计算机程序设计

刘龙军 副教授

2025年10月27日

Python 对比学
复习 C/C++ 体系
从两者对比程序设计本质

函数

Python体系 | C/C++体系

西安交通大学 XI'AN JIAOTONG UNIVERSITY

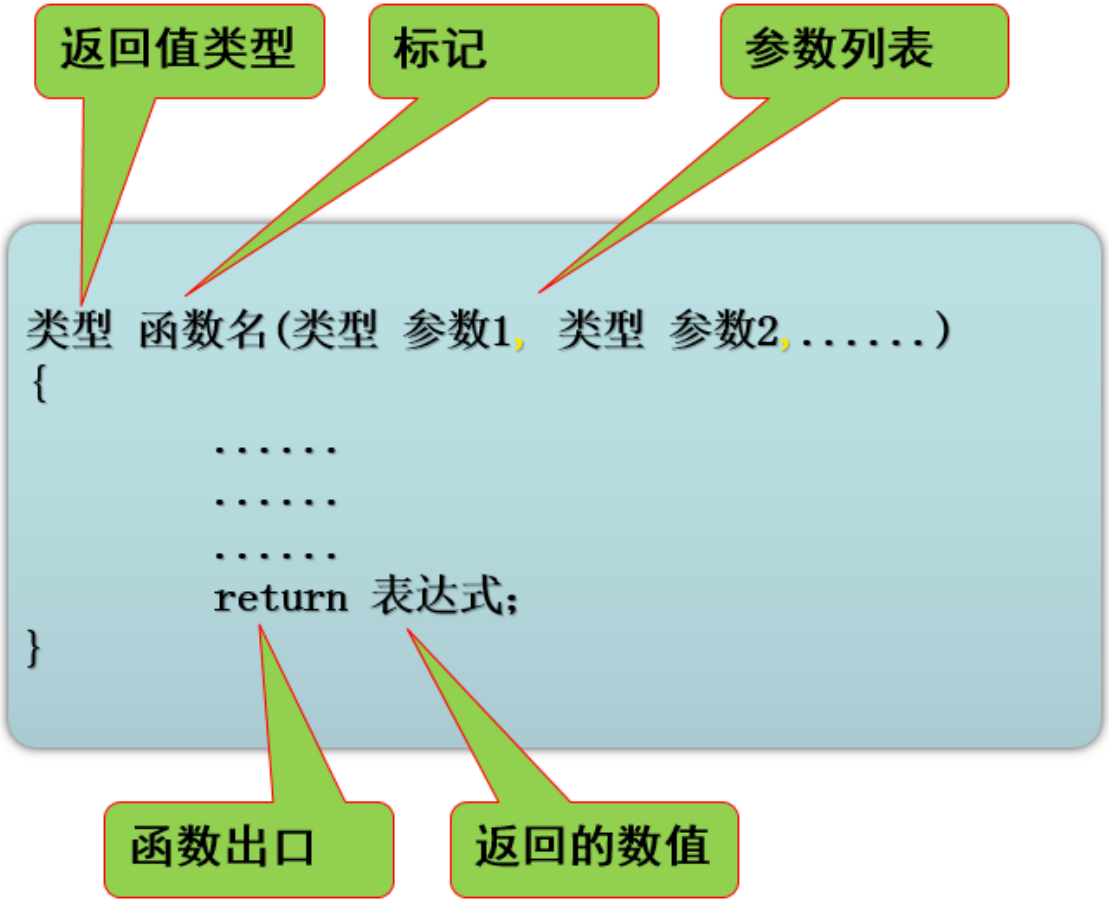
函数三要素 函数定义

```
def    <函数名>(<参数(0个或多个)>) :
           <函数体>
           return    <返回值>
```

- 在Python中，定义一个函数要使用def语句，依次写出函数名、括号、括号中的参数和冒号:，然后，**在缩进块中编写函数体**，函数的返回值用return语句返回。参数和返回值都可省略。

```
def nop():
    pass
```

- 如果想定义一个什么事也不做的空函数，可以用pass语句。



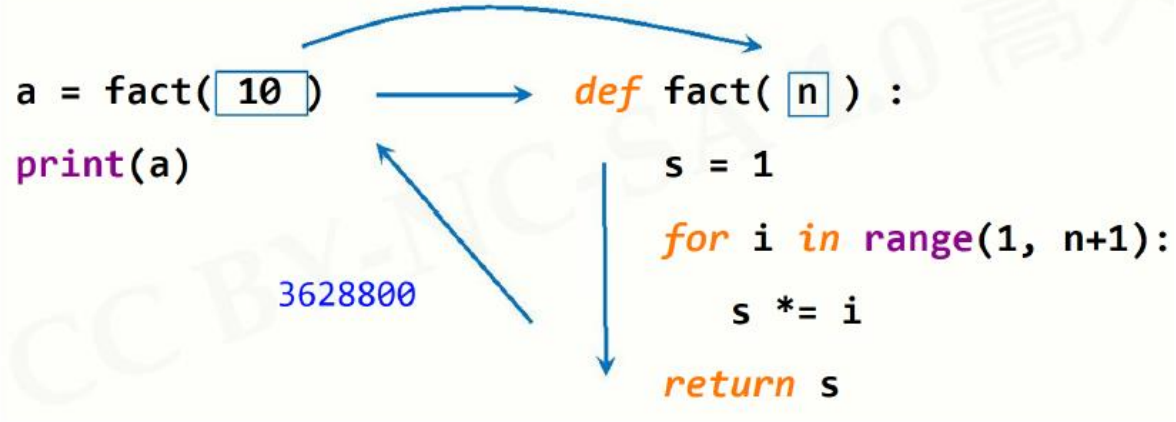
函数

Python体系 | C/C++体系

函数调用

函数名(参数表)

- 函数的调用必须出现在函数的定义之后;调用时要给出实际参数替换定义中的参数函数; 调用后得到返回值



print(fact(10))

函数名(参数1, 参数2, ...)

作为语句　　printstar( );

作为表达式　　c=max (a,b);

作为另一个函数的参数　　cout<<max (a,b);

函数

Python体系 | C/C++体系

- 函数是一段具有特定功能的、可重用的语句组。
- 两个作用：降低编程难度 和 代码复用。
- 参数和返回值都可省略。

例1：计算 n!

函数名 ↘ ↙ 参数

```
def fact(n) :
    s = 1
    for i in range(1, n+1):
        s *= i
    return s
```

← 返回值

```
>>> def hello():          #定义函数
...     print('Python你好')
...
>>> hello()               #调用函数
Python你好
```

hello()函数没有参数和返回值；
调用print()函数输出一个字符串。

- **对比找“不同”与“相同”**
- **基本相同**

函数

Python体系 | C/C++体系

参数

- 在函数定义的参数表中的参数称为形式参数，简称形参。
- 调用函数时，参数表中提供的参数称为实际参数，简称实参。
- 实参可以是常量、表达式或变量。
- 实参是常量或表达式时，直接将常量或表达式计算结果传递给形参。
- 在Python中，变量保存的是对象的引用，实参为变量时，参数传递会将实参对对象的引用赋值给形参。
- 为函数指定参数时，参数之间用逗号分隔。
- **参数的个数：**函数可以有参数，也可以没有，但必须保留括号。
- **可选参数传递**：函数定义时可以为某些参数指定默认值，构成可选参数。

对比找“不同”与“相同”

略有相同

函数

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

可选参数：

```
def   <函数名>(<非可选参数>, <可选参数>) :
      <函数体>
      return    <返回值>
```

例：计算 n!//m

```
def fact(n, m=1) :
    s = 1
    for i in range(1, n+1):
        s *= i
    return s//m
```

```
>>> fact(10)
3628800
>>> fact(10,5)
725760
```

```
>>> fact( 10,5 )          位置传递
725760
>>> fact( m=5,n=10 )      关键字传递
725760
```

函数

在定义函数时，如果在参数名前面使用星号“*”，表示形参是一个位置参数元组，可接受任意个数的参数。

调用函数时，可以不为带星号的形参提供数据。

```
>>> def add(a,*b):
...     s=a
...     for x in b:              #用循环迭代元组b中的对象
...         s+=x                 #累加
...     return s                 #返回累加结果
...
```

```
>>> add(1)              #不为带星号的形参提供数据，此时形参b为空元组
1
>>> add(1,2)            #求两个数的和，此时形参b为元组(2,)
3
>>> add(1,2,3)          #求3个数的和，此时形参b为元组(2,3)
6
>>> add(1,2,3,4,5)      #求5个数的和，此时形参b为元组(2,3,4,5)
15
```

- 对比找“不同”与“相同”
- 不相同

函数

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- 不相同

在定义函数时，带星号参数之后的参数必须通过关键字赋值传递。

```
>>> def add(a,*b,c):
...     s=a+c
...     for x in b:
...       s+=x
...     return s
...
>>> add(1,2,3)                          #形参c未使用关键字赋值传递，出错
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: add() missing 1 required keyword-only argument: 'c'
>>> add(1,2,c=3)                        #形参c使用关键字赋值传递
6
>>> add(1,c=3)                          #带星号参数可以省略
4
```

函数

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- 不相同
- C体系只能返回1个。可用指针或结构体

函数可以返回0个或多个结果，保留字*return*用来传递返回值。

```
def fact(n, m=1) :
    s = 1
    for i in range(1, n+1):
        s *= i
    return s//m, n, m
```

```
>>> fact( 10, 5 )
(725760, 10, 5)
```

元组类型

```
>>> a, b, c = fact(10, 5)
>>> print(a, b, c)
725760 10 5
```

函数

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- 不相同

同一个函数，传递的实际参数类型不同时，可获得不同的结果，体现了多态性。

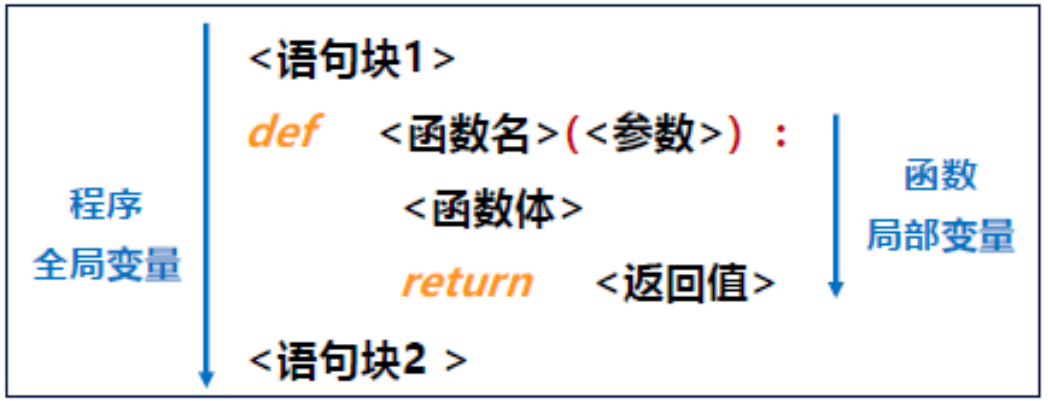
```
>>> def add(a,b):
...     return a+b                    #两个参数执行加法运算
...
>>> add(1,2)                          #执行数字加法
3
>>> add('abc','def')                  #执行字符串连接
'abcdef'
>>> add((1,2),(3,4))                  #执行元组合并
(1, 2, 3, 4)
>>> add([1,2],[3,4])                  #执行列表合并
[1, 2, 3, 4]
```

函数

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **相同**

局部变量和全局变量




```
n, s = 10, 100
def fact(n):
    s = 1
    for i in range(1, n+1):
        s *= i
    return s
print(fact(n), s)
```

n和s是全局变量

fact()函数中的n和s是局部变量

n和s是全局变量

运行结果

```
>>>
3628800 100
```

函数

- 对比找“不同”与“相同”
- 有区别

局部变量和全局变量是不同变量

- 局部变量是函数内部的占位符，与全局变量可能重名但不同
- 函数运算结束后，局部变量被释放
- 可以使用*global*保留字在函数内部使用全局变量

```
n, s = 10, 100
def fact(n) :
    global s          ← fact()函数中使用global保留字声明
                        此处s是全局变量s
    for i in range(1, n+1):
        s *= i
    return s          ← 此处s指全局变量s
print(fact(n), s)     ← 此处全局变量s被函数修改
```

运行结果

```
>>>
3628800 100
```

运行结果

```
>>>
362880000 362880000
```

函数

- 对比找“不同”与“相同”
- **基本相同**

当实参引用的是可变对象时，如列表、字典等，若在函数中修改形参，通过共享引用，实参也获得修改后的对象。

```
ls = ["F", "f"]
def func(a):
    ls.append(a)
    return
func("C")
print(ls)
```

通过使用[]真实创建了一个全局变量列表ls

此处ls是列表类型，未真实创建
则等同于全局变量

全局变量ls被修改

运行结果

```
>>>
['F', 'f', 'C']
```

函数

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

```
ls = ["F", "f"]
def func(a) :
    ls = []
    ls.append(a)
    return
func("C")
print(ls)
```

通过使用[]真实创建了一个全局变量列表ls

此处ls是列表类型，真实创建

ls是局部变量

局部变量ls被修改

运行结果

```
>>>
['F', 'f']
```

- 对比找“不同”与“相同”
- **基本相同**

当实参引用的是可变对象时，如列表、字典等，若在函数中修改形参，通过共享引用，实参也获得修改后的对象。

```
>>> def f(a):
...     a[0]='abc'              #修改列表第一个值
...
>>> x=[1,2]
>>> f(x)                       #调用函数，传递列表对象的引用
>>> x                          #变量x引用的列表对象在函数中被修改
['abc', 2]
```

函数

- 对比找“不同”与“相同”
- **基本相同**

如果不希望函数中的修改影响函数外的数据，应避免传递可变对象的引用。

如果要避免列表在函数中被修改，可使用列表的拷贝作为实参。

```
>>> def f(a):
...     a[0]='abc'              #修改列表第一个值
...
>>> x=[1,2]
>>> f(x[:])                    #传递列表的拷贝
>>> x                          #结果显示原列表不变
[1, 2]
```

函数

- 对比找“不同”与“相同”
- **基本相同**

- 也可以在函数内对列表进行拷贝，调用函数时实参仍使用变量。

```
>>> def f(a):
...     a=a[:]                  #拷贝列表
...     a[0]='abc'              #修改列表的拷贝
...
>>> x=[1,2]
>>> f(x)                       #调用函数
>>> x                          #结果显示原列表不变
[1, 2]
```

函数

- 嵌套

Python允许在函数内部定义函数。

内部函数只能在函数内部使用。

```
>>> def add(a,b):
...     def getsum(x):                    #在函数内部定义的函数，将字符串转换为Unicode码求和
...         s=0
...         for n in x:
...             s+=ord(n)
...         return s
...     return getsum(a)+getsum(b)        #调用内部定义的函数getsum
...
>>> add('12','34')                        #调用函数
202
```

- 对比找“不同”与“相同”
- 有点区别

- 对比找“不同”与“相同”
- 不相同

通过nonlocal声明，在inner函数中使用outer函数中定义的变量x，而没有重新定义一个x。

```
>>> def outer():
...     x=10
...     def inner():
...         nonlocal x
...         print("inner x : ", x)
...     inner()
...     print("outer x:", x)
...
>>> outer()
```

输出：

```
>>> inner x:10
>>> outer x:10
```

函数

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- 不相同

lambda函数也称表达式函数，与def不同，lambda的函数体只能是一个表达式。

用于定义匿名函数，适合定义简单的函数。

lambda函数定义的基本格式：lambda 参数表:表达式

可将lambda函数赋值给变量，通过变量调用函数。

可在表达式中调用其他的函数，但不能使用其他的语句。

```
>>> add=lambda a,b:a+b                #定义表达式函数，赋值给变量
>>> add(1,2)                          #函数调用格式不变
3
>>> add=lambda a,b:ord(a)+ord(b)      #在lambda表达式中调用其他的函数
>>> add('1','2')
99
```

函数

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

高阶函数是把函数作为参数的一种函数。

```
>>> def FunAdd(f,a,b):                    #定义FunAdd函数
...     return f(a)+f(b)
...
>>> def Square(x):                        #定义求平方函数
...     return x**2
...
>>> def Cube(x):                          #定义求立方函数
...     return x**3
...
>>> print(FunAdd(Square,3,5))
>>> print(FunAdd(Cube,3,5))
```

- 对比找“不同”与“相同”
- 相同

递归函数是指在函数体内调用函数本身。

例如，下面的函数fac()实现计算阶乘。

```
>>> def fac(n):                    #定义函数
...     if n==0:                    #递归调用的终止条件
...         return 1
...     else:
...         return n*fac(n-1)       #递归调用函数本身
...
>>> fac(5)
120
```

注意，递归函数必须在函数体中设置递归调用的终止条件。

如果没有设置递归调用终止条件，程序会在超过Python允许的最大递归调用深度后，产生RecursionError异常（递归调用错误）。

- 对比找“不同”与“相同”
- **基本相同**

10 函数列表

因为函数是一种对象，所以可将其作为列表元素使用，然后通过列表索引来调用函数。

```
>>> d=[lambda a,b: a+b,lambda a,b:a*b]        #使用lambda函数建立列表
>>> d[0](1,3)                                 #调用第一个函数
4
>>> d[1](1,3)                                 #调用第二个函数
3
```

函数

Python体系

也可使用def定义的函数来创建列表。

```
>>> def add(a,b):                          #定义求和函数
...     return a+b
  >>> def fac(n):                         #定义求阶乘函数
  ...     if n==0:
  ...         return 1
  ...     else:
  ...         return n*fac(n-1)

  >>> d=[add,fac]                         #建立函数列表
  >>> d[0](1,2)                           #调用求和函数
  3
  >>> d[1](5)                             #调用求阶乘函数
  120

  >>> d=(add,fac)                         #建立包含函数列表的元组对象
  >>> d[0](2,3)                           #调用求和函数
  5
  >>> d[1](5)                             #调用求阶乘函数
  120
```

C/C++体系

- 对比找“不同”与“相同”

- 不相同

Python体系 | C/C++体系

西安交通大学 XI'AN JIAOTONG UNIVERSITY

对比找“不同”与“相同”

Python还允许使用字典来建立函数映射。

```
>>> d={'求和':add,'求阶乘':fac}          #用函数add和fac建立函数映射
>>> d['求和'](1,2)                       #调用求和函数
3
>>> d['求阶乘'](5)                       #调用求阶乘函数
120
```

不相同

类

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- 相同

类提供了一种将**数据和功能**捆绑在一起的方法。

类就是对象的**属性和方法**的封装，静态的特征称为属性，动态的动作称为方法。

创建一个新类将创建一种新的对象类型，从而允许创建该类型的新实例。

每个类实例拥有附加的属性以维护其状态。

类实例还可以具有用于修改、访问其属性状态的方法。

```
class ClassName:
    #属性
    [属性定义体]
    #方法
    [方法定义体]
```

类的定义以关键字class开始，类名以大写字母开头，类名后面紧跟冒号

静态

动态

类

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

```
class Person:                    ←—— Python的类名约定以大写字母开头
    # 属性                       ←—— 静态
    skincolor = "yellow"
    high = 168
    weight = 65
    # 方法                       ←—— 动态
    def goroad(self):
        print("人走路动作的测试.....")
    def sleep(self):
        print("睡觉，晚安！")
```

类

- 对比找“不同”与“相同”
- **基本相同(构造函数)**

- 类定义好之后，就可将类实例化为对象。类实例化对象： 对象名 = 类名（）

```
>>> p = Person()
```

- 类可以定义一个特殊方法 __init__(self)，创建具有定制为特定初始状态的实例的对象 。
- 实例化对象时，这个方法就会在对象被创建的时候自动调用。

```
>>> class Bear:
        def __init__(self, name):
            self.name = name
        def kill(self):
            print("%s,是保护动物，不能杀..."% self.name)
```

运行结果如下：

```
>>> a = Bear('狗熊')
>>> a.kill()
狗熊,是保护动物，不能杀...
```

类

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

- 可以把传入的参数设置为默认参数，在实例化的时候不传入参数系统也不会报错。

```
>>> class Bear:
        def __init__(self, name= “默认的熊” ):
            self.name = name
        def kill(self):
            print( “%s，是保护动物，不能杀..."% self.name)
```

运行结果如下：

```
>>> b = Bear()
>>> b.kill()
默认的熊，是保护动物，不能杀...
```

- 对比找“不同”与“相同”
- 有区别

2.3 类的访问权限

- 在Python中，默认情况下对象的属性和方法都是公开的、公有的，通过点(.)操作符来访问：

```
>>> class Bear:
        name = “默认的熊”
>>> d = Bear()
>>> d.name
```

输出结果：

默认的熊

类

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

- 为了实现类似于私有变量的特征，Python内部采用了name mangling的技术（名字重整或名字改变），在变量名或函数名前加上两个下划线（“__”），这个函数或变量就变为私有了。

```
>>> class Bear:
        __name = “默认的熊”
        def getname(self):
            return self.__name
>>> d = Bear()
>>> d.getname()
```

输出结果：

默认的熊

- 对比找“不同”与“相同”
- 不相同（friend）

- 把双下划线“__”开头的变量名，改为了：单下划线“_”类名+双下划线“__”变量名，即 _类名__变量名，可以通过以下的访问方式来访问类的私有变量。

```
>>> class Bear:
        __name = “默认的熊”
        def getname(self):
            return self.__name
>>> d = Bear()
>>> d.getname()
>>> d._Bear__name
```

输出结果：

默认的熊

默认的熊

类

- 对比找“不同”与“相同”
- **基本相同**

（1）封装

列表(list)是Python的一个序列对象，要对列表进行调整，代码如下：

```
>>> list1 = ['K','J','L','Q','M' ]
>>> list1.sort()
>>> list1
['J', 'K', 'L', 'M', 'Q' ]
```

Python的列表就是对象，它提供了若干种方法来供用户根据需求调整整个列表，但是用户不知道列表对象里面的方法是如何实现的，也不知道列表对象里面有哪些变量，这就是封装。它封装起来，只给用户需要的方法的名字，然后调用这个名字，知道它可以实现就可以了，然而它没有具体告诉用户是怎么实现的。

- 对比找“不同”与“相同”
- **基本相同**

(2) 多态

- 不同对象调用了同名方法(函数)，实现的结果不一样，这就是多态。
- 类Test_X()的实例对象为：x，类Test_Y()的实例对象为：y，对象x和对象y分别调用了同名函数func()，分别运行得到了不同的结果：测试X...和测试Y...

```
>>> class Test_X:
        def func(self):
            print("测试X...")
>>> class Test_Y:
        def func(self):
            print("测试Y...")
```

运行结果如下：

```
>>> x = Test_X()
>>> y = Test_Y()
>>> x.func()
测试X...
>>> y.func()
测试Y...
```

类

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

(3) 继承

- 子类自动共享父类的数据和方法的机制。
- 语法格式如下：

```
Class ClassName(BaseClassName):
        ......
```

- ClassName：是子类的名称，第一个字母必须大写。
- BaseClassName：是父类的名称。
- 被继承的类我们称为基类、父类或超类，而继承者我们称为子类。
- 子类可以继承父类的任何属性和方法。

```
>>> class Test_list(list):          <-- 定义了一个类Test_list(),继承list
        pass
运行结果如下：
>>> list1 = Test_list()             <-- list1为实例化
>>> list1.append('O')               <-- 调用append
>>> list1
['O']
>>> list1.append('MT')
>>> list1
['O', 'MT']
>>> list1.sort()                    <-- 调用sort
>>> list1
['MT', 'O']
```

- 对比找“不同”与“相同”
- **基本相同**

(3) 继承

- **多重继承：**同时继承多个父类的属性和方法，称为多重继承。
- 语法格式如下：

```
Class ClassName(Base1, Base2, Base3):
    ......
```

- 虽然多重继承的机制可以让子类继承多个基类的属性和方法使用起来很方便，但很容易导致代码混乱，有时候会引起不可预见的Bug，对程序而言不可预见的Bug几乎就是致命的。因此，当我们不确定必须要使用“多重继承”语法的时候，尽量避免使用它。

- 对比找“不同”与“相同”
- **基本相同**

3. 模块(module)

- 写的代码保存的以.py结尾的Python文件就是一个独立的模块。
- 模块中包含变量、函数或类，也可包含其他的Python语句。
- 模块需要先导入，才能使用其中的变量、函数或者类等。
- 可使用import或from语句**导入模块**。

```
import 模块名称
import 模块名称 as 新名称

from 模块名称 import 导入对象名称
from 模块名称 import 导入对象名称 as 新名称
from 模块名称 import *
```

- 对比找“不同”与“相同”
- **基本相同**

(1) import语句

- import语句用于导入整个模块，可用as为导入的模块指定一个新名称。
- 导入模块后，使用“模块名称.对象名称”格式来引用模块中的对象。

```
>>> import math                  #导入模块
>>> math.fabs(-5)                #调用模块中的函数
5.0
>>> math.e                       #使用模块中的常量
2.718281828459045
>>> fabs(-5)                     #试图直接使用模块中的函数，出错
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'fabs' is not defined
```

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- 不相同

(1) import语句

```
>>> import math as m                #导入模块并指定新名称
>>> m.fabs(-5)                      #通过新名称调用模块函数
5.0
>>> m.e                             #通过新名称使用模块常量
2.718281828459045
```

- 对比找“不同”与“相同”
- 不相同

(2) from语句

- from语句用于导入模块中的指定对象，导入的对象可直接使用，不需要使用模块名称作为限定符。

```
>>> from math import fabs                    #从模块导入指定函数
>>> fabs(-5)
5.0
>>> from math import e                       #从模块导入指定常量
>>> e
2.718281828459045
>>> from math import fabs as f1              #导入时指定新名称
>>> f1(-10)
10.0
```

- 对比找“不同”与“相同”
- 不相同

(3) from … import *语句

- 使用星号时，可导入模块顶层的所有全局变量和函数。

```
>>> from math import *          #导入模块顶层的全局变量和函数
>>> fabs(-5)                    #直接使用导入的函数
5.0
>>> e                           #直接使用导入的常量
2.718281828459045
```

类

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- 不相同

- import和from语句在执行导入操作时，会执行导入模块中的全部语句。
- 只有执行了模块，模块中的变量和函数才会被创建，才能在当前模块中使用。
- 只有在第一次执行导入操作时，才会执行模块。
- import和from语句是隐性的赋值语句，两者的区别如下。
- **执行import语句:**
  - 会创建一个模块对象和一个与模块文件同名的变量，并建立变量和模块对象的引用。
  - 模块中的变量和函数等均作为模块对象的属性使用。
  - 再次导入时，不会改变模块对象属性的当前值。
- **执行from语句:**
  - 会同时在当前模块和被导入模块中创建同名变量，这两个变量引用同一个对象。
  - 再次导入时，会将被导入模块的变量的初始值赋值给前模块的变量。

类

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- 不相同

例：创建模块文件test.py，其代码如下：

```
x=100                                   #赋值，创建变量x
print('这是模块test.py中的输出！' )       #输出字符串
def show():                             #定义函数，执行时创建函数对象
    print('这是模块test.py中的show()函数中的输出！')
```

- 进入Python交互环境：

```
>>> import test                         #导入模块，下面的输出说明模块在导入时被执行
这是模块test.py中的输出!
>>> test.x                              #使用模块变量
100
>>> test.x=200                          #为模块变量赋值
```

- 对比找“不同”与“相同”
- 不相同

```
>>> import test        #重新导入模块
>>> test.x             #使用模块变量，输出结果显示重新导入未影响变量的值
200
>>> test.show()        #调用模块函数
这是模块test.py中的show()函数中的输出!
>>> abc=test           #将模块变量赋值给另一个变量
>>> abc.x              #使用模块变量
200
>>> abc.show()         #调用模块函数
这是模块test.py中的show()函数中的输出!
```

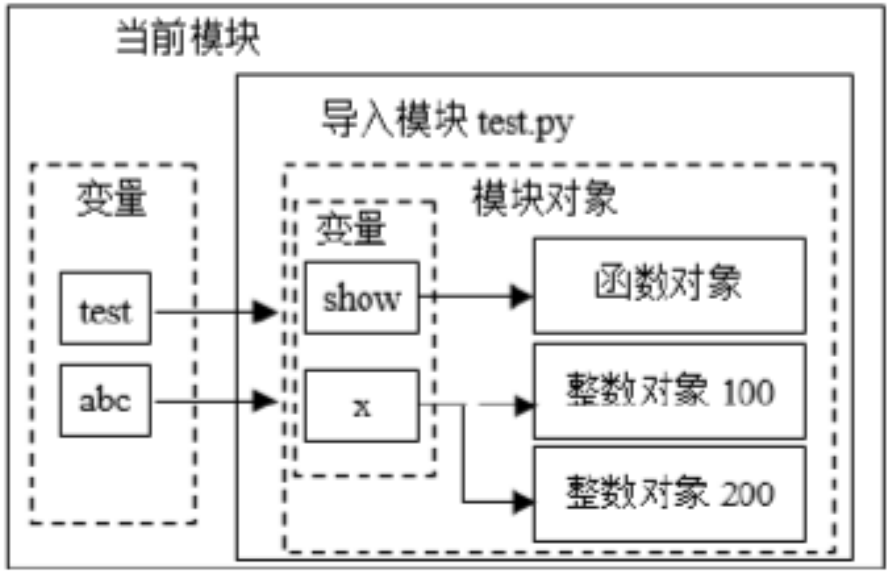


图 6-2 执行 import 导入后模块与变量的关系

- 对比找“不同”与“相同”
- 不相同

- 使用from语句导入test模块。

```
>>> from test import x,show        #导入模块的变量x、show
这是模块test.py中的输出!
>>> x                              #输出模块的变量的初始值
100
>>> show()                         #调用模块函数
这是模块test.py中的show()函数中的输出!
>>> x=200                          #这里是为当前模块的变量赋值
>>> from test import x,show        #重新导入
>>> x                              #x的值为模块的变量的初始值
100
```

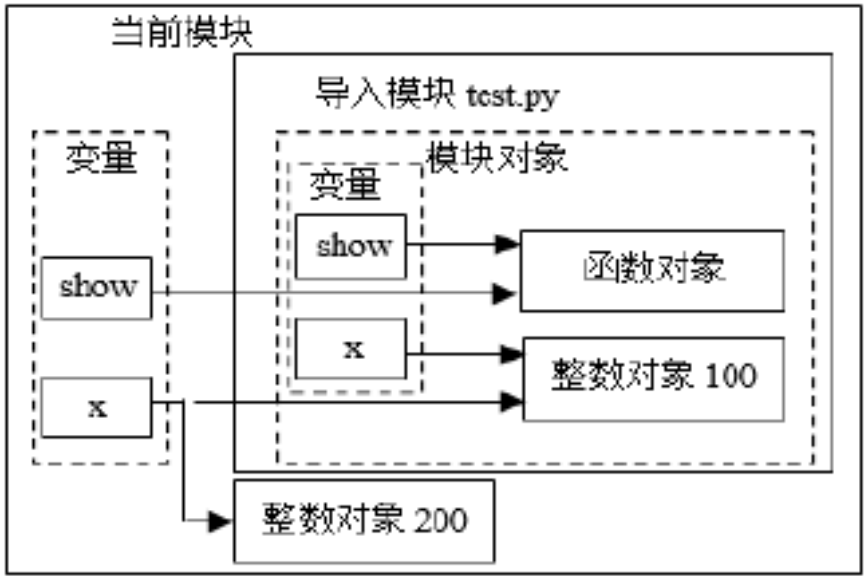


图 6-3 执行 from 导入后模块与变量的关系

对比找“不同”与“相同”

用import还是from

- 使用import导入模块时，模块的变量使用“模块名.”作为限定词，所以不存在歧义，即使与其他模块的变量同名也没有关系。
- 在使用from时，当前模块的同名变量引用了模块内部的对象，应注意引用模块变量与当前模块或其他模块的变量同名的情况。

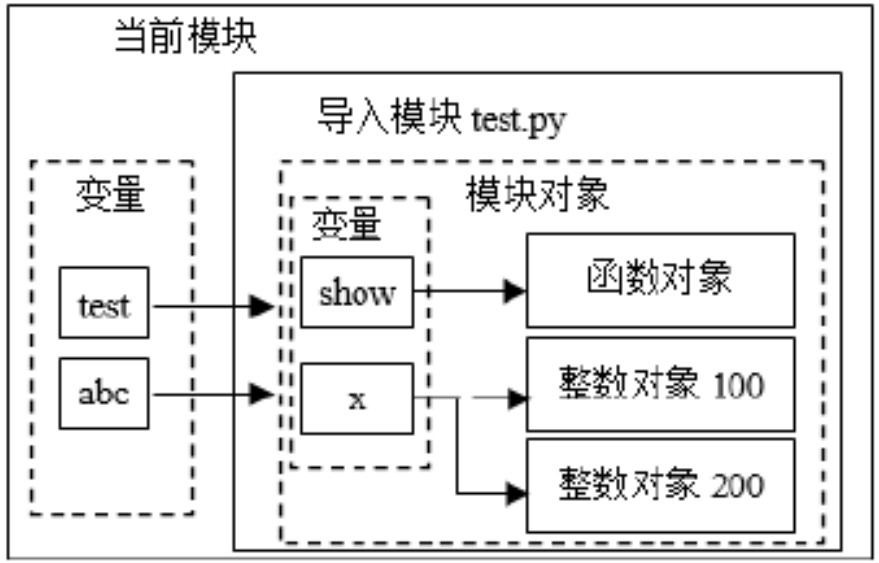


图 6-2 执行 import 导入后模块与变量的关系

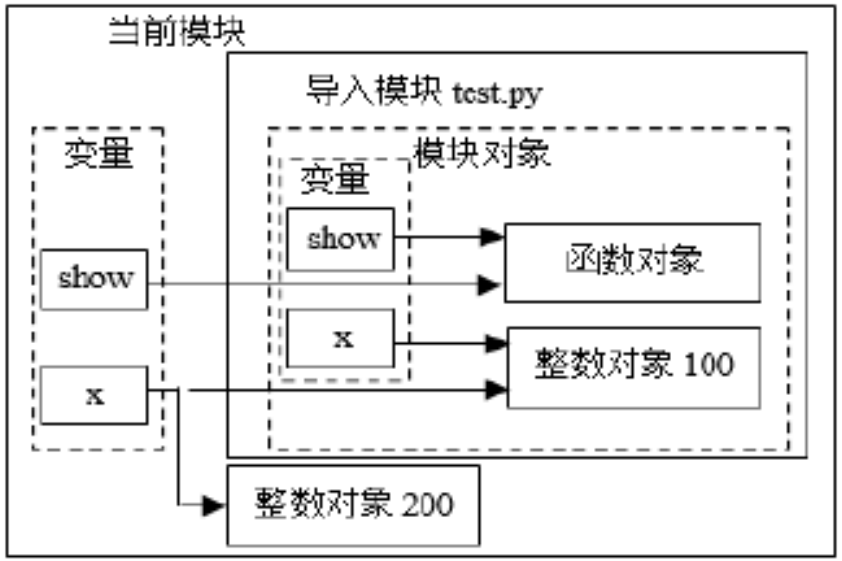


图 6-3 执行 from 导入后模块与变量的关系

- 对比找“不同”与“相同”
- 不相同

3. 6 模块搜索路径

- 通常，sys.path由4部分设置组成。

  （1）Python的当前工作目录（可用os模块中的getcwd()函数查看当前目录名称）。

  （2）操作系统的环境变量PYTHONPATH中包含的目录（如果存在）。

  （3）Python标准库目录。

  （4）任何pth文件包含的目录（如果有存在）。

- Python按照上面的顺序搜索各个目录。
- pth文件通常放在Python安装目录中，文件名可以任意，例如searchpath.pth。
- 在pth文件中，每个目录占一行，可包含多个目录。

```
C:\myapp\hello
D:\pytemp\src
```

- 对比找“不同”与“相同”
- 不相同

3. 7 嵌套导入模块

- Python允许任意层次的嵌套导入。
- 每个模块都有一个名字空间，嵌套导入意味着名字空间的嵌套。
- 在使用模块的变量名时，应依次使用模块名称作为限定符。
- 例如，有两个模块文件test.py和test2.py，下面的代码说明了嵌套导入时应如何使用模块中的变量。

```
#test.py
x=100
def show():
    print('这是模块test.py中的show()函数中的输出！')
print('载入模块test.py！')
import test2
```

```
#test2.py
x2=200
print('载入模块test2.py！')
```

- 对比找“不同”与“相同”
- 不相同

- 在交互模式下导入test.py。

```
>>> import test                          #导入模块test
载入模块test.py!
载入模块test2.py!
>>> test.x                              #使用test模块的变量
100
>>> test.show()                         #调用test模块的函数
这是模块test.py中的show()函数中的输出!
>>> test.test2.x2                       #使用嵌套导入的test2模块中的变量
200
```

- 对比找“不同”与“相同”
- 不相同

包（Package）

- 包通常是一个目录，可以使用import导入包，或者from语句来导入包中的部分模块。
- 目录下有__init__.py，和其他模块文件及子目录。
- __init__.py 可以是一个空文件，也可以包含包的初始化代码或者设置__all__列表，用于提示该目录不是普通目录。

```
>>> import mycompany.abc as mc_abc
>>> import mycompany.web.www
```

```
mycompany
├── web
│   ├── __init__.py
│   ├── utils.py
│   └── www.py
├── __init__.py
├── abc.py
└── xyz.py
```

文件

Python体系

1、文件操作（TXT）

2、CSV文件

3、JSON文件

4、XLSX文件

5、Pandas

6、图像文件

C/C++体系

- 对比找“不同”与“相同”

- 基本相同

文件

Python体系 | C/C++体系

1. 文件操作（TXT）

- 文件的类型
- 文件的打开和关闭
- 文件的读写
- 用文件存储对象
- 目录操作

- 对比找“不同”与“相同”

- 基本相同

文件

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

1.1 文件的类型

- 文件是数据存储的一种形式。
- **文件展现形态**：文本文件和二进制文件，本质上所有文件都是二进制形式存储，形式上所有文件采用两种方式展示。
- **文本文件**根据字符编码保存文本（如ASCII、UTF-8、GB2312等），**按字符读取**文件，一个字符占用一个或多个字节。整个文件可看作一个长字符串。
- **二进制文件**存储的是数据的二进制代码，即将数据在内存中的存储形式复制到文件中。二进制文件没有字符编码，文件的存储格式与用途无关，通常用于保存图片、音频和视频等数据（如png、mp3、mp4等），二进制文件通常**按字节读取**文件。
- Python根据打开模式按文本文件或二进制文件格式读写文件中的数据。

文件

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

文本文件 vs 二进制文件

- **文本形式:**

f.txt文件保存: "中国是个伟大的国家!"

```
#文本形式打开文件
tf = open("f.txt", "rt")
print(tf.readline())
tf.close()

>>>
中国是个伟大的国家!
```

- **二进制形式**

f.txt文件保存: "中国是个伟大的国家!"

```
#二进制形式打开文件
bf = open("f.txt", "rb")
print(bf.readline())
bf.close()

>>>
b'\xd6\xd0\xb9\xfa\xca\xc7\xb8\xf6\xce\xb0
\xb4\xf3\xb5\xc4\xb9\xfa\xbc\xd2\xa3\xa1'
```

文件

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

- 用Python内置的open()函数来打开文件，并返回关联的文件句柄。
- open()函数基本格式：

```
myfile = open(filename[,mode])
```

  - myfile为引用文件对象的变量
  - filename为文件名字符串
  - mode为文件读写模式
- 在文件名中可包含相对或绝对路径，省略路径时，Python在当前工作目录中搜索文件。
- 在系统命令提示符窗口中进入交互环境或执行Python程序时，当前目录为Python的当前工作目录。

文件

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

myfile = open(filename[,mode])

打开模式

- 打开文件后，Python用一个文件指针记录当前读写位置。
- 以“r”模式打开文件时，文件指针指向文件开头。
- 以“w”或“a”模式打开文件时，文件指针指向文件末尾。
- 始终在文件指针的位置读写数据，读取或写入数据后，根据数据长度向后移动文件指针。

优美胜于丑陋
明了胜于隐晦
简洁胜于复杂

文件的打开模式	描述
'r'	只读模式，默认值，如果文件不存在，返回FileNotFoundError
'w'	覆盖写模式，文件不存在则创建，存在则完全覆盖
'x'	创建写模式，文件不存在则创建，存在则返回FileExistsError
'a'	追加写模式，文件不存在则创建，存在则在文件最后追加内容
'b'	二进制文件模式
't'	文本文件模式，默认值
'+'	与r/w/x/a一同使用，在原功能基础上增加同时读写功能

文件

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

f = open("f.txt")	文本形式、只读模式、默认值，等同于“rt”
f = open("f.txt", "rt")	文本形式、只读模式
f = open("f.txt", "w")	文本形式、覆盖写模式，等同于“wt”
f = open("f.txt", "r+")	文本形式、覆盖写模式+ 读文件
f = open("f.txt", "a+")	文本形式、追加写模式+ 读文件
f = open("f.txt", "x")	文本形式、创建写模式
f = open("f.txt", "b")	二进制形式、只读模式
f = open("f.txt", "rb")	二进制形式、只读模式，等同于“b”
f = open("f.txt", "rb+")	二进制形式、覆盖写模式+ 读文件
f = open("f.txt", "wb")	二进制形式、覆盖写模式
f = open("f.txt", “ab+")	二进制形式、追加写模式+ 读文件

文件

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

文件的关闭

- close()方法用于关闭文件： myfile.close()
- 通常，Python会使用内存缓冲区缓存文件数据。
- 关闭文件时，Python将缓冲的数据写入文件，然后关闭文件，释放对文件的引用。
- 程序结束时，Python可自动关闭未使用的文件。
- flush()方法可将缓冲区的内容写入文件，但不关闭文件： myfile.flush()
- 为了确保打开的文件对象正常关闭，一般结合异常机制的finally或者with关键字。

```
try:
    tf = open("f.txt", "rt")
    print(tf.readline())
finally:
    if tf:
        tf.close()
```

```
with open("f.txt", "rt") as tf:
    print(tf.readline())
```

- 对比找“不同”与“相同”
- **基本相同**

1.3 文件的读写

- **read()**：将从文件指针位置开始到文件末尾的内容作为一个字符串返回。
- **read(n)**：将从文件指针位置开始的n个字符作为一个字符串返回。
- **readline()**：将从文件指针位置开始到下一个换行符号的内容作为一个字符串返回，读取内容包含换行符号。
- **readlines()**：将从文件指针位置开始到文件末尾的内容作为一个列表返回，每一行的字符串作为一个列表元素。
- 文本文件按字符读取数据，如果文件包含Unicode字符，Python会自动进行转换。
- 文本文件中每行末尾以回车换行符号结束。在读取的字符串中，Python用“\n”代替回车换行符号。二进制文件读取的回车换行符号为“\r\n”。

文件

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

```
# 一次读入，统一处理
fname = input("请输入要打开的文件名称:")
fo = open(fname,"r")
txt = fo.read()
#对全文txt进行处理
fo.close()
```

```
# 按数量读入，逐步处理
fname = input("请输入要打开的文件名称:")
fo = open(fname,"r")
txt = fo.read(2)
while txt != "":
    #对txt进行处理
    txt = fo.read(2)
fo.close()
```

文件

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

```
# 一次读入，分行处理
fname = input("请输入要打开的文件名称:")
fo = open(fname,"r")
for line in fo.readlines():
    print(line)
fo.close()
```

```
# 分行读入，逐行处理
fname = input("请输入要打开的文件名称:")
fo = open(fname,"r")
line = fo.readline()
while line:
    print(line)
    line = fo.readline()
fo.close()
```

```
for line in fo:
    print(line)
```

- 对比找“不同”与“相同”
- **基本相同**

1.3 文件的读写

- **write(xstring)**：在文件指针位置写入字符串或字节流，返回写入的字符个数。
- **writelines(xlist)**：将列表中的数据合并为一个字符串写入到文件指针位置，返回写入的字符个数。
- **seek(n[, whence])**：将文件指针移动到第n个字节。whence：0表示指向文件开头，1表示当前位置，2表示文件结尾。用“t”模式打开文件，只允许从文件头开始计算相对位置。
- **tell()**：返回文件指针当前位置。

文件

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

写文件

```
# 写入一个字符串列表
fo = open("output.txt","w+")
ls = ["中国", "法国", "美国"]
fo.writelines(ls)
for line in fo:
    print(line)
fo.close()
>>> (没有任何输出)
```

```
# 写入一个字符串列表
fo = open("output.txt","w+")
ls = ["中国", "法国", "美国"]
fo.writelines(ls)
fo.seek(0)
for line in fo:
    print(line)
fo.close()
>>>
中国法国美国
```

文件

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- **基本相同**

例1： **以“r”模式打开文件。**

- code.txt:

one第一行
two第二行
three第三行

```
>>> myfile=open('code.txt')          #以默认只读方式打开文件
>>>x=myfile.read()                   #读文件全部内容到字符串
>>>x                                 #每行末尾的换行符号在字符串中为“\n”
'one第一行\ntwo第二行\nthree第三行'
>>> print(x)                         #打印格式与原文件完全一致
one第一行
two第二行
three第三行
>>> myfile.read()                    #文件指针已指向文件末尾，返回空字符串
''
```

- 对比找“不同”与“相同” **基本相同**

- code.txt:

one第一行
two第二行
three第三行

```
>>> myfile.seek(0)                    #将文件指针移动到文件开头
0
>>> myfile.read(5)   #读5个字符
'one第一'
>>> myfile.tell()                     #返回文件指针的当前位置
9
>>> myfile.readline()                 #读取从文件指针当前位置到当前行末尾的字符串
'行\n'
>>> myfile.readline()                 #读下一行
'two第二行\n'
```

文件

西安交通大学 XI'AN JIAOTONG UNIVERSITY

- 对比找“不同”与“相同” **基本相同**

```
>>> myfile.seek(0)
0
>>> myfile.readlines()                    #读文件全部内容到列表
['one第一行\n', 'two第二行\n', 'three第三行']
>>> myfile.seek(0)
0
>>> for x in myfile:print(x)              #以迭代方式读文件
...
one第一行

                                          #请思考两行数据之间为什么有一个空行?
two第二行

three第三行
>>> myfile.close()                        #关闭文件
```

- code.txt:

```
one第一行
two第二行
three第三行
```

西安交通大学 XI'AN JIAOTONG UNIVERSITY

■ 对比找“不同”与“相同” 基本相同

例2： 以“r+”模式打开文件。

- 可从文件读取数据，也可向文件写入数据。
- 刚打开文件时，文件指针指向文件开头。
- 在执行读操作后执行写操作时，不管文件指针位置在哪里，都将数据写入文件末尾。
- 要在特定位置写入数据，需要先执行seek()函数指定文件指针位置，然后写入数据。

```
>>> myfile=open('code.txt','r+')
>>> myfile.write('oneabc')          #写入字符串，此时写入到文件开头，覆盖原数据
6
>>> myfile.seek(0)                  #定位文件指针到文件开头
0
```

- code.txt:

```
one第一行
two第二行
three第三行
```

- 对比找“不同”与“相同” **基本相同**

```
>>> myfile.read()                    #读取全部数据
'oneabc一行\ntwo第二行\nthree第三行'
>>> myfile.seek(6)                   #将文件指针指向第7个字节（位置为6）
6
>>> myfile.write('1234567')          #写入数据，会覆盖原第一行末尾的换行符号
7
>>> myfile.seek(0)                   #定位文件指针到文件开头
0
>>> myfile.read()                    #读取数据，查看前面写入的数据
'oneabc1234567two第二行\nthree第三行'
>>> myfile.seek(0)                   #定位文件指针到文件开头
0
```

- code.txt:

```
oneabc一行
two第二行
three第三行
```

文件

西安交通大学 XI'AN JIAOTONG UNIVERSITY

- 对比找“不同”与“相同” **基本相同**

```
>>> myfile.read(5)          #读取5个字符
'oneab'
>>> myfile.tell()           #查看文件指针位置
5
>>> myfile.write('xxx')     #写入数据，读取数据后立即写入，数据写入文件末尾
3
>>> myfile.seek(0)          #定位文件指针到文件开头
0
>>> myfile.read()           #读取数据，查看前面写入的数据，“xxx”在文件末尾
'oneabc1234567第二行\nthree第三行xxx'
>>> myfile.close()          #关闭文件
```

- code.txt:

oneabc1234567two第二行
three第三行

■ 对比找“不同”与“相同” 基本相同

例3：以“w”模式打开文件。

- 以“w”模式打开文件时，只能向文件写入数据。如果存在同名文件，原来的文件会被覆盖。

```
>>> myfile=open('code_w.txt','w')
>>> myfile.write('one\n')                    #将字符串写入文件
4
>>> myfile.writelines(['1','2','abc'])       #将列表写入文件，列表对象必须都是字符串
>>> myfile.close()                           #关闭文件
>>> myfile=open('code_w.txt' )               #重新打开文件，读取前面写入的数据
>>> myfile.read()
'one\n12abc'
>>> myfile.close()                           #关闭文件
```

- 对比找“不同”与“相同” **基本相同**

例4：以“w+”模式打开文件时，允许同时读写文件。

```
>>> myfile=open('code_w.txt','w+')
>>> myfile.read()                    #新建文件，所以其中没有数据，返回空字符串
''
>>> myfile.write('one\n')            #将字符串写入文件
4
>>> myfile.writelines(['1', '2','abc'])
>>> myfile.seek(0)                   #将文件指针移动到文件开头
0
>>> myfile.readline()                #读第1行
'one\n'
```

- code_w.txt:

```
one
12abc
```

文件

西安交通大学 XI'AN JIAOTONG UNIVERSITY

- 对比找“不同”与“相同” **基本相同**

```
>>> myfile.readline()                  #读第2行
'12abc'
>>> myfile.readline()                  #已经到文件末尾，返回空字符串
''
>>> myfile.seek(4)                     #将文件指针移动到第4个字节之后
4
>>> myfile.write('xxxxxxx')            #将字符串写入文件
7
>>> myfile.seek(0)                     #将文件指针移动到文件开头
0
>>> myfile.read()                      #读取全部数据
'one\nxxxxxxx'
>>> myfile.close()                     #关闭文件
```

- code_w.txt:

```
one
12abc
```

- code_w.txt:

```
one
12abcxxxxxxx
```

- code_w.txt:

```
one
xxxxxxx
```

■ 对比找“不同”与“相同” 基本相同

例5：以“a”模式打开文件时，只能向文件写入数据，文件打开时文件指针指向文件末尾，向文件写入的数据始终添加到文件末尾。

```
>>> myfile=open('code_w.txt','a')
>>> myfile.write('\n123456')              #将字符串写入文件
7
>>> myfile.seek(4)
4
>>> myfile.write('*****')     #虽然文件指针指向第5个字符，但仍写入文件末尾
5
>>> myfile.close()
```

- code_w.txt:

```
one
xxxxxxx
```

- code_w.txt:

```
one
xxxxxxx
123456*****
```

西安交通大学 XI'AN JIAOTONG UNIVERSITY

■ 对比找“不同”与“相同” **基本相同**

例6： “a+” 除了允许写入数据，还可以读取文件数据。

```
>>> myfile=open('code_w.txt','a+')
>>> myfile.tell()                    #查看文件指针位置，此时应为文件末尾
25
>>> myfile.write('\n新添加的数据' )    #将字符串写入文件
7
>>> myfile.seek(0)                   #将文件指针移动到文件开头
0
>>> print(myfile.read())             #打印读取的文件内容
one
xxxxxxx
123456*****
新添加的数据
```

- code_w.txt:

```
one
xxxxxxx
123456*****
```

- 对比找“不同”与“相同” **基本相同**

```
>>> myfile.seek(5)                #将文件指针移动第5个字符之后
5
>>> myfile.write('newdata' )      #将字符串写入文件
7
>>> myfile.seek(0)                #将文件指针移动到文件开头
0
>>> print(myfile.read())          #打印读取的数据，查看前面写入的“newdata”的位置
one
xxxxxxx
123456
新添加的数据newdata
>>> myfile.close()                #关闭文件
```

西安交通大学 XI'AN JIAOTONG UNIVERSITY

- 对比找“不同”与“相同” **基本相同**

例7：读写二进制文件

- 文本文件的读写方法均可用于二进制文件，区别在于：二进制文件读写的是字节流。

```
>>> myfile=open('code.txt','wb')          #创建二进制文件
>>> myfile.write('aaaaa')                 #出错，二进制文件只能写入字节流
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: a bytes-like object is required, not 'str'
>>> myfile.write(b'aaaaa')                #正确，将字节流写入文件
5
>>> myfile.write(b'\nbbbb')
5
```

- 对比找“不同”与“相同” **基本相同**

```
>>> myfile.close()
>>> myfile=open('code.txt','r')
>>> print(myfile.read())                    #打印读取文件的全部内容
aaaaa
bbbb
>>> myfile.close()                          #关闭文件
>>> myfile=open('code.txt','rb')
>>> print(myfile.read())                    #打印读取文件的全部内容
b'aaaaa\nbbbb'
>>> myfile.close()                          #关闭文件
```

■ 对比找“不同”与“相同” 基本相同

1.4 用文件存储对象

- 用文本文件或二进制文件格式直接存储Python中的各种对象，通常需要进行繁琐的转换。
- 可以使用Python标准模块Pickle处理文件中对象的读写。
- **Pickle 模块函数：**
- dump()：将 Python 中的对象序列化成二进制对象，并写入文件。
- load()：读取指定的序列化数据文件，并返回对象。
- dumps()：将 Python 中的对象序列化成二进制对象，并返回。
- loads()：读取给定的二进制对象数据，并将其转换为 Python 对象。
- **其中：**
- dump 和 load 实现基于文件的 Python 对象与二进制互转。
- dumps 和 loads 实现基于内存的 Python 对象与二进制互转。

文件

- 对比找“不同”与“相同” **基本相同**

```
>>> x=[1,2,'abc']                          #创建列表对象
>>> y={'name':'John','age':25}             #创建字典对象
>>> myfile=open('objdata.bin','wb')


>>> import pickle                          #导入pickle模块
>>> pickle.dump(x,myfile)                  #将列表对象写入文件
>>> pickle.dump(y,myfile)                  #将字典对象写入文件
>>> myfile.close()                         #关闭文件
>>> myfile=open('objdata.bin','rb')
```

- 对比找“不同”与“相同” **基本相同**

```
>>> myfile.read()                          #直接读取文件中的全部内容，查看内容
b'\x80\x03]q\x00(K\x01K\x02X\x03\x00\x00\x00abcq\x01e.\x80\x03}q\x00(X\x03\x00\
x00\x00ageq\x01K\x19X\x04\x00\x00\x00nameq\x02X\x04\x00\x00\x00Johnq\x03u.'
>>> myfile.seek(0)                         #将文件指针移动到文件开头
0
>>> x=pickle.load(myfile)                  #从文件读取对象
>>> x
[1, 2, 'abc']
>>> y=pickle.load(myfile)                  #从文件读取对象
>>> y
{'age': 25, 'name': 'John'}
```

■ 对比找“不同”与“相同” **基本相同**

1.5 目录操作

- 目录是一种特殊的文件，它存储当前目录中的子目录和文件的相关信息。
- Python的os模块提供了目录操作函数，使用之前应先导入模块：import os

os.getcwd()

- 该方法返回Python的当前工作目录。

```
>>> os.getcwd()
'D:\\code'
```

os.mkdir()

- 该方法用于创建子目录。

```
>>> os.mkdir('temp')                    #在当前目录中创建子目录
>>> os.mkdir('d:\\code\\temp')          #在绝对路径d:\code中创建子目录temp
```

■ 对比找“不同”与“相同” **基本相同**

os.rmdir()

- 该方法用于删除指定的空子目录。

```
>>> os.rmdir('temp')                    #删除当前目录的子目录
>>> os.rmdir('d: \\code\\temp')         #删除绝对路径中的子目录test
```

- os.rmdir()只能删除空的子目录，删除非空子目录时会出错。

```
>>> os.rmdir('temp')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OSError: [WinError 145] 目录不是空的。: 'temp'
```

■ 对比找“不同”与“相同” 基本相同

- **os.remove()**
- 该方法用于删除指定的文件。

```
>>> os.remove('d : \\code\\temp\\code.py')
```

- **os.listdir()**
- 该方法返回目录包含的子目录和文件名称。

```
>>> os.listdir()                          #列出当前目录内容
['code.txt', 'pycode', 'temp']
>>> os.listdir('d : \\code\\temp')        #列出指定目录内容
['test', 'test.py'  ]
```

os.rename()

- 该方法用于修改文件的名称。

```
>>> os.rename('d : \\code\\temp\\test.py',' 'd : \\code\\temp\\code.py')
```

■ 对比找“不同”与“相同” **基本相同**

os.chdir()

- 该方法用于改变当前目录。

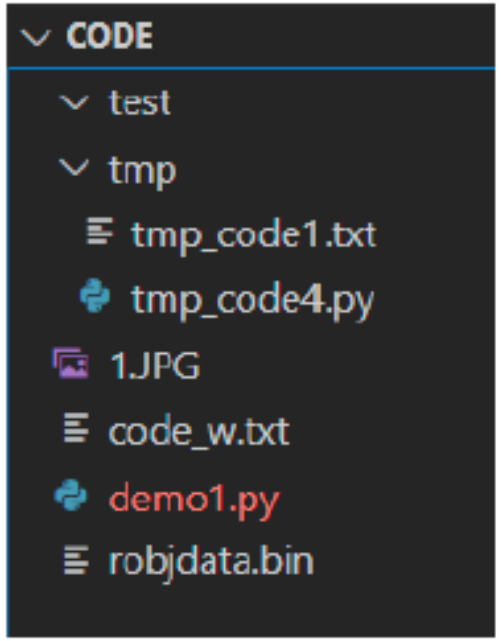
```
>>> os.getcwd()                    #查看当前目录
'd:\\code'
>>> os.mkdir('temp')               #在当前目录中创建子目录
>>> os.chdir('temp' )              #切换到新建目录
>>> os.getcwd()                    #查看新的当前目录
'd:\\code\\temp'
>>> os.chdir('d:\\code_temp')      #用绝对路径指定要切换的目录
>>> os.getcwd()
'd:\\code_temp'
>>> os.chdir('c:/' )               #切换到其他磁盘的目录
>>> os.getcwd()
'c:\\'
```

■ 对比找“不同”与“相同” 基本相同

os.walk()

- 该方法用于获取目录下所有路径、目录列表和文件列表。os.walk(top)
- 需要传入一个路径作为top参数，以top为根节点的目录树中游走，对树中的每个目录生成一个由(dirpath, dirnames, filenames)三项组成的三元组。
- 注意最终返回的结果是一个迭代器。

```
>>> os.listdir('.')
['1.JPG', 'code_w.txt', 'demo1.py', 'robjdata.bin', 'test', 'tmp']
>>> for item in os.walk("."):
...     print(item)
...
('.', ['test', 'tmp'], ['1.JPG', 'code_w.txt', 'demo1.py', 'robjdata.bin'])
('.\\test', [], [])
('.\\tmp', [], ['tmp_code1.txt', 'tmp_code4.py'  ])
```



- 对比找“不同”与“相同” **基本相同**

os.path模块

- 该模块包含了路径名称相关的函数。

方法	说明
os.path.abspath(path)	返回路径 path 的绝对路径
os.path.exists(path)	如果路径 path 存在，返回 True；如果路径 path 不存在，返回 False。
os.path.isfile(path)	判断路径是否为文件
os.path.isdir(path)	判断路径是否为目录
os.path.join(path1[, path2[, ...]])	把目录和文件名合成一个路径
os.path.split(path)	把路径分割成 dirname 和 basename，返回一个元组
os.path.splitext(path)	分割路径，返回路径名和文件扩展名的元组
...	...

西安交通大学 XI'AN JIAOTONG UNIVERSITY

■ 对比找“不同”与“相同” **基本相同**

2.1 CSV文件的基本概念

- CSV指Comma-Separated Values，即逗号分隔值。
- CSV文件也是**文本文件**，通常由多个记录组成，**第1行通常为记录的各个字段名称，第2行开始为记录数据**，每条记录包含相同的字段，字段之间用分隔符分隔。
- 分隔符可使用**逗号、空格、制表符、其他字符或字符串**。

专业名称，层次，科类

工程造价，高起专，文科

工商企业管理，高起专，文科

建筑工程技术，高起专，理科

工程造价，高起专，理科

汽车服务工程，专升本，理工类

文件

西安交通大学 XI'AN JIAOTONG UNIVERSITY

■ 对比找“不同”与“相同” **基本相同**

- CSV文件可用记事本创建、查看和编辑。
- 也可使用Excel打开、查看和编辑数据。
- 也可使用open()函数打开CSV文件，按文本文件方式读写CSV数据。采用这种方式需要将读取的每行字符串转换成字段数据，写入时需要添加分隔符。
- Python提供的csv模块提供了CSV文件读写功能。

	A	B	C	D	E	F	G	H	I
1	Name	Team	Number	Position	Age	Height	Weight	College	Salary
2	Avery Bradley	Boston Celtics	0	PG	25	6月2日	180	Texas	7730337
3	Jae Crowder	Boston Celtics	99	SF	25	6月6日	235	Marquette	6796117
4	John Holland	Boston Celtics	30	SG	27	6月5日	205	Boston University	
5	R.J. Hunter	Boston Celtics	28	SG	22	6月5日	185	Georgia State	1148640
6	Jonas Jerebko	Boston Celtics	8	PF	29	6月10日	231		5000000
7	Amir Johnson	Boston Celtics	90	PF	29	6月9日	240		12000000
8	Jordan Mickey	Boston Celtics	55	PF	21	6月8日	235	LSU	1170960
9	Kelly Olynyk	Boston Celtics	41	C	25	7-0	238	Gonzaga	2165160
10	Terry Rozier	Boston Celtics	12	PG	22	6月2日	190	Louisville	1824360
11	Marcus Smart	Boston Celtics	36	PG	22	6月4日	220	Oklahoma State	3431040
12	Jared Sullinger	Boston Celtics	7	C	24	6月9日	260	Ohio State	2569260
13	Isaiah Thomas	Boston Celtics	4	PG	27	5月9日	185	Washington	6912869
14	Evan Turner	Boston Celtics	11	SG	27	6月7日	220	Ohio State	3425510
15	James Young	Boston Celtics	13	SG	20	6月6日	215	Kentucky	1749840
16	Tyler Zeller	Boston Celtics	44	C	26	7-0	253	North Carolina	2616975
17	Bojan Bogdanovic	Brooklyn Nets	44	SG	27	6月8日	216		3425510
18	Markel Brown	Brooklyn Nets	22	SG	24	6月3日	190	Oklahoma State	845059
19	Wayne Ellington	Brooklyn Nets	21	SG	28	6月4日	200	North Carolina	1500000
20	ndse Hollis-Jeffers	Brooklyn Nets	24	SG	21	6月7日	220	Arizona	1335480
21	Jarrett Jack	Brooklyn Nets	2	PG	32	6月3日	200	Georgia Tech	6300000
22	Sergey Karasev	Brooklyn Nets	10	SG	22	6月7日	208		1599840
23	Sean Kilpatrick	Brooklyn Nets	6	SG	26	6月6日	219	Cincinnati	134215
24	Shane Larkin	Brooklyn Nets	0	PG	23	5月11日	175	Miami (FL)	1500000
25	Brook Lopez	Brooklyn Nets	11	C	28	7-0	275	Stanford	19689000
26	Chris McCullough	Brooklyn Nets	1	PF	21	6月11日	200	Syracuse	1140240

C:\Users\DAISY\Desktop\code\nba.csv - Notepad++

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?

nba.csv

```
1  Name,Team,Number,Position,Age,Height,Weight,College,Salary
2  Avery Bradley,Boston Celtics,0.0,PG,25.0,6-2,180.0,Texas,7730337.0
3  Jae Crowder,Boston Celtics,99.0,SF,25.0,6-6,235.0,Marquette,6796117.0
4  John Holland,Boston Celtics,30.0,SG,27.0,6-5,205.0,Boston University,
5  R.J. Hunter,Boston Celtics,28.0,SG,22.0,6-5,185.0,Georgia State,1148640.0
6  Jonas Jerebko,Boston Celtics,8.0,PF,29.0,6-10,231.0,,5000000.0
7  Amir Johnson,Boston Celtics,90.0,PF,29.0,6-9,240.0,,12000000.0
8  Jordan Mickey,Boston Celtics,55.0,PF,21.0,6-8,235.0,LSU,1170960.0
9  Kelly Olynyk,Boston Celtics,41.0,C,25.0,7-0,238.0,Gonzaga,2165160.0
10 Terry Rozier,Boston Celtics,12.0,PG,22.0,6-2,190.0,Louisville,1824360.0
11 Marcus Smart,Boston Celtics,36.0,PG,22.0,6-4,220.0,Oklahoma State,3431040.0
12 Jared Sullinger,Boston Celtics,7.0,C,24.0,6-9,260.0,Ohio State,2569260.0
13 Isaiah Thomas,Boston Celtics,4.0,PG,27.0,5-9,185.0,Washington,6912869.0
14 Evan Turner,Boston Celtics,11.0,SG,27.0,6-7,220.0,Ohio State,3425510.0
15 James Young,Boston Celtics,13.0,SG,20.0,6-6,215.0,Kentucky,1749840.0
```

■ 对比找“不同”与“相同” 基本相同

2.2 读CSV文件数据

csv模块提供了两种读取器对象来读取CSV文件数据：常规读取器和字典读取器。

常规读取器

- csv模块中的reader()函数用于创建常规读取器对象，其基本格式:

```
csvreader = csv.reader(csvfile, delimiter='分隔符')
```

- 变量csvreader用于引用读取器对象；
- csvfile是open()函数返回的文件对象；
- delimiter参数指定CSV文件使用的数据分隔符，默认为逗号。
- 常规读取器对象是一个可迭代对象，每次迭代返回一个包含一行数据的列表，列表元素对应CSV记录的各个字段。
- 可用for循环或next()函数迭代常规读取器对象。

招生专业.csv

```
1  专业名称,层次,科类
2  工程造价,高起专,文科
3  工商企业管理,高起专,文科
4  建筑工程技术,高起专,理科
5  工程造价,高起专,理科
6  汽车服务工程,专升本,理工类
```

■ 对比找“不同”与“相同” **基本相同**

3.1 JSON文件的基本概念

- JSON (JavaScript Object Notation) 是一种轻量级的数据交换格式，易于人阅读和编写，同时也易于机器解析和生成。
- 基于 ECMAScript（欧洲计算机协会制定的js规范）的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。
- **JSON基于两种结构构建：**
- **JSON对象：**无序，左花括号（{ ）开头，右花括号（} ）结尾，键值对表示属性，属性间用逗号（，）隔开。

  Json对象结构： {key1:value1,key2:value2...}

  key的数据类型：字符串，Key值必须用双引号标起来。

  value的数据类型：字符串、数值、布尔值、 null、json数组[]、json对象{}

■ 对比找“不同”与“相同” **基本相同**

- **JSON数组**：类似数组或序列，有序

  JSON数组结构：[value1，value2...]

  value的数据类型：字符串、数值、布尔值、null、json数组[]、json对象{}

```
{
   "addressbook": {"name":"Mary Lebow",
                   "address": {"street": "5 Main Street",
                   "city": " San Diego, CA",
                   "zip": "91912",
                   "phoneNumbers": ["619 332-3452", "664 223-4667 "]}
                    }
}
```

文件

- 对比找“不同”与“相同” **基本相同**

0001.jpg

https://github.com/wkentaro/labelme

0001.json


```
{
  "version": "4.0.0",
  "flags": {
    "__ignore__": false,
    "cat": true,
    "dog": false
  },
  "shapes": [],
  "imagePath": "0001.jpg",
  "imageData": null,
  "imageHeight": 480,
  "imageWidth": 640
}
```

■ 对比找“不同”与“相同” 基本相同

4. XLSX文件

- xlsx文件，即excel文件。Excel具有强大的数据处理功能。
- Python操作xlsx文件，可以使用openpyxl库或pandas库。

```
import openpyxl
```

- 打开excel文件后生成一个工作簿workbook。
- 每个工作簿中包含多张表单worksheet，正在操作的表单被称为活跃表单active sheet。
- 对于某一特定行和列的小格子称为单元格cell。

- 对比找“不同”与“相同” **基本相同**

例：

```
from openpyxl import Workbook
# 创建一个Workbook对象
wb = Workbook()
# 如果不指定sheet索引和表名，默认在第二张表位置新建表名sheet1
wb.create_sheet(index=1, title="sheet2")
# 获取当前活跃的sheet，默认为第一张sheet
ws = wb.active
print(ws)
# 获取当前工作簿的所有sheet对象
sheets = wb.worksheets
print(sheets)
```

- 对比找“不同”与“相同” **基本相同**

```
# 获取所有sheet的名字
sheets_name = wb.sheetnames
print(sheets_name)
# 保存为工作簿data1.xlsx
wb.save('data1.xlsx')
```

打印结果如下：

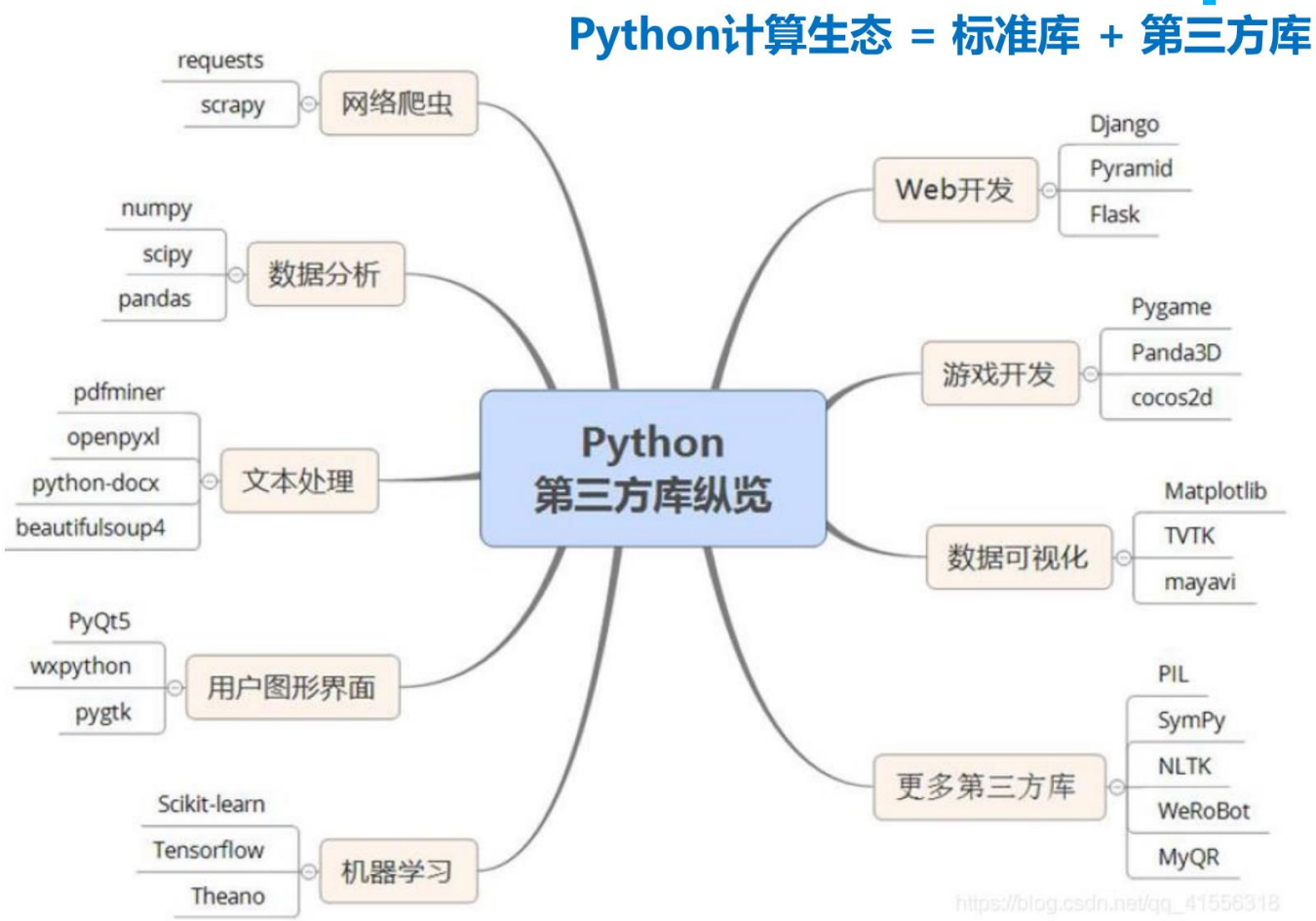
- 当前sheet名字为Sheet。
- 新建的sheet名字为sheet2。
- wb.worksheets返回的是Worksheet对象。
- wb.sheetnames返回的是表名字符串列表。

```
>>>
<Worksheet "Sheet">
[<Worksheet "Sheet">,
<Worksheet "sheet2">]
['Sheet', 'sheet2']
```

库

Python体系 | C/C++体系

- 对比找“不同”与“相同”
- 思想相同 有区别



- Standard C I/O
- Standard C String & Character
- Standard C Math
- Standard C Time & Date
- Standard C Memory
- Other standard C functions
- C++ I/O
- C++ String
- C++ 标准模板库
  - C++ Bitset
  - C++ Double-Ended Queue
  - C++ List
  - C++ Map
  - C++ Multimap
  - C++ Multiset
  - C++ Priority Queue
  - C++ Queue
  - C++ Set
  - C++ Stack
  - C++ Vector

总结

西安交通大学 XI'AN JIAOTONG UNIVERSITY

Python体系

1、Python概述
2、基本语法和数据类型
3、程序控制结构
4、组合数据类型（列表、元组、字典）
5、函数 、类和模块
6、文件和数据组织
7、库（标准库和第三方库）

C/C++体系

1、编程环境配置、程序运行和调试方法
2、输入输出、数学运算
3、分支结构、循环结构、程序异常处理
4、组合数据类型
5、函数 、类和模块
6、文件和图像操作（txt/csv/json/png/jpg等）
7、库：机器学习（Scikit库）、深度学习（Pytorch平台）、计算机视觉（Opencv库）

继续自学

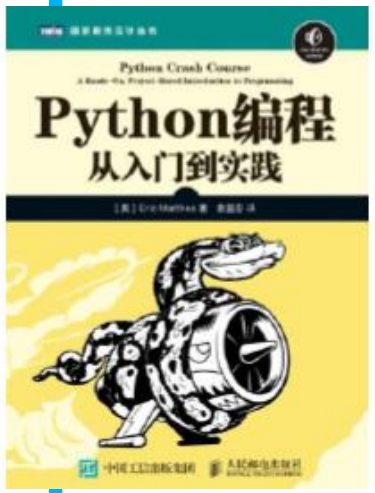
Python体系 | C/C++体系

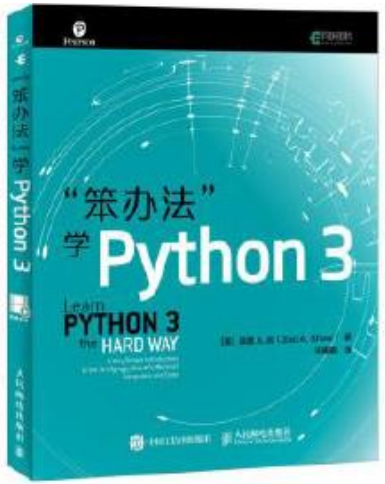
《Python编程：从入门到实践》

《笨办法学Python》

https://wizardforcel.gitbooks.io/lpthw/content/1.html

《简明Python教程》





本章结束！