #Create using specified datetime
>>> dt.timestamp()    #Convert datetime into timestamp
1429417200.0
```

- Note that Python's timestamp is a floating-point number. If there are decimal places, the decimal places represent the number of milliseconds.
- The timestamp of some programming languages, such as Java and JavaScript, uses integers to represent the number of milliseconds, in which case a floating-point representation of Python can only be achieved by dividing the timestamp by 1000.

# Converting timestamp to datetime

- To convert timestamp to datetime, use the fromtimestamp () method provided by datetime:

```
>>> from datetime import datetime
>>> t = 1429417200.0
>>> print(datetime.fromtimestamp(t))
2015-04-19 12:20:00
```

- Note that timestamp is a floating-point number, and it does not have the concept of a time zone, but datetime has a time zone. Local time refers to the time zone set by the current operating system. For example, the Beijing time zone is East 8, then local time: 2015-04-19 12:20:00. That is, the time of the utc+8:00 time zone: 2015-04-19 12:20:00 utc+8:00. At the moment Greenwich Standard Time and Beijing time offset by 8 hours. That is, time in utc+0:00 time zone should be: 2015-04-19 04:20:00 utc+0:00. Timestamp can also be converted directly to time in the UTC standard time zone:

```
>>> from datetime import datetime
>>> t = 1429417200.0
>>> print(datetime.fromtimestamp(t)) # local time
2015-04-19 12:20:00
>>> print(datetime.utcfromtimestamp(t)) # UTC time
2015-04-19 04:20:00
```

# Converting Local Time to UTC Time

- Local time is the time when the system sets the timezone. For example, Beijing time is the UTC+8:00 time zone, and UTC time refers to the time of the utc+0:00 time zone. A datetime type has a time zone attribute tainfo, but the default is None, and therefore it is impossible to identify a time zone of the datetime, unless you force a time zone for the datetime.

```
>>> from datetime import datetime, timedelta, timezone
>>> tz_utc_8 = timezone(timedelta(hours=8)) #create time UTC+8:00
>>> now = datetime.now()
>>> now
datetime.datetime(2015, 5, 18, 17, 2, 10, 871012)

>>> dt = now.replace(tzinfo=tz_utc_8)      #force time zone as UTC+8:00 >>> dt
datetime.datetime(2015, 5, 18, 17, 2, 10, 871012, tzinfo=datetime.timezone(datetime.timedelta(0,  28800)))
```

# Contents

# Regular Expressions (1)

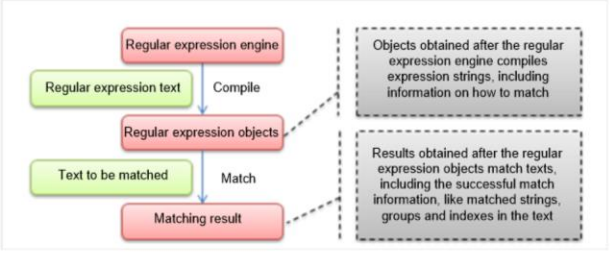
- A regular expression is a string of characters and special symbols that describe a pattern's repetition or multiple characters, and therefore a regular expression can match a series of strings with similar characteristics in a pattern.
- Regular expressions provide the basis for advanced text pattern matching, extraction, and/or text-style search and replace functions.
- Python supports regular expressions by using the re modules in the standard library.

# Regular Expressions (2)

- A regular expression is a special sequence of characters that can help you easily check whether a string matches a pattern.
- The re module enables the Python language to have full regular expression functionality.
- The re module also provides functions that are identical to those of the methods, which use a pattern string as their first argument.
- The compile function generates a regular expression object based on a pattern string and an optional flag argument. This object has a series of methods for regular expression matching and substitution.
- match attempts to match a pattern from the starting position of the string, and match () returns none if the start position match succeeds.
- search scans the entire string and returns the first successful match.

# Matching Process for Regular Expressions

- The approximate matching process for regular expressions is to match characters between regular expressions and texts. If each character matches, the match succeeds, and the match fails if there is any character match failure.



# re Modules

- Python provides support for regular expressions by using the re module.
- The general step in using re is to compile the string form of a regular expression into a pattern instance, and then use the patterns instance to process the text and get the matching result (a match instance), and finally use the match instance to get the information and do other things.

```
import re

# Compile a regular expression into a patten object
pattern = re.compile(r'hello')

# Match text with pattern and return none if no match
match = pattern.match('hello  world!')

if match:
    # Get group information using match
    print(match.group())

# Output
hello
```

# re Module Functions and Regular Expression Object Methods

| Function/Method | Description | Example | res.group()/res |
|---|---|---|---|
| compile(pattern,flag=0) | Compiles a regular expression pattern using any optional flag and returns regular expression objects. | res = re.compile(".*")<br>print res.search("abcd").group() | abcd |
| match(pattern,string,flag=0) | Matches from the start of string. | res = re.match(".*","abcdxxxx") | abcd |
| search(pattern,string,flag=0) | Matches from any position of string. | res = re.search(".*","xxxabcdxx") | abcd |
| findall(pattern,string,flag=0) | Finds all regular expression patterns in strings and returns a list. | res = re.findall("a", "abdadafaf") | ['a','a','a','a'] |
| finditer(pattern,string,flag=0) | Finds all regular expression patterns in strings and returns an iterator. | res = re.finditer("a", "abdadafaf")<br>print res.next().group() | a |
| split(pattern,string,max=0) | Splits a string into a list by regular expression pattern. | re.split(",","li,yang,zhao") | ['li','yang','zhao'] |
| sub(pattern,repl,string,count=0) | Counts the positions with repl regular expressions in strings. | res = re.sub(",","-","l,y,z") | l-y-z |

# Common Object Matching Methods

| Function/Method | Description | Example | Result |
|---|---|---|---|
| group(num=0) | Returns the entire match object or a numbered group. | Print(re.match(".*","abcdxxxx").group()) | abcdxxxx |
| groups(default=None) | Returns a tuple with all groups. | Print(re.search("(\w\w\w)-(\d\d\d)","abc-123").groups()) | ('abc', '123') |
| groupdict(default=None) | Returns a dictionary with all matched named groups (names are key values). | res = re.search("(?P<lamb>\w\w\w)-(?P<num>\d\d\d)","abc-123")<br>Print(str(res.groupdict())) | {'lamb':'abc', 'num': '123'} |
| re.i,re.IGNORECASE | Ignore the upper and lower case. | res =re.search("abc","aBcxx",re.I)<br>Print(res.group()) | aBc |
| re.L,re.LOCAL | Performs matching by using \w, \W,\b,\B,\s,\S based on the local language environment. | res = re.search("\w\w\w","aBcxx",re.L)<br>Print(res.group()) | aBc |
| re.M,re.MULTILINE | Respectively matches start and end of target strings with ^ and $, but not exact start and end of strings. | res = re.search("^aB","aBcxx",re.M)<br>Print(res.group()) | aB |

# compile

- This method is the factory method of the pattern class, which is used to compile a regular expression in a string as a patterns object. The second argument flag is a matching pattern, which can be used by bitwise or operator ' | ', meaning that it takes effect at the same time, such as re.I | re.M. Alternatively, you can specify patterns in the regex string, such as e.compile('pattern', re.I | re.M), which is equivalent to re.compile('(?im)pattern').
- Optional values:
  - re.I(re.IGNORECASE): ignores case
  - M(MULTILINE): multi-line mode, changing behavior of'^' and '$'.
  - S(DOTALL): dot all match mode, changing behavior of'.'
  - L(LOCALE): makes predefined string classes \w \W \b \B \s \S depend on setting of a current area.
  - U(UNICODE): makes predefined string classes \w \W \b \B \s \S \d \D depend on character properties defined by unicode.
  - X(VERBOSE): verbose mode. In this mode, regular expressions can be multi-lined, space characters are ignored, and comments can be added.

# Patterns

- A pattern object is a compiled regular expression that can be matched to a search by a series of methods provided by pattern.
- Pattern cannot be directly instantiated and must be constructed using re.compile().
- Pattern provides several readable properties for getting information about an expression:
  - pattern: an expression string used during compilation.
  - flags: a matching pattern for compilation, which has a digital form.
  - groups: the number of groups in an expression.
  - groupindex: A dictionary with the alias of an alias group in an expression as the key and the group number as the value. Groups with no alias are excluded.

# match (1)

- The match object is a matching result that contains a lot of information about the match, and you can use the readable properties or methods provided by match to get that information.
- Match attributes:
  - string: The text to be used for matching.
  - re: The pattern object used for matching.
  - pos: The index in which the regular expression begins the search. The value is the same as the argument with the same name for the Pattern.match () and Pattern.seach () method.
  - endpos: The index in which the regular expression ends the search. The value is the same as the argument with the same name for Pattern.match () and Pattern.seach () methods.
  - lastindex: The index of the last captured group in text. If no groups are captured, it will be none.
  - lastgroup: The alias of the last captured group. If this group has no alias or is not a captured group, it will be none.

# match (2)

- match methods:
  - group([group1, ...]): Gets a string that is intercepted from one or more groups, and returns a tuple when multiple arguments are specified. group1 can use the number or alias, and number 0 represents the entire matched substring. It returns group(0) if no argument is filled, none for groups without string interception, and the last intercepted substring for groups with repeated string interceptions.
  - groups([default]): Returns intercepted strings of all groups in a tuple form. It is equivalent to calling group(1,2,... last). default indicates that a group with no intercepted string is substituted with this value, and it defaults to none.
  - groupdict ([default]): Returns a dictionary with the alias of an alias group as the key and the substring intercepted from the group as the value. A group without alias is excluded. default has the same meaning.
  - start ([group]): Returns the start index (the index of the first character of the substring) of the intercepted substring in string for the specified group. group defaults to 0.
  - end ([group]): Returns the end index (the index of the last character of the substring + 1) of the intercepted substring in string for the specified group. group defaults to 0.
  - span ([group]): Returns (start(group), end (group)).
  - expand (template): Converts a matched group into a template and returns the conversion result. In template, id or g<id>, g <name> can be used for reference, but number 0 cannot be used. id<id> is equivalent to g<id>, but \10 will be considered as the 10th group. If you want to express \1 followed by character ' 0 ', you can only use /g<1>0.





- **group([group1, ...]):**
  **Gets a string that is intercepted from one or more groups, and returns a tuple when multiple arguments are specified. group1 can use the number or alias, and number 0 represents the entire matched substring. It returns group(0) if no argument is filled, none for groups without string interception, and the last intercepted substring for groups with repeated string interceptions.**
- **groups([default]):**
  **Returns intercepted strings of all groups in a tuple form. It is equivalent to calling group(1,2,... last). default indicates that a group with no intercepted string is substituted with this value, and it defaults to none.**
- **groupdict ([default]):**
  **Returns a dictionary with the alias of an alias group as the key and the substring intercepted from the group as the value. A group without alias is excluded. default has the same meaning.**
- **start ([group]):**
  **Returns the start index (the index of the first character of the**

**substring) of the intercepted substring in string for the specified group. group defaults to 0.**

- **end ([group]):**
  **Returns the end index (the index of the last character of the substring + 1) of the intercepted substring in string for the specified group. group defaults to 0.**
- **span ([group]):**
  **returns (start(group), end (group)).**
- **expand (template):**
  **Converts a matched group into a template and returns the conversion result. In template, id or g<id>, g <name> can be used for reference, but number 0 cannot used. id<id> is equivalent to g<id>, but \10 will be considered as the 10th group. If you want to express \1 followed by character ' 0 ', you can only use /g<1>0.**

# Special Symbols and Characters - Symbols (1)

| Symbol | Description | Matched Expression | res.group() |
|---|---|---|---|
| literal | Matches literal values of text strings. | res=re.search("foo","xxxfooxxx") | foo |
| re1\|re 2 | Matches regular expressions re1 or re2. | res=re.search("foo\|bar","xxxfooxxx") | foo |
| . | Matches any character (except \n). | res=re.search("b.b","xxxbobxxx") | bob |
| ^ | Matches string start. | res=re.search("^b.b", "bobx xx") | bob |
| $ | Matches string end. | res=re.search("b.b$","xx xbob") | bob |
| * | Matches regular expressions that appear none or many times (from string start). | res= re.search("bob*","bobbo")<br>res1= re.search(".*","bobboddd") | Bobb<br>bobboddd |
| + | Matches regular expressions that appear once or many times. | res= re.search("bob+","xxxxbobbbbob") | bobbbb |

# Special Symbols and Characters - Symbols (2)

| Symbol | Description | Matched Expression | res.group() |
|---|---|---|---|
| ? | Matches regular expressions that do not appear or appear once. | res=re.search("bob?","bobbod") | bob |
| {N} | Matches regular expressions that appear N times. | res=re.search("bob{2}","bobbbbod") | bobb |
| {M,N} | Matches regular expressions that appear M to N times. | res=re.search("bob{2,3}","bobbbbod") | bobbb |
| [...] | Matches any character from a character set. | res= re.search("[b,o,b]","xxbobxx") | b |
| [..X-Y..] | Matches any character in the x to y range. | res= re.search("[a-z]","xxbobxx") | x |
| [^...] | Does not match any character in a string, including those in a range. | res= re.search("[^a-z]","xx214bobxx")<br>res1=re.search("[^2,x,1]","xx214bobxx") | 2<br>4 |
| (*\|+\|{})? | Matches non-greedy versions of symbols that appear frequently or repeatedly. | res= re.search(".[+\|?]?[1-9]","ds4b") | s4 |

# Special Symbols and Characters - Characters

| Character | Description | Matched Expression | res.group() |
|---|---|---|---|
| \d | Any decimal number (\D does not match any decimal number). | res=re.search("xx\dxx","oxx4xxo") | xx4xx |
| \w | Matches any letter or number (\W means no match). | res=re.search("xx\w\wxx","oxxa4xxo") | xxa4xx |
| \s | Matches any space character (\S means no match). | res=re.search("xx\sxx","oxx xxo") | xx xx |
| \b | Matches any letter boundary (\B means reverse). | res=re.search(r"\bthe","xxx the xxx") | the |
| \N | Matches saved sub-group. | | |
| \c | Matches any special character c one by one. | res=re.search("*","x*x") | * |
| \A(\Z) | Matches string start (or end). | res=re.search("\ADear","Dear Mr.Li") | Dear |

# Contents

# Python File Manipulation

- File manipulation is of great importance to programming languages, and information technologies will be meaningless if data cannot be persistently read, saved, or used.
- Common types of file manipulation include opening and closing files, reading and writing files, and backing up files.

# File Manipulation (1)

- Opening a file
  - f.open('file name','access mode')
  - Common access modes:

| Access Mode | Description |
|---|---|
| r | Opens a file only for reading. |
| w | Opens a file only for writing. |
| a | Opens a file only for addition. |
| rb | Opens a file using the binary format only for reading. |
| wb | Opens a file using the binary format only for writing. |
| ab | Opens a file using the binary format only for addition. |
| r+ | Opens a file for reading. |
| w+ | Opens a file for writing. |
| a+ | Opens a file for addition. |
| rb+ | Opens a file using the binary format for reading. |
| wb+ | Opens a file using the binary format for writing. |
| ab+ | Opens a file using the binary format for addition. |

# File Manipulation (2)

- Writing data:

```
f = open("name.txt","w")
f.write("libai")
f.close()
```

- Reading data:

```
f = open("name.txt","r")

lines = f.realines()
for line in lines:
   print(line)
```

- Closing a file:

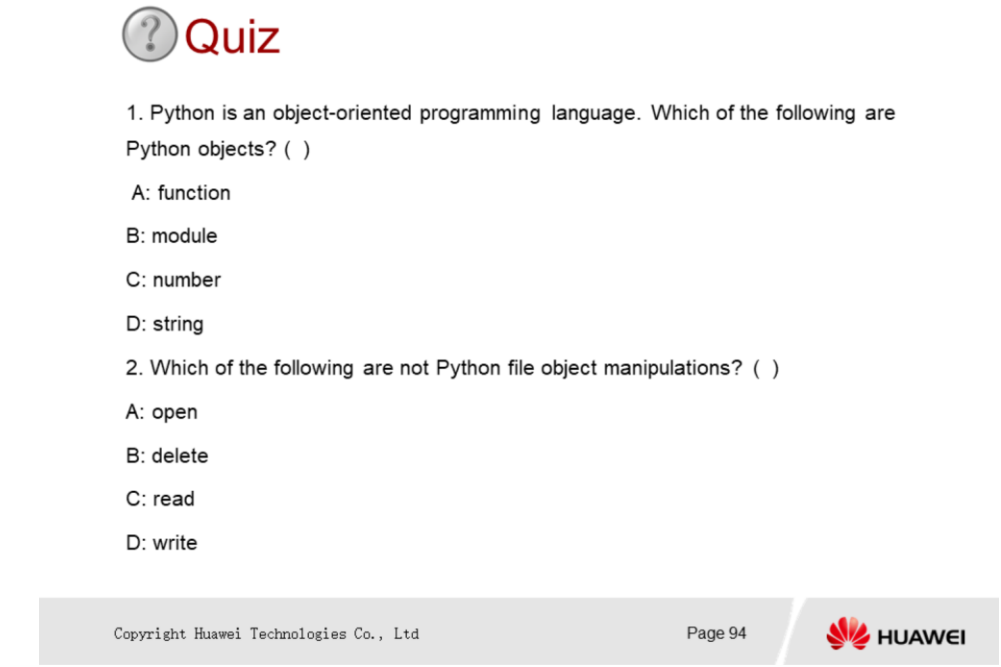
```
f.close()
```

# Renaming Files in a Batch

- Get the target folder:
  - dirName = input(“enter specified folder:”).
- Get names of files in the target folder:
  - fileNames = dirName.listdir(dirName).
  - os.chdir(dirName).
- Rename
  - for name in fileNames.
  - os.rename(name,“zhangsan”+name).

# Other File Manipulation Functions

- Writing and reading files:
  - f.write("a") f.write (str) writes a string f.writeline (). f.readlines () is similar to the following read class.
  - f.read() reads all. f.read (size) reads characters of the size number from a file.
  - f.readline() reads a line, and returns an empty string to the file end. f.readlines() reads all, and returns a list. Each element of the list represents a row containing the " \n" \.
  - f.tell() returns the current file read location.
  - f.seek (off, where) locates the file read/write location. off represents an offset, a positive number means offset to the file end, and a negative number means offset to the file start.
  - where: 0 means counting from the beginning, 1 means counting from the current position, and 2 means counting from the end.
  - f.flush() flushes the cache.




- Answer:

1. Because they all have similar object-oriented features and syntax. Java syntax is much simpler than C++, but it's still a bit cumbersome, especially if you want to accomplish a small task. Python's simplicity provides a faster development environment than the sheer use of Java. A very important revolution in the relationship between Python and Java is the development of Jython.

# Summary of the Chapter

- This course focuses on the Python language and its basic syntax, such as the Python compilation environment and installation of digital expressions, variables, statements, strings, access to user input, functions, modules and other common operations.

# More Information

- Official site:
  - www.python.org
- References
  - Learning Python
  - Python Standard Library
  - Programming Python



# Recommended for Learning

- Huawei e-learning site:
  - http://support.huawei.com/learning/Index!toTrainIndex
- Huawei knowledge base on the support website:
  - http://support.huawei.com/enterprise/servicecenter?lang=zh

# Thanks

www.huawei.com