

python3进阶： 函数和类

- 编程语言：一套拥有严谨语法规则，用于给计算机下指令的语言
- 按抽象程度划分
 - 低级语言：接近机器语言
 - 优点：极致的执行效率； 缺点：不易于理解和学习
 - 高级语言：接近人类的语言
 - 优点：易于理解和学习、传播； 缺点：执行效率相对不高
- 按程序的设计划分
 - 面向过程：将功能的逻辑步骤一一实现，注重过程
 - 优点：执行效率奇高； 缺点：复用性和灵活度低
 - 面向对象：将实现功能的代码标签化，后续只需调用适当的标签即可完成相应的功能
 - 优点：强大的复用性和灵活性； 缺点：执行效率不高
- 按执行方式划分
 - 编译型：根据严格语法逻辑，将代码统一编译成机器能识别的字节码
 - 优点：执行效率高； 缺点：容错性差
 - 解释型：逐句读取代码并编译成机器能识别的字节码
 - 优点：强大的试错能力； 缺点：执行效率不高
 - 混合型：将代码先编译成第三方机器码，然后在相应的虚拟机中执行第三方机器码
 - 优点：较高的执行效率，较强的脚本化能力，极致的移植性； 缺点：需要依赖第三方虚拟机
- python是一门解释型、面向对象的高级语言。python易于学习和理解传播，但效率奇差
-
- 函数：将实现功能的代码语句组标签化，后续可以通过调用该标签，传入适当参数得到相应结果
- 定义一个函数相当于制定一个计划，计划中需要的前提条件称为形式上的参数，简称形参
- 调用一个函数相当于实现一个计划，计划中需要的条件必须被满足被称为实际上的参数，简称实参
- 定义函数的关键字：def
- 定义函数的格式：
 - def 函数名 (形参, ...) :
 - 实现功能的代码语句组
 - 根据实际情况，形参可以有多个或没有形参
 - 函数被调用，则会执行实现功能的代码语句组

```
206  
207     def add(a,b):  
208         print(a+b)  
209  
210     add(1,2)  
211  
212 # class Demo:  
  
test01  
D:\Python38\python.exe D:/PycharmProjects/test105/test01.py  
3  
  
Process finished with exit code 0
```

- 位置参数：传递实参的顺序依照形参的位置
- 关键字参数：传递实参的时候指定传递给某个形参
 - 关键字参数可以无视位置关系，直接传递给指定形参
- 默认值参数：形参在被定义时，指定了默认值。后续传入实参的时候，如果不给该形参传值，则使用默认值
 - 默认值参数需要在没有默认值参数形参之后

```
207     def add(a,b=9):  
208         print(a+b)  
209     add(1)  
  
test01  
D:\Python38\python.exe D:/PycharmProjects/test105/test01.py  
10  
  
Process finished with exit code 0
```

```
207     def add(a,b=9):  
208         print(a+b)  
209     add(1,6)  
210  
  
test01  
D:\Python38\python.exe D:/PycharmProjects/test105/test01.py  
7  
  
Process finished with exit code 0
```

- **函数的返回值：函数中如果有return关键字，则该函数具有返回值
 - 有返回值的函数需要进一步处理
 - 可以将返回值输出在控制台，也可以传递给其他函数充当参数
- ***当代码执行到return关键字时，立刻退出函数返回结果
 - 多条件分支中，也只返回一次

```
210     def bdd(a,b):  
211         return a+b  
212     i=bdd(1,2)  
213     print(i)  
  
test01  
D:\Python38\python.exe D:/PycharmProjects/test105/test01.py  
3  
  
Process finished with exit code 0
```

```

210     def bdd(a,b):
211         if a>b:
212             return a-b
213         elif a==b:
214             return a*b
215         else:
216             return b-a
217 i=bdd(1,2)
218 print(i)
219

```

t01
D:\Python38\python.exe D:/PycharmProjects/test105/test01.py
1

Process finished with exit code 0

- 当返回多个结果时，默认以元祖的方式打包返回

```

219     def cdd(a,b):
220         return a+b,a-b
221     print(cdd(1,2))
222

```

Run test01
D:\Python38\python.exe D:/PycharmProjects/test105/test01.py
(3, -1)

Process finished with exit code 0

全局变量与局部变量

- 全局变量：作用域为全局的变量。通常是顶格定义的变量
- 局部变量：作用域为局部某个范围内的变量
 - 可以通过global关键字将局部变量转换成全局变量。前提是global关键字要被执行

```

219     # print(i)
220     def cdd(a,b):
221         return a+b,a-b
222     #print(cdd(1,2))
223     i=10
224     def ddd(a,b):
225         global c
226         c=a+b
227         ddd(1,2)
228         print(cdd(i,c))
229     # class Demo:

```

Run test01
D:\Python38\python.exe D:/PycharmProjects/test105/test01.py
(13, 7)

Process finished with exit code 0

*组包与拆包

- 组包：当函数的形参不固定，通常使用组包的方式来接受参数
 - *：通常表示为 *args，接受所有独立的参数，组包成一个元祖，以供后续使用

```

229     def edd(*args):
230         print(args)
231
232     edd(21,43,64,321,654,768)
233

```

Run test01
D:\Python38\python.exe D:/PycharmProjects/test105/test01.py
(21, 43, 64, 321, 654, 768)

Process finished with exit code 0

- **：通常表示为 **kwargs，接受所有的键值对参数，组包成一个字典，以供后续使用

```

233     def edd(**kwargs):
234         print(kwargs)
235         edd(a=1,b=2,c=3)
236
237 # class Demo:

```

test01
D:\Python38\python.exe D:/PycharmProjects/test105/test01.py
{'a': 1, 'b': 2, 'c': 3}
Process finished with exit code 0

- *拆包：当给函数传递一个集合时，通常使用拆包的方式将集合拆分成相应的实参传递

- *：单 * 可以将列表/元组拆分成独立的参数，传递给函数
 - 传入的列表或元组的元素个数，需要与函数的形参个数相匹配

```

236     l=[1,2,3,4]
237     def edd(a,b,c,d):
238         print(a,b,c,d)
239         edd(*l)
240

```

test01
D:\Python38\python.exe D:/PycharmProjects/test105/test01.py
1 2 3 4
Process finished with exit code 0

- **：双 * 可以将字典拆分成独立的键值对，以关键字参数的方式，传递给函数
 - 传入的字典的键，需要与函数的形参名称相匹配

```

240     d={'a':1,'d':5,'b':2,'c':3}
241     def edd(a,b,c,d):
242         print(a,b,c,d)
243         edd(**d)
244

```

test01
D:\Python38\python.exe D:/PycharmProjects/test105/test01.py
1 2 3 5
Process finished with exit code 0

-

**强类型语言与弱类型语言

- 强类型语言：变量在被定义时就已经指定了该变量的数据类型，并且后续该变量也只能接受被指定的数据类型
- 弱类型语言：变量的数据类型，取决于被赋予哪种数据类型

- 在弱类型语言中通常使用“形参:数据类型”的方式来约束传入实参的数据类型**

```

def fdd(a=''):    可以通过给形参默认值的方式来提前使用某种数据类型的方法
    a.|          不推荐通过默认值的方式
    a.|__count(self, x)      str
    a.|__isdigit(self)      str
    a.|__join(self, iterable) str
    a.|__index(self, sub, start, end) str
    a.|__endswith(self, suffix, start, end) str
    a.|__startswith(self, prefix, start, end) str
    a.|__title(self)        str
    a.|__capitalize(self)   str
    a.|__lower(self)        str
    a.|__upper(self)        str
    a.|__str__(self)        str
    a.|__len__(self)        le>

```

Press Ctrl+Shift to choose the selected (or first) suggestion and insert a dot afterwards ↵

```
def fdd(a:str): 通过“形参:数据类型”的方式, 可以预约传入实参的数据类型
    # print(a.strip())
    a.|
```

The screenshot shows a PyCharm code editor with a tooltip displaying the available methods for a string object. The tooltip lists methods like strip, count, isdigit, join, index, endswith, startswith, title, capitalize, and lower. The 'strip' method is highlighted.

- 函数的扩展

- 匿名函数：通过lambda关键字定义的函数称为匿名函数
 - 匿名函数只是一个表达式，格式为：函数名 = lambda 形参,...: 逻辑

```
246 hehe=lambda x,y:x+y
247 print(hehe(1,2))
```

The screenshot shows a PyCharm code editor with a terminal window below it. The terminal shows the execution of a lambda function: 'hehe=lambda x,y:x+y' on line 246 and 'print(hehe(1,2))' on line 247. The output is '3'.

- 匿名函数只能编辑有限的逻辑
- ***如何编写一个函数
- 套壳已有的函数或逻辑

- 被套壳的函数需要什么参数，套壳函数就提供什么参数

```
menie = lambda x,y:x+y
def 输出(x): 套壳函数
    print(x) 被套壳的函数
```

The screenshot shows a PyCharm code editor with a tooltip explaining function wrapping. It says '被套壳的函数需要什么参数，套壳函数就提供什么参数'. Below the code, there is a call to '输出(13213+32131)'.

```
248 def 输出(x):
249     print(x)
250     输出(1213+313)
251
```

The terminal window shows the output of the wrapped function: '1526'.

- 写算法实现
 - 先流水账代码实现逻辑
 - 先通过伪代码的方式梳理逻辑功能，然后通过代码实现伪代码的思路
 - 然后通过def关键字，定义函数名，将流水账代码统一起来
 - 将流水账代码中涉及的数据剥离出来，以形参的方式传入

```

254     def sort(l:list):
255         #l=[3,7,6,9,1]
256         for i in range(len(l)-1):#i:0,1,2,3
257             for j in range(len(l)-1-i):#5-1-0=4;5-1-1=3;5-1-2=2;5-1-3=1
258                 if(l[j]>l[j+1]):
259                     # temp=l[j]
260                     # l[j]=l[j+1]
261                     # l[j+1]=temp
262                     l[j],l[j+1]=l[j+1],l[j]
263             print(l)
264
265         sort([1,2,4,6,1,7,3,9])
266
267     # class Demo:

```

test01
D:\Python38\python.exe D:/PycharmProjects/test105/test01.py
[1, 1, 2, 3, 4, 6, 7, 9]
Process finished with exit code 0

- 类：类是具有相同属性和行为的一类事物的抽象
- 类可以理解成一个年度计划或阶段性计划
- 定义类的关键字：class
- 定义类的格式：
 - class 类名：
 - 有用的变量
 - 有用的函数
- 定义类的过程相当于制定一个工具箱，无需考虑客观事实，有用是唯一标准
- 类中的变量称为成员变量或类的属性；类中的函数称为类方法
- 类中可以没用成员变量或类方法
- **self形参：在类中定义类方法时必须要有的形参**
 - self形参只作为身份标识，调用类方法时不需要给self形参传值
 - 在类的作用域中，self代表类本身
 - 在类方法中，可以通过self形参调用类中的成员属性和其他类方法

```

class Demo:
    x = 0
    y = 1
    def add(self,a,b):
        print(a+b)
    def new(self):
        print('Hello world')
    def test(self):
        print(self)

demo = Demo()

```

在类方法中，可以通过self形参调用类中其他类方法或者类中的成员变量
self: 在类的作用域中，相当于类本身

/PycharmProjects/test105/test01.py
t code 0

```

class Demo:
    x=3
    y=5
    __z=2
    def __init__(self,n):
        self.n=n
    def add(self,a,b):
        print(a+b)
    def hel(self):
        print('hello_world')
    def t(self):
        print(self)
    def test(self):
        print(self.__z)
    def new_test(self):
        print(self.n)

```

- 使用类时需要实例化该类
 - 实例化：将类赋予某个变量的过程。成功的实例化会得到类的实例化对象
 - 实例化对象：可以理解成类的代言人

The screenshot shows a Python code editor with the following code:

```

class Demo:
    x = 0    成功的实例化会得到类的实例化对象，可以通过类的实例化对象
    y = 1    调用类中已有的成员变量和类方法
    def add(self,a,b):
        print(a+b)
    def new(self):
        print('hello world')

demo = Demo()
demo.  方法
        add(self, a, b)  方法
        new(self)
        f x
        f y

```

Below the code, there is a list of member variables:

成员变量

Demo
Demo
Demo
Demo

On the left, the project structure is visible:

- bases
- pages
- temp
- External Libraries

At the bottom, the terminal window shows:

```

D:\Python38\python.exe D:/PycharmProjects/test105/temp/test01.py
2
7
Process finished with exit code 0

```

- self和实例化对象的区别

- 作用域不同：在类的作用域之内，调用成员变量或类方法时通常使用self；在类的作用域之外，通常使用实例化对象来调用
- self形参可以调用类中私有的成员变量和私有的类方法，实例化对象不可以
 - 私有的成员变量和私有的类方法：以双下划线开头命名的成员变量/类方法

The screenshot shows the PyCharm interface. The top window, titled 'test10.py', contains Python code defining a class 'Demo' with private attributes and methods. The bottom window, titled 'test01', shows the execution results of the code.

```

class Demo:
    x=3
    y=5
    __z=2
    def add(self,a,b):
        print(a+b)
    def hel(self):
        print('hello_world')
    def test(self):
        print(self.__z)
        self.add(3,4)

demo=Demo()
#demo.add(1,2)
#demo.hel()
demo.test()

```

The output window shows the results of the execution:

```

D:\Python38\python.exe D:/PycharmProjects/test105/test01.py
2
7
Process finished with exit code 0

```

- **重要格式：标识符 = 标识符 ()
 - 第一种情况：实例化一个类，得到类的实例化对象
- ```

demo=Demo(2)
demo.test()

```
- 第二种情况：调用一个函数，获取函数的返回值
- \*\*\*类的初始化方法：以 \_\_init\_\_ 命名的类方法
  - init是initial的缩写
  - 初始化方法具有最高的优先级，实例化某个类，相当于实现该类的初始化方法

The screenshot shows the PyCharm interface. The top window, titled 'test11u.py', contains Python code defining a class 'Demo' with an \_\_init\_\_ method and other methods. The bottom window, titled 'test01', shows the execution results of the code.

```

class Demo:
 x=3
 y=5
 __z=2
 def __init__(self,n):
 self.n=n
 def add(self,a,b):
 print(a+b)
 def hel(self):
 print('hello_world')
 def test(self):
 print(self.__z)
 self.add(3,4)
 def new_test(self):
 print(self.n)

demo=Demo(1)
demo.new_test()

```

The output window shows the results of the execution:

```

D:\Python38\python.exe D:/PycharmProjects/test105/test01.py
1
Process finished with exit code 0

```

- 可以理解成实例化某个类的门槛
- 可以在类的初始化方法中，提出参数或其他要求

```

class Demo:
 x = 0
 y = 1
 __z = 2
 n = 3
 def __init__(self,n):
 self.n = n
 def add(self,a,b):
 print(a+b)
 def new(self):
 print('hello world')
 def test(self):
 print(self.__z)
 self.add(3,4)
 def new_test(self):
 print(self.n)

demo = Demo()
demo.test()

```

```

class Demo:
 x=3
 y=5
 __z=2
 def __init__(self,n):
 self.n=n
 def add(self,a,b):
 print(a+b)
 def hel(self):
 print('hello_world')
 def t(self):
 print(self)
 def test(self):
 print(self.__z)
 self.add(3,4)
 def new_test(self):
 print(self.n)

```

- \*\*什么情况下会给类添加初始化方法
- 当类中依赖某个关键参数时，通常在初始化方法中提出参数需求
- 当类中依赖某种环境时，可以在初始化方法中通过逻辑检查系统是否存在某种环境

```

test105.py
External Libraries
267 class Demo:
268 x=3
269 y=5
270 __z=2
271 def __init__(self,n):
272 self.n=n
273 def add(self,a,b):
274 print(a+b)
275 def hel(self):
276 print('hello_world')
277 def test(self):
278 print(self.__z)
279 def new_test(self):
280 print(self.n)
281
282 demo=Demo(1)
283 demo.new_test()

```

D:\Python38\python.exe D:/PycharmProjects/test105/test01.py  
1  
Process finished with exit code 0

- \*类的继承：类可以继承其他类，也可以被其他类继承。继承其他类的类称为子类，被其他类继

## 承的类称为父类

- 类的继承主要是：在不改变父类的代码的前提下，扩展类的功能
- 类的继承格式
  - class 类名（父类， ...）：
    - 新的成员变量
    - 新的类方法
  - 子类默认具有父类的所有成员变量和类方法，私有的除外
  - 可以理解成将父类的代码在子类中创建快捷方式

The screenshot shows the PyCharm IDE interface. At the top, there is a code editor with two classes defined:

```
❶ class Demo:
 x = 0
 y = 1
 __z =
 # n = 0, 如果存在成员变量n, 会被初始化方法接受到的参数n给重新赋值; 如果不存在成员变量n, 则会新增一个成员变量
 def __init__(self,n):
 self.n = n
 def add(self,a,b):
 print(a+b)
 def new(self):
 print('hello world')
 def test(self):
 print(self.__z)
 def new_test(self):
 print(self.n)

❷ class NewDemo(Demo):
 pass
```

A red arrow points from the text "可以理解成将父类的代码，在子类中创建快捷方式" to the inheritance line "class NewDemo(Demo):". Below the code editor, a tooltip shows the inheritance path: "子类默认具有父类的所有成员变量和类方法，私有的除外".

In the bottom right corner, there is a terminal window showing the output of a Python script named "test01". The script defines a class "NewDemo" that inherits from "Demo". It contains a method "haha" that prints "haha". An instance of "NewDemo" is created and its "haha" method is called.

```
❶ 266 ❷ class Demo:...
267 ❸ class NewDemo(Demo):
282
283 hehe=0
284 def haha(self):
285 print('haha')
286
287 new=NewDemo(2)
288 new.haha()
290
291 #demo=Demo(1)

❷ test01
❸ D:\Python38\python.exe D:/PycharmProjects/test105/test01.py
haha

Process finished with exit code 0
```

- 重写：当子类中有与父类重名的成员变量/类方法时，称为重写
  - 重写是针对子类视角，相当于父类的成员变量和类方法，在子类中有新的实现
  - 子类的实例化对象和self形参，优先调用自己的成员变量和类方法，自己没有才去父类中获取

```

50 class Demo:
51 x = 0
52 y = 1
53 __z = 2
54 # n = 0 如果存在成员变量n, 会被初始化方法接受到的参数n给重新赋值; 如果不存在成员变量n, 则会新增一个成员变量
55 def __init__(self, n):
56 self.n = n
57 def add(self, a, b):
58 print(a+b)
59 def new(self):
60 print('hello world')
61
62 def test(self):
63 print(self.__z)
64 self.add(3, 4)
65 def new_test(self):
66 print(self.n)

67 class NewDemo(Demo):
68 hehe = 0
69 def haha(self):
70 print('hahaha')
71 def test(self):
72 print('子类的test方法')
73
74 new = NewDemo(1)
75 new.
76
77 new(self) Demo
78 n Demo
79 test(self) NewDemo
80 hehe Demo
81 haha(self) NewDemo
82 new_test(self) Demo
83 x Demo
84 y Demo
85 __class__ object
86
87
88 ranzhi_login_page.py
89 temp
90 test01.py
91 test10.py
92 test13.py
93 test14.py
94
95 External Libraries
96
97 Run test01
D:\Python38\python.exe D:/PycharmProjects/test105/temp/test01.py
子类的test方法
Process finished with exit code 0

```

- 在子类的类方法中，可以通过super () 超类来调用父类被重写的成员变量和类方法

```

1 class Demo:
2
3 class NewDemo(Demo):
4 hehe = 0
5 y = 10
6 def haha(self):
7 print('hahaha')
8 def test(self):
9 print('子类的test方法')
10 print(self.y)
11 print(super().__dict__)
12
13 new = NewDemo(1)
14 new.
15
16 new(self) Demo
17 test(self) Demo
18 add(self, a, b) Demo
19 new_test(self) Demo
20 x Demo
21 y Demo
22 __z Demo
23 __class__ object
24
25 D:/PycharmProjects/test105/test12.py", line 15
26 Press Ctrl+Shift+F10 to choose the selected (or first) suggestion and insert a dot afterwards >>>
missing 1 required positional argument: 'n'

```

```

class Demo:
 x=3
 y=5
 __z=2
 def __init__(self,n):

```

The screenshot shows a PyCharm interface with a code editor and a terminal window.

**Code Editor:**

```

285 class NewDemo(Demo):
286 hehe=0
287 y=10
288 def hahaha(self):
289 print('hahaha')
290 def test(self):
291 print('子类的test方法')
292 print(self.y)
293 print(super().y)
294

```

**Terminal Output:**

```

D:\Python38\python.exe D:/PycharmProjects/test105/temp/test01.py
子类的test方法
10
5
Process finished with exit code 0

```

### \*\*\*子类的初始化方法

- 子类默认具有父类的初始化方法，以`__init__`命名的类方法
- 子类也可以扩展自己的初始化方法，但必须先满足父类的初始化方法

The screenshot shows a Pycharm interface with a code editor and a terminal window.

**Code Editor:**

```

284 class NewDemo(Demo):
285 hehe=0
286 y=10
287 def __init__(self):
288 pass
289 def hahaha(self):
290 print('hahaha')
291 def test(self):
292 print('子类的test方法')
293 print(self.y)
294 print(super().y)
295
296 new=NewDemo()
297 new.test()
298
299
300

```

**Terminal Output:**

```

D:\Python38\python.exe D:/PycharmProjects/test105/temp/test01.py
子类的test方法
10
5
Process finished with exit code 0

```

- 可以在子类的初始化方法中，通过`super().__init__()`的方式满足父类的初始化方法

The screenshot shows a PyCharm interface with a code editor containing annotated code. Red boxes highlight specific parts of the code, and red text provides explanatory comments.

**Annotated Code:**

```

62 z = 2
63 # n = 0 如果存在成员变量n，会被初始化方法接受到的参数n给重新赋值；如果不存在成员变量n，则会新增一个成员变量
64 def __init__(self,n):
65 self.n = n
66 def add(self,a,b):
67 print(a+b)
68 def new(self):
69 print('hello world')
70 def test(self):
71 print(self...z)
72 self.add(3,4)
73 def new_test(self):
74 print(self.n)
75
76 class NewDemo(Demo):
77 hehe = 0
78 y = 10
79 def __init__(self,n,m):
80 super().__init__(n) 子类可以有自己的初始化方法，但必须先满足父类的初始化方法，确保父类能够正常实例化
81 m = m
82 def hahaha(self):
83 print('hahaha')
84 def test(self):
85 print('子类的test方法')
86 print(self.y)
87 print(super().y)
88
89 new = NewDemo(1,6)
90 new.test()
91 new.new_test()

```

```

66
67 ① class Demo:
68 x=3
69 y=5
70 z=2
71 ① def __init__(self,n):
72 self.n=n
73 def add(self,a,b):
74 print(a+b)
75 def hel(self):
76 print('hello_world')
77 def t(self):
78 print(self)
79 ① def test(self):
80 print(self.__z)
81 self.add(3,4)
82 def new_test(self):
83 print(self.n)

② class NewDemo(Demo):
 hehe=0
 y=10
 def __init__(self,n,m):
 super().__init__(n)
 self.m=m
 def haha(self):
 print('haha')
 def test(self):
 print('子类的test方法')
 print(self.y)
 print(super().y)

new=NewDemo(1,6)

```

- 面向对象的三大特征：继承、封装和多态
  - 封装：通过class关键字，将函数和变量统一起来的过程
  - 多态：父类的方法在子类中有不同的实现