Huawei AI Academy Training Materials

Python Basics

Huawei Technologies Co., Ltd.

Copyright © Huawei Technologies Co., Ltd. 2020. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services, and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services, and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either expressed or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base Bantian, Longgang Shenzhen 518129

Website: https://e.huawei.com

Contents

1 Introduction to Python

Python is one of the most popular programming languages, and is the most widely used programming language in the artificial intelligence (AI) field. Python 2 and Python 3 are mainstream versions. Here, we will learn about Python 3.

1.1 Overview

Python is a universal advanced programming language. It is completely open-source. The author of Python is Guido Van Rossum.

1.2 Advantages and Disadvantages of Python

Advantages:

Python is an advanced object-oriented programming language.

It is a dynamic and interpretive language.

It has elegant structure and clear syntax, which is easy to learn.

It has a huge collection of third-party library, and

It can invoke code written in other languages, therefore it is known as "glue language".

It also supports functional programming.

Disadvantages:

Low running speed

1.3 Python Application Fields

Python has abundant third-party libraries and advantages of the Python language. Therefore, Python can be used in many fields, such as artificial intelligence, data science, system tool compilation, application development, O&M script automation, and web development.

1.4 Python Environments

1.4.1 Installing the Python Interpreter

Download the interpreter file from the official website and install it (each system has its mapping version). After the installation is complete, configure environment variables (Python allows multiple versions to coexist).

Install the Anaconda. The Anaconda is a Python interpreter that integrates multiple third-party libraries and is widely used in AI and scientific computing. The Anaconda has two versions that apply to Python 2 and Python 3.

1.4.2 IDE

PyCharm: a development environment with extremely powerful and convenient functions

Eclipse: a development tool widely used in Java and Python

Jupyter Notebook: web-based interactive computing environment.

2 Basic Programming

2.1 Python Basics

2.1.1 Basic Syntax

Python uses indentation to divide statement blocks.

An indentation contains four spaces and can be inserted by pressing the Tab key.

Python programs are executed from top to bottom.

Packages and modules are imported using the **import** and **from...import...** statements.

If multiple statements are in one line, use semicolons (;) to separate them.

A number sign (#) is used to comment out one line, and a doc string('''... ''',"""..."""") is used to comment out multiple lines.

PEP8 (not mandatory)

PEP8 is a style guide that Python code complies with, not a syntax rule. PEP 8 helps improve code readability and elegance.

Keywords: identifiers defined in Python with special functions.

Identifier naming rules:

An identifier consists of letters, underscores (_), and digits, and cannot start with a digit. User-defined identifiers cannot share the same names as predefined keywords.

Variables (reference of data storage addresses):

When data is stored in a computer, a corresponding storage address can be obtained.

Assigning a value to a variable is not assigning data to the variable, but assigning the storage address of the data to the variable.

Scope: scope within which variables can be accessed when a program is running.

Local variables: variables defined in a function. The variables can only be used within the function.

Global variables: variables defined outside of functions and objects. The variables can be used within the entire module.

2.1.2 Basic Python Built-in Function

print(): output function.

print("hello world"): generates "hello world".

input(): receives user input.

del(obj): deletes an object from the memory.

a="python"; del(a)

range(start, stop, [step]): generates an iterative sequence (including the start position, end position, and step).

range(0,20,4)

type(obj): type of the returned object.

type(print): generates builtin_function_or_method.

dir(obj): views the built-in methods and attributes of an object.

dir(print)

```
>>> dir(print)
['__call__', '__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__
gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__name__', '__ne__', '__new__',
__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__self__', '__setattr__', '__sizeof__', '__str__', '__subclas
shook__', '__text_signature__']
>>>
```

id(obj): views the object memory address.

A=1;id(A): generates result 1735879424.

help(obj): displays the help information about an object.

help(print)

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

2.2 Data Structure in Python

2.2.1 Data Structure Classification

Python has the following common data types: number, string, list, tuple, dictionary, and set. These data types not only improve Python running efficiency, but also greatly improve our development efficiency. They also make Python easy to operate.

Python data types are classified into the following types:

Sequential: Subscripts (indexes) can be used to access elements. You can access elements in slice mode such as [start:stop:step].

Nonsequential: Subscripts (indexes) cannot be used to access elements.

Changeable: Values can be modified.

Unchangeable: Values cannot be modified.

2.2.2 Number

Python 3 supports values of int, float, bool, and complex types.

Basic operations of the Number type are as follows:

Addition (+)

Subtraction (-)

Multiplication (x)

Division (/)

Modulo/Rounding (%, //)

Power (**)

If the operation is performed on numbers of different types (such as int and float), the result type is the type with higher precision.

2.2.3 String

In Python, a string is a sequence with multiple characters. The number of characters indicates the length of the string.

Python does not have a character data type. A single character is considered as a string with length 1.

To declare a string, you only need to use single quotation marks ('...') or double quotation marks ("...") to enclose the content. You can also use three consecutive quotation marks ('''...''' or """...""").

The escape character (\) and the original string **r** can be used in the string.

Operator:

+: Two strings are concatenated. Example: a="hello";b="world" =>a+b= 'helloworld'.

*: A new string is obtained by multiplying a string by a number. Example: "a"*2=>"aa"

2.2.4 Common Operations on Strings

Table 2-1 Common operations on strings

Operation	Definition	Examples
Segmentation	str.split(str1): Splits a string using str1 as a separator.	'python'.split('h') Output: ['pyt', 'on']
Replace	str.replace(str1, str2): Replaces str1 in the string with str2 to generate a new character string.	'python'.replace('py','PY') Output: PYthon
Uppercase	str.lower(): Converts uppercase letters in a string to lowercase letters.	'PYTHON'.lower() Output: python

Operation	Definition	Examples
Lowercase	str.upper(): Converts lowercase letters in a string to uppercase letters.	'python'.upper() Output: PYTHON
Stitching	str.join(iter): Concatenates each element in the given parameter with the specified character to generate a new string.	"-".join("huawei") Output: h-u-a-w-e-i
Formatted output	Uses the formatting operator (%), string conversion type, and formatting operator auxiliary instructions to implement formatted output.	'My name is %s , age is %d' %('AI', 63) Output: My name is AI, age is 63

2.2.5 String Formatted Output

Table 2-2 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 and uppercase)
%e	Floating point (scientific exponential notation)
%E	Floating point (scientific E-notation)
%f	Floating point (decimal)

Format	Description
%g	Floating point (same as %e or %f depending on the value)

Table 2-3 Auxiliary formatting commands

Symbol	Description
*	Defines the width or decimal precision.
-	Used for left alignment.
+	Displays a plus sign (+) before a positive number.
<sp>	Displays a space before a positive number.
#	Displays a zero (0) before an octal number, and 0x or 0X before a hexadecimal number (depending on whether x or X is used).
0	Pads a zero (0) for numeric values instead of a default space.
(var)	Mapping variable (dictionary parameter)
m.n	m indicates the minimum total width of the display, and n indicates the number of digits after the decimal point.

2.2.6 List

A list is a sequence in which elements can be of any data type and elements can be added or deleted at any time.

In a list, elements are enclosed by a pair of square brackets and are separated by commas (,). You can create a list in either of the following ways:

List = list(obj1, obj2,...)

List = [obj1, obj2, ...]

List comprehensions

Operator:

+: Combines lists, for example, the result of [1,2]+[2,3] is [1, 2, 2, 3].

x: Multiplies a list by a number to obtain a new list, for example, the result of [1,2] x 2 is [1, 2, 1, 2].

2.2.7 Common Operations on Lists

Table 2-4 Common operations on lists

Operation	Definition	Examples
Add	list.append(obj): adds an object in a parameter to the end of the list.	a=[1,2]; a.append(3); a Output: [1,2,3]
	list.insert(index, obj): inserts an object to the index position of a list.	a=[1,2];a.insert(0,3);a Output: [3, 1, 2]
	list.extend(iter): inserts each element of an iterable object into the tail of a list one by one.	a=[1,2];a.extend([3,4]);a Output: [1, 2, 3, 4]
Delete	list.pop([index]): deletes the element from the position of the index parameter and returns the deleted element. If no parameter is passed, the last element is deleted by default.	a=[1,2];b=a.pop(1);a,b Output: [1],2
	list.remove(obj): deletes the first given element in the list.	a=[1,2];a.remove(2);a Output: [1]
Search	list.index(obj): returns the index of the first occurrence of a given element.	a=[1,2];a.index(2);a Output: 1
Sort	list.sort(): sorts the list. The default sorting order is ascending.	a=[3,1,2];a.sort();a Output: [1,2,3]
Reverse	list.reverse(): reverses the elements in the list (by directly modifying the list itself).	a=[3,1,2];a.reverse();a Output: [2,1,3]
Count	list.count(obj): returns the number of occurrences of a given element.	a=[3,1,2,1];a.count(1) Output: 2

2.2.8 Tuple

A tuple is a sequence in which elements can be of any data type.

Data stored in tuples is of higher security than that on lists.

Elements in a tuple are enclosed by a pair of parentheses and are separated by commas (,). A tuple can be created in the following three ways:

Tuple = tuple(obj1, obj2, ...)

Tuple = (obj1, obj2, ...)

Tuple = obj1,obj2,obj3

If a tuple has only one element when it is created, a comma must be added to the end of the element to tell the interpreter that this is not a parenthesis of the operator.

2.2.9 Dictionary

Each element of the dictionary consists of a key and value. Therefore, the elements of the dictionary are also called key-value pairs. A key is immutable and unique. If a dictionary has duplicate keys, the value of a later key overwrites the value of the previous key.

When there is a large amount of data, the access speed of dictionary data is higher than that of a list block.

Elements in a dictionary are enclosed by a pair of braces and are separated by commas (,). Common methods of creating a dictionary are as follows:

Dict = {key:value,}

Dict = dict(key=value,)

Dict = dict([(key,value),])

Dictionary comprehensions

2.2.10 Common Operations on Dictionaries

Table 2-5 Common operations on dictionaries:

Operation	Definition	Examples
acquisition	dict.get(key, default=None): obtains the value based on the key. If the key does not exist, the default value is returned.	Dict={'a':1,'b':2}; Dict.get('a') Output: 1
	dict.items(): returns a list of all (key, value) tuples.	Dict={'a':1,'b':2}; Dict.items() Output: dict_items([('a', 1), ('b', 2)])
	dict.keys(): returns a list of all keys.	Dict={'a':1,'b':2}; Dict.keys() Output: dict_keys(['a', 'b'])
	dict.values(): returns a list of all values.	Dict={'a':1,'b':2}; Dict.items() Output: dict_values([1, 2])
Add a member drive	dict[key] = value: adds the key-value pair {key:value}. If the key already exists, change the value of the existing key.	Dict={'a':1,'b':2}; Dict['a']=3; Dict Output: {'a':3,'b':2}

Operation	Definition	Examples
Update	dict.update(dict1): uses dict1 to update the dictionary.	Dict={'a':1,'b':2}; Dict2={'a':3,'c':3}; Dict.update(Dict2); Dict Output: {'a': 3, 'b': 2, 'c': 3}
Delete	dict.pop(key): deletes and returns the value of the key.	Dict={'a':1,'b':2};a=Dict.pop('a') ; Dict,a Output: ({symptom: 2}, 1)
	Dict.popitem(): deletes and returns a key-value pair randomly.	Dict={'a':1,'b':2};a=Dict.popitem(); Dict,a Output: ({'a', 1}, ('b', 2))
	dict.clear(): clears the dictionary.	Dict={'a':1,'b':2}; Dict.clear(); Dict Output: {}

2.2.11 Set

Every element in a set is unique, and duplicate elements are deleted.

Elements in a set are enclosed by braces and are separated by commas (,). You can create a set in the following ways:

Set = set()

Set = {obj1,obj2,...}

Logical operations:

Intersection set1 & set2: same elements in the two sets

Symmetric difference set1 ^ set2: elements which are in either of the sets and not in their intersection

Union set1 | set2: all elements in the two sets with duplicate elements deleted

Difference set set1 - set 2: elements contained in set 1 but not contained in set 2.

2.2.12 Common Operations on Sets

Table 2-6 Common operations on sets

Operation	Definition	Examples
Add a member drive	set.add(obj): adds an element. If the element already exists, no operation is performed.	Set={1,2,3}; Set.add(4); Set Output: {1, 2, 3, 4}

Operation	Definition	Examples
	set.update(obj): adds an object which can be a list, a dictionary, or others. Multiple objects can be added and need to be separated by commas (,).	Set={1,2};Set.update({2,3});Set Output: {1, 2, 3}
Delete	set.remove(obj): removes an element. (If the element to be deleted does not exist, an exception is thrown.)	Set={1,2};Set.remove(1);Set Output: {2}
	set.discard(obj): deletes an element. (No exception is thrown if the element does not exist.)	Set={1,2};Set.discard(1);Set Output: {2}
	set.clear(): removes all elements from a set.	Set={1,2};Set.clear();Set Output: set()
	set.pop(): removes a random element from a set.	Set={1,2};a=Set.pop();Set,a Output: ({2}, 1)

2.2.13 Deep Copy and Shallow Copy

In Python, data copy can be classified into deep copy and shallow copy.

Shallow copy (copy()): Copies the data structure. If the data is in a nested structure, the elements in the nested structure are references to the original data. Modification of the original data affects the copied data.

Deep copy: Compared with the structure reference in shallow copy, all data is copied, and modification of the original data does not affect the copied data.

To use deep copy, import the copy module in Python and use the deepcopy() method in the module.

2.2.14 Operator

Python has the following operators:

Arithmetical operator

Comparison operator: ==, !=, >, <, >=, <=

Assignment operator: =, +=, –=, /=, *=, **=, //=

Bitwise operator: &, |, ^

Logical operator: and, or, not

Membership operator: in, not in

Identity operator: is, is not

2.3 Control Flow

2.3.1 Judgment Statement – if

The condition control in Python determines the code block to be executed based on the execution result (True or False) of the conditional statement.

In Python, if is used to control program execution. If there are multiple conditions, the if – elif – else format can be used.

```
if condition 1:
    Statement 1
elif condition 2:
    Statement 2
else:
    Statement 3
```

2.3.2 Loop Statement – for

The for statement in Python is different from the ones in other languages. It takes an iterable object (such as a sequence) as its parameter and iterates one element at a time.

You can add the else statement block following the for loop and execute the statement block when the loop ends.

If a for loop is not terminated by a break statement, the statement in the else block is executed.

The for statement is used in the following way:

```
for iter in iters:
    Loop statement block
else:
    Statement block
```

2.3.3 Loop Statement – while

In Python, the while statement is used to execute a loop program. Under certain conditions, a loop program is executed to process the same tasks repeatedly.

When the condition of the while statement is always true, the loop will never end, forming an infinite loop.

You can add the else statement block to the end of the while statement to execute the statement when the condition is false.

Avoid empty while loops which waste resources.

The while statement is used in the following way:

```
while condition statement:
    Statement block that is executed circularly # Execute the statement block when the condition is true.
else:
```

Statement block. # Execute the statement block when the condition is false.

2.3.4 Loop Termination – break and continue

If you want to interrupt a loop, break and continue can be used.

A break statement ends the entire loop. If break is triggered, the current loop ends and the corresponding else statement is not executed.

If a break statement is used in a nested loop, the loop at the layer where break is located terminates and the next line of code starts to be executed.

A continue statement is used to tell Python to skip the remaining statements of the current loop and continue the next loop.

Both break and continue statements can be used in while and for loops.

2.4 Functions and Object-oriented Programming

2.4.1 Functions

A function is a code segment that is organized and can be reused to implement a single or associated functions.

Functions can improve the modularity of applications and code reusability.

Python provides many built-in functions such as print(). You can also customize functions.

2.4.2 Function Definition and Calling

Definition:

In Python, the keyword def is used to mark the start of a function, followed by the function name and required parameters enclosed in parentheses.

Optional documentation string can be used in the first line of the function body to describe the function.

The function content starts with a colon (:) and is indented.

The return statement indicates the end of a function and is used to return the execution result of a function.

You must use parentheses to call functions with required parameters enclosed in the parentheses.

def function(param): # Define functions and required parameters.

'''Description documentation''' # Function description.

Function body # Content to be executed by the function.

function(param) # Call the function.

2.4.3 Function Return Values

Functions can be classified into functions with return values and those without return values.

Without a return value:

If the function body does not contain a return statement, the function returns **None**.

With a return value:

If the function body contains a return statement, the corresponding expression or value is returned.

Python can return multiple values upon a function call. By default, the return value is a tuple.

2.4.4 Function Arguments

Function arguments can be classified into the following types:

Required arguments: must be passed to a function in correct positional order, and the number of arguments in the function call should match exactly with the function definition.

Keyword arguments: During a function call, the equal sign (=) is used to assign values to passed arguments.

Default arguments: A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

Variable-length arguments: You may need to process a function for more arguments than you specified while defining the function. These arguments are called variable-length arguments and are not named in the function definition.* *args and **kwargs: The variable args with the asterisk (*) stores all unnamed variable arguments. The variable args is a tuple. The variable kwargs with the two asterisks (**) stores named arguments, such as key=value. The variable kwargs is a dictionary.

Argument positions: def func (Required arguments, Keyword arguments, Default arguments, Variable-length arguments)

2.4.5 Anonymous Functions

In addition to def, Python provides lambda to create anonymous functions. Compared with common functions, anonymous functions have the following features:

lambda is only an expression, and the function body is much simpler than def.

lambda is not a code block. Only limited logic can be encapsulated in the lambda expression.

The lambda functions have their own namespace and cannot access variables other than those in their parameter list and those in the global namespace.

An anonymous function can be defined as follows:

lambda x:x+1

2.4.6 Object-oriented and Procedure-oriented Processes

Object-oriented and procedure-oriented processes are commonly used in programming.

Object-oriented: An object is basically a self-contained entity that accumulates both data and procedures to manipulate the data. A computer program is considered as a set of objects. Each object can receive and process messages from other objects. The execution of a computer program is to transmit a series of messages between objects.

Procedure-oriented: A computer program is considered as a set of commands, that is, a set of functions that are executed in sequence. In order to simplify the program design, the procedure-oriented approach divides functions into sub-functions, that is, the system complexity is reduced by dividing a large block function into smaller blocks.

2.4.7 Advantages of Object-oriented Process

Improves code reusability.

Makes coding more flexible and improves code maintainability.

Improves program scalability.

Improves development efficiency.

2.4.8 Terminologies in Object-oriented Process

Class: A class, like a blueprint, is a code block used to create an object. It describes object features and attributes, how to use an object to complete tasks, and how an object responds to events.

Object: An object is an instance of a class. An object is usually created by calling a constructor in the class.

Method: A method is a function defined in a class. Generally, a method describes an operation that an object can perform.

Attribute: An attribute is a variable defined in a class. The attributes of a class highlight the property or status of an object.

Encapsulation: Encapsulation is to integrate methods, attributes, and events into a unified class and shields the details of the class from users.

Inheritance: Inheritance is a method to create a class. Based on the existing class (inherited class), a new class is derived, which is called a child class. The inherited class is called parent class, base class, or superclass.

Polymorphism: A function may have different implementations for different objects.

2.4.9 Object-oriented Process in Python

Python is an object-oriented programming language with built-in object-oriented features.

In Python, everything is an object.

Python supports multiple inheritance, where a class can have multiple parent classes.

You can create a class as follows:

class class name (parent class): # class keyword to declare the class. Multiple parent classes are supported. By default, the classes inherit from the object class.

'''Class description document'''

...Class body...

2.4.10 Privatization of Classes in Python

By default, the attributes in Python are public, and the module where the class resides and other modules that have imported the class can be accessed. If you want to restrict the access to or the inheritance from some attributes in a class, you can make them private.

Making modules private: You can prefix an underscore to an attribute or method. You need to prevent the module attributes from being loaded using **from mymodule import ***. The module attributes can be used only in this module.

Full privatization: Only you can access the attributes. Prefix double underscores to a method or attribute. (Python does not have mechanism for full privatization. In this process, the attribute or method name is changed to _class name.__attribute/method.)

2.4.11 Programming Paradigms

Common programming paradigms used in Python are process-oriented programming, object-oriented programming, and functional programming.

Functional programming: An operation process is abstracted as a series of nested function invokings, which have the following features:

All functions return a new value without depending on or changing external data.

A function can assign a value to a variable as a parameter or return value of another function.

Functional programming is flexible, and code can be written very close to natural language.

A function written in a pure functional programming language has no variable. Therefore, any function whose input is definite and output is definite is called a pure function without side effects.

2.5 Standard Libraries

2.5.1 Python Standard Libraries – sys

The sys module is responsible for the interaction between the program and Python interpreter, and provides a series of functions and variables to control the Python running environment.

Common attributes and methods:

sys.argv: returns command-line arguments.

sys.exit(): allows the developer to exit from Python.

sys.path(): returns the search paths of Python modules.

sys.platform: returns the system running platform.

sys.stdin/stdout/stderr: standard input/standard output/errors

2.5.2 Python Standard Libraries – os

The os module is responsible for the interaction between the program and operating system, and provides interfaces for accessing the operating system.

Common methods and attributes:

os.path.basename(): returns the base name in a specified path.

os.path.dirname(): returns the directory name in a specified path.

os.environ: contains the mapping information of environment variables. For example, os.environ["HOMEPATH"] obtains the value of environment variable HOMEPATH.

os.chdir(dir): changes the current working directory. For example, os.chdir("d:\\outlook") changes the current working directory to d:\\outlook.

os.getcwd(): returns the current directory.

2.5.3 Python Standard Libraries – time

The time module is an important module in Python for processing time. It contains various time-related methods.

Common methods:

time.sleep(secs): halts execution of a thread for given time.

time.strftime(format[, t]): converts struct_time (indicating the current time by default) to a string in a format specified by the format parameter.

time.time(): returns the timestamp of the current time.

time.localtime([secs]): converts a timestamp to struct_time of the current time zone.

2.6 I/O Operations

2.6.1 File Read and Write

Python has built-in functions for reading and writing files.

The open() function returns a file object. Generally, parameters filename, mode, and encoding are required.

filename: name of the file to be opened.

mode: file opening mode.

encoding: encoding format of the file to be opened. The default value is utf8.

Example: f = open("file_name","r",encoding="utf8") # Open the file whose name is file_name in read-only mode. The encoding format is utf8.

Use the f.close() function to close the file after the operation is complete.

2.6.2 File Opening Modes

Table 2-7 File opening modes

Access Mode	Description
r	Open a file in read-only mode. The pointer will be placed at the beginning of the file. This is the default mode.
w	Open a file in write-only mode. If the file already exists, the content in the file will be overwritten. If the file does not exist, a new file will be created.
a	Open a file for appending new content to it. If the file already exists, the file pointer is placed at the end of the file. New content will be written after the existing content. If the file does not exist, a new file will be created and content will be written to it.
rb	Open a file for reading in binary mode. The pointer will be placed at the beginning of the file. This is the default mode.
wb	Open a file for writing in binary mode. If the file already exists, the content in the file will be overwritten. If the file does not exist, a new file will be created.
ab	Open a file for appending in binary mode. If the file already exists, the file pointer is placed at the end of the file. New content will be written after the existing content. If the file does not exist, a new file will be created and content will be written to it.
r+	Open a file for reading and writing. The pointer will be placed at the beginning of the file.
w+	Open a file for reading and writing. If the file already exists, the content in the file will be overwritten. If the file does not exist, a new file will be created.
a+	Open a file for reading and writing. If the file already exists, the file pointer is placed at the end of the file. Content will be appended to the file. If the file does not exist, a new file will be created for reading and writing.
rb+	Open a file for reading and writing in binary mode. The pointer will be placed at the beginning of the file.
wb+	Open a file for reading and writing in binary mode. If the file already

Access Mode	Description
	exists, the content in the file will be overwritten. If the file does not exist, a new file will be created.
ab+	Open a file for appending in binary mode. If the file already exists, the file pointer is placed at the end of the file. If the file does not exist, a new file will be created for reading and writing.

2.6.3 Common File Handling Functions

f.write(str): writes contents of string to the file.

f.read([size]): reads data. size indicates the number of bytes to be read. If size is not specified, all bytes are read.

f.readline(): reads a line from a file and returns an empty string if the end of the file is reached; f.readlines(): reads all lines of a file and returns a list, on which each element indicates a line of data and \n is included.

f.tell(): returns the current position of the file read pointer.

f.seek(off, where): moves the file read or write pointer to a specific position. off indicates the offset. A positive offset will move the pointer forwards and a negative offset will move the pointer backwards. where indicates where to start from. Value 0 indicates the beginning of the file, value 1 indicates the current position, and value 2 indicates the end of the file.

f.flush(): updates the cache.

f.close(): closes the file.

2.6.4 Context Managers

A context manager is used to execute the preprocessing and cleanup operations as a pair, with a block of code in between.

The __enter__ and __exit__ methods of a context manager execute the preprocessing and cleanup operations, respectively.

Open a file using a context manager.

```
with open(file_name, mode,encoding) as f:
        # File handling statement.
```

A file will be automatically closed after it is handled by using a context manager.

2.7 Modules and Exceptions

2.7.1 Modules

As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use a handy function in several programs without copying its definition into each program. To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module.

A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended. Within a module, the module name (a string) can be obtained based on the value of the global variable __name__.

2.7.2 Exceptions

In most cases, there are two distinguishable kinds of errors: syntax errors and exceptions.

Syntax errors: Errors occur when you write the code and before you execute the code.

Exceptions: Exceptions occur when you attempt to execute the code. For example, an exception occurs when a divisor is 0.

Common Python exceptions are as follows:

ZeroDivisionError: division or modulo by zero (all data types)

OSError: operating system error

SyntaxError: syntax error

IndentationError: indentation error

StopIteration: end of iteration

2.7.3 Exception Handling

The try and except keywords are used for exception handling.

```
try:
    a=1/0
except Exception as e:
print("The exception is captured.")
```

User-defined exception: You can define an exception class that is inherited from the Error or Exception class. You can use the raise statement to raise the exception.

3 Advanced Programming

3.1 Database Programming

3.1.1 Database Programming

Database: A warehouse that organizes, stores, and manages data based on data structures. Users can add, delete, query, and modify data in a database.

Database technologies are widely used and the development of various database applications has become an important aspect of computer science.

The Python standard for database interfaces is the Python DB-API, which is used by Python's database interfaces.

The Python DB-API supports various databases, such as MySQL, PostgreSQL, and Oracle.

3.1.2 MySQL Operations

Import the required database: import pymysql

Enable the database connection: db = pymysql.connect("localhost", "root", "mysql", "my_database", charset='utf8')

In the preceding statement, localhost indicates a local connection, and you can change it to the IP address of the database; root and mysql indicate the account name and password; my_database indicates the name of the connected database; charset='utf8' indicates that the data encoding format is UTF-8.

Obtain the operation cursor: cursor = db.cursor()

Execute SQL statements: cursor.execute(sql)

Disable the database connection: db.close()

3.2 Multitasking

3.2.1 Multitasking

The operating system can run multiple programs at the same time.

Execution modes of multiple tasks:

- Concurrency: Multiple tasks are executed alternately.
- Parallel: Multiple tasks are executed at the same time.

Implementation of multiple tasks:

- Thread
- Process
- Coroutine

3.2.2 Thread

A thread is the minimum execution unit of the operating system.

All threads in a process share global variables.

Because of the global interpreter lock (GIL), multithreading cannot be implemented in Python.

GIL is a lock used for data integrity protection and status synchronization among threads.

3.2.3 Thread Synchronization

If multiple threads perform operations on the same global variable at the same time, resource contention occurs, causing data errors. To solve this problem, thread synchronization is required.

Thread synchronization indicates that multiple threads are executed in sequence. To implement thread synchronization, a lock mechanism needs to be introduced.

Mutex lock: When a thread is executed, a mutex lock is added to the resource so that other threads cannot operate the resource. Only after the thread releases and unlocks the resource, other threads can operate the resource.

Deadlock: A deadlock occurs when a process enters a waiting state because a requested resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process.

3.2.4 Process

A process is the minimum resource allocation unit of the operating system.

A program has at least one process, and a process has at least one thread.

Data is shared among threads in the same process.

Each process has independent memory space.

3.3 Magic Methods

There are magic methods in Python that can be used to enhance your class functionality, which start and end with two underscores (_).

Common magic methods:

__init__: Defines the initial attributes of an object when an object is initialized.

__str__: Returns the string representation of an object. The goal is to be readable.

__repr__: Returns the string representation of an object. The goal is to be unambiguous.

__getattr__: Obtains an attribute. It is invoked only when the attribute is not found.

__setattr__: Sets an attribute.

__iter__: Creates an iterator.

3.4 Higher-Order functions

zip([iterable1, iterable2 ...]): Aggregates elements from each of the given iterable objects into a tuple and returns a list of tuples. (If the iterable objects are of different lengths, the returned list is as long as the smallest object.)

print(*zip([1,2,3],["a","b","c"],["A","B","C"]))

Output: (1, 'a', 'A') (2, 'b', 'B') (3, 'c', 'C')

map(function, iterable, ...): Applies the given function to each item in a sequence.

print(*map(lambda x:x*x, [1,2,3]))

Output: 1 4 9

filter(function, iterable): Filters the given iterable object based on the given function.

print(*filter(lambda x: x%2==1, [1,2,3]))

Output: 1 3

sorted(iterable[, cmp[, key[, reverse]]]): Sorts iterable objects. (You can specify the element key and function cmp for sorting. You can also specify the sorting order: reverse = True indicates the descending order and reverse = False indicating the ascending order. The ascending order is used by default.)

sorted([('b',2),('a',1),('c',3),('d',4)], key=lambda x:x[1])

Output: [('a', 1), ('b', 2), ('c', 3), ('d', 4)]

3.5 Regular Expression

3.5.1 Regular Expression

Regular expressions are an important part of many programming languages. A regular expression is a string of special characters that describe the rules for matching a series of characters.

Regular expressions provide the basis for advanced text pattern matching and extraction, and/or text-based search and replacement.

The re module is used in Python to implement regular expressions.

3.5.2 Regular Expression Execution Process

The regular expression matching process is as follows: Match the characters in the text with the regular expression in sequence. If all characters can be matched, the matching is successful. If any character fails to be matched, the matching fails.

3.5.3 Common Matching Methods of the re Module

Table 3-1 Common matching methods of the re module

Function/Method	Description	Examples	res.group()/res
compile(pattern,flag=0)	Compiles the regular expression pattern with any optional flag and returns the regular object.	res = re.compile(".*") print res.search("abcd").group()	abcd
match(pattern,string,flag=0)	Checks for a match from the beginning of a string.	res = re.match(".*","abcdxxxx")	abcd
search(pattern,string,flag=0)	Checks for a match anywhere in the given string and returns the result if found.	res = re.search(".*", "xxxabcdxx")	abcd
findall(pattern,string,flag=0)	Searches for all regular expression patterns in a string and returns a list.	res = re.findall("a", "abdadafaf")	['a','a','a','a']
split(pattern,string,max=0)	Splits a string into lists based on the regular pattern.	re.split(",","li,yang,zhao")	['li','yang','zhao']
sub(pattern,repl,string,count=0)	Replaces the regular expression in a string by repl.	res = re.sub(",","-","l,y,z")	l-y-z

3.5.4 Common Methods for Match Object Instances

Table 3-2 Common methods for match object instances

Function/Method	Description	Examples	Result
group(num=0)	Returns the entire match object, or a specific	print(re.match(".*", "abcdxxxx").group())	abcdxxxx

Function/Method	Description	Examples	Result
	subgroup whose number is num.		
groups(default=None)	Returns a tuple containing all subgroups.	print(re.search("(\w\w\w)-(\d\d\d)","abc-123").groups())	('abc', '123')
groupdict(default=None)	Returns a dictionary containing all named subgroups of the match, keyed by the subgroup name.	res = re.search("(?P<lamb>\w\w\w)-(?P<num>\d\d\d)","abc-123") Print(str(res.groupdict()))	{'lamb':'abc', 'num': '123'}
re.i,re.IGNORECASE	The value is case insensitive.	res =re.search("abc","aBcxx",re.I) print(res.group())	aBc
re.M,re.MULTILINE	In this mode, ^ and $ match the start and end of the target string respectively, but not the start and end of any line within the target string.	res = re.search("^aB","aBcxx",re.M) print(res.group())	aB

3.5.5 Special Symbols and Characters

Table 3-3 Special symbols and characters

Representation	Description	Matched Expression	res.group()
re1|re 2	Matches the regular expression re1 or	res=re.search("foo|bar", "xxxfooxxx")	foo

Representation	Description	Matched Expression	res.group()
	re2.		
.	Matches any character (except \n).	res=re.search("b.b", "xxxbobxxx")	bob
^	Matches the start of a string.	res=re.search("^b.b", "bobx xx")	bob
$	Matches the end of a string.	res=re.search("b.b$","xx xbob")	bob
*	Matches a regular expression that appears zero or multiple times (matching from the start of the string).	res= re.search("bob*","bobbo") res1= re.search(".*","bobboddd")	Bobb bobboddd
+	Matches a regular expression that appears one or multiple times.	res= re.search("bob+","xxxxbobbbbbob")	bobbbb
?	Matches a regular expression that appears zero or one time.	res=re.search("bob?","bobbod")	bob
{N}	Matches a regular expression that appears N times.	res=re.search("bob{2}","bobbbbod")	bobb
{M,N}	Matches a regular expression that appears M to N times.	res=re.search("bob{2,3}","bobbbbod")	bobbb
[...]	Matches any single character from the character set.	res= re.search("[b,o,b]","xxbobxx")	b
[..X-Y..]	Matches any single character within the	res= re.search("[a-	x

Representation	Description	Matched Expression	res.group()
	range from x to y.	z]","xxbobxx")	
[^...]	Does not match any character in the string, including characters within a specific range.	res= re.search("[^a-z]","xx214bobxx") res1=re.search("[^2,x,1]","xx214bobxx")	2 4
(*|+|{})?	Matches the non-greedy version of a specific character that appears frequently or repeatedly.	res= re.search(".[+|?]?[1-9]","ds4b")	s4

3.6 Generators, Iterators, and Decorators

3.6.1 Iterators

Iterators are used to access elements of a set. An iterator is an object that remembers the positions of all elements of a set. An iterator object accesses the elements of a set from the beginning to the end. Iterators can only iterate forward, not backward.

An iterable object is an object that implements the __iter__ method that returns an iterator, or an object that defines the __getitem__ method that supports subscript indexes.

Use the isinstance(obj, Iterable) function to determine whether an object is an iterable object based on the return value.

Two basic methods of iterators:

next(): Outputs the next element of the iterator. When all data is iterated, a StopIteration exception will be thrown if the next() method is used again.

iter(): Creates an iterator object.

3.6.2 Generators

A generator is a special kind of iterator.

A generator can return one or more values each time it iterates, and it can record the current state.

Generator creation methods:

Use the yield keyword.

Use the generator expression (derivation).

The execution of a program suspends when the generator statement is encountered. The execution resumes only when the next() or send() method is used.

When the send() method is used, data is transferred.

3.6.3 Closures

A closure is an entity consisting of functions and their associated reference environments.

In Python, if an internal function references a variable in an external scope (not a global scope), the internal function is considered as a closure.

You cannot modify a local variable in an external scope in a closure.

Simple implementation of a closure:

```
def func():
    n=1
    def inner():
        return n+1
    return inner
```

3.6.4 Decorators

A decorator is essentially a Python function that adds new functions to the code without changing the code structure.

The working process of a decorator is as follows: Transfer the function to be decorated as a parameter to the decorator function (function with the same name), and return the decorated function.

Decorator is an application of closures.

Usage of decorators:

```
@ Decorator function
def func():
    pass
```

3.7 Extension

3.7.1 JSON

JavaScript Object Notation (JSON) is a lightweight data exchange format that is easy to read and write. Its form is similar to that of a dictionary in Python.

To use JSON functions in Python, you need to import the JSON library using the import json statement.

json.dumps: Encodes a Python object into a JSON string.

json.loads: Decodes an encoded JSON string into a Python object.

3.7.2 Metaclasses

In Python, a class itself is an object, and the class that creates such an object is called a metaclass.

You can use type() to create metaclasses in Python.

type(name, base, dict), in which name indicates the class name, base indicates the tuple of classes from which the current class derives (which is used in inheritance scenarios and can be left empty), and dict indicates the dictionary that contains attributes, including names and values.

Metaclasses of all classes are created using type().

Metaclasses are used to create APIs.

Everything in Python is an object, either an instance of a class or an instance of a metaclass, except the type class. The type class is its own metaclass.

3.7.3 Garbage Collection Mechanism in Python

In python, interpreters are responsible for memory management, saving developers' time and workloads.

The garbage collection mechanism in Python uses the reference counting technique to trace and collect garbage. On the basis of reference counting, you can also solve the circular reference problem that may be generated by container objects by using the mark-and-sweep method. Generation collection improves the efficiency of garbage collection at the cost of using more storage space.

4 Quiz

4.1 Short Answer Questions

1. If two variables have the same value, are their storage addresses in the computer the same?
2. What is the simplest way to delete duplicate elements from a list?
3. What if two lists take each other as the parameter of the append method? (a.append(b);b.append(a))

4.2 Multiple-Choice Questions

1. Which of the following statements is incorrect? ( )
   A. program can contain multiple processes.
   B. process can have 10 threads.
   C. program can have no process but only threads.
   D. Thread synchronization can be implemented using locks.
2. Which of the following statements about Python database programming is correct? ( )
   A. Python can operate only the MySQL database.
   B. Python 3 uses PyMySQL for database connection.
   C. Python 3 uses MySQLdb for database connection.
   D. The procedure for operating a database in Python is as follows: Enable the database connection, obtain the cursor, execute SQL statements, and disable the database connection.