


图解 Python 编程(25) | 面向对象编程

🕒 2021-11-09 👁 1651 💬 0 📁 工具教程 python 编程语言

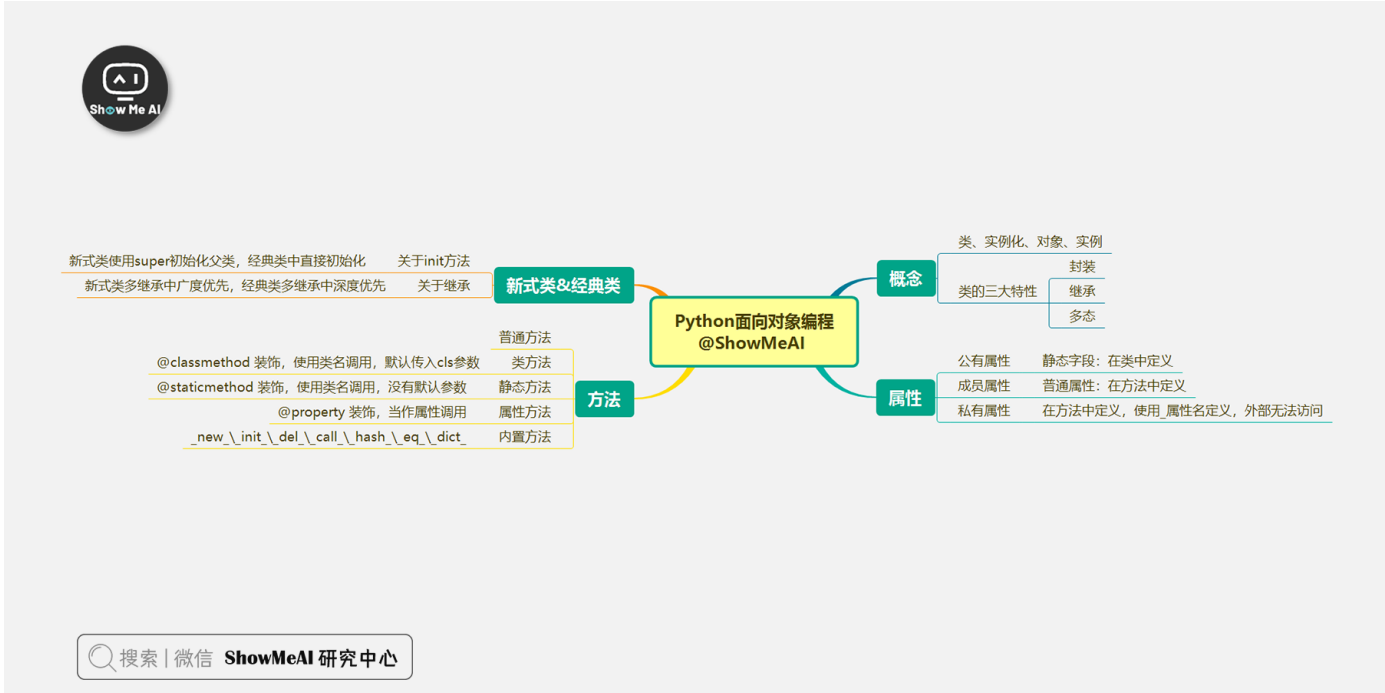
 ShowMeAI用知识加速每一次技术成长

作者：韩信子@ShowMeAI
教程地址：<https://www.showmeai.tech/tutorials/56>
本文地址：<https://www.showmeai.tech/article-detail/88>
声明：版权所有，转载请联系平台与作者并注明出处
收藏ShowMeAI查看更多精彩内容

Python面向对象

面向对象编程，在英文中称之为Object Oriented Programming，简称OOP，是一种程序设计思想。OOP把对象作为程序的基本单元，一个对象包含了数据和操作数据的函数。

Python是一个纯天然面向对象的编程语言，在Python中，所有数据类型都可以视为对象。自定义的对象数据类型就是面向对象中的类（Class）的概念。



@ShowMeAI">

面向对象概念

- **类(Class)**: 用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。
- **方法**: 类中定义的函数。
- **类变量**: 类变量在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用。
- **数据成员**: 类变量或者实例变量用于处理类及其实例对象的相关的数据。
- **方法重写**: 如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（override），也称为方法的重写。
- **局部变量**: 定义在方法中的变量，只作用于当前实例的类。
- **实例变量**: 在类的声明中，属性是用变量来表示的，这种变量就称为实例变量，实例变量就是一个用 **self** 修饰的变量。
- **继承**: 即一个派生类（derived class）继承基类（base class）的字段和方法。继承也允许把一个派生类的对象作为一个基类对象对待。例如，有这样一个设计：一个Dog类型的对象派生自Animal类，这是模拟”是一个（is-a）”关系（例图，Dog是一个Animal）。
- **实例化**: 创建一个类的实例，类的具体对象。
- **对象**: 通过类定义的数据结构实例。对象包括两个数据成员（类变量和实例变量）和方法。

相比其它编程语言，Python的类机制非常简洁，Python中的类提供了面向对象编程的所有基本功能：

- 类的继承机制允许多个基类
- 派生类可以覆盖基类中的任何方法
- 方法中可以调用基类中的同名方法

对象可以包含任意数量和类型的数据。

类定义

语法格式如下：

```
1. class ClassName:  
2.     <statement-1>  
3.     .  
4.     .  
5.     .  
6.     <statement-N>
```

类实例化后，可以使用其属性，实际上，创建一个类之后，可以通过类名访问其属性。

类对象



创建类对象

```
#创建类  
class Foo:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
#根据类Foo创建对象  
#自动执行Foo类的__init__方法  
obj1 = Foo('ShowMeAI', 18)
```

称为**构造方法**
根据类创建对象时自动执行

将**ShowMeAI**和**18**
分别封装到obj1和self的
name和age属性中

obj1 = Foo('ShowMeAI', 10)

#根据类Foo创建对象
#自动执行Foo类的__init__方法
obj2 = Foo('google', 24)

}
将google和24
分别封装到obj2和self的
name和age属性中

 <http://www.showmeai.tech/>

 搜索 | 微信 **ShowMeAI** 研究中心

属性引用和实例化

类对象支持两种操作：属性引用和实例化。

属性引用使用和Python中所有的属性引用一样的标准语法：**obj.name**。

类对象创建后，类命名空间中所有的命名都是有效属性名。所以如果类定义是这样：

```
1. class NewClass:
2.     """一个简单的类实例"""
3.     num = 123456
4.     def f(self):
5.         return 'hello ShowMeAI'
6.
7. # 实例化类
8. x = NewClass()
9.
10. # 访问类的属性和方法
11. print("NewClass 类的属性 num 为：", x.num)
12. print("NewClass 类的方法 f 输出为：", x.f())
```

以上创建了一个新的类实例并将该对象赋给局部变量 x，x 为空的对象。

执行以上程序输出结果为：

```
1. NewClass 类的属性 num 为： 123456
2. NewClass 类的方法 f 输出为： hello ShowMeAI
```

构造函数

类有一个名为 `__init__()` 的特殊方法（构造方法/构造函数），该方法在类实例化时会自动调用，像下面这样：

```
1. def __init__(self):
2.     self.data = []
```

类定义了 `__init__()` 方法，类的实例化操作会自动调用 `__init__()` 方法。如下实例化类NewClass，对应的 `__init__()` 方法就会被调用：

```
1. x = NewClass()
```

当然，`__init__()` 方法可以有参数，参数通过 `__init__()` 传递到类的实例化操作上。例如（代码可以在在线python3环境中运行）：

```
1. class MyComplex:
2.     def __init__(self, real_part, imag_part):
3.         self.r = real_part
4.         self.i = imag_part
5. x = MyComplex(5.0, -3.4)
6. print(x.r, x.i) # 输出结果：5.0 -3.4
```

(3) self代表类的实例，而非类

类的方法与普通的函数只有一个特别的区别——它们必须有一个额外的第一个参数名称，按照惯例它的名称是 **self**。

```
1. class Test:
2.     def prt(self):
3.         print(self)
4.         print(self.__class__)
5.
6. t = Test()
7. t.prt()
```

以上实例执行结果为：

```
1. <__main__.Test instance at 0x100724179>
2. __main__.Test
```

通过执行结果，可以看出，**self**代表的是类的实例（包含当前对象的地址信息），而**self.class**则指向类。

注意这里的**self**并不是python关键字，把他换成其他名称，比如showmeai，也是可以正常执行的（代码可以在在线python3环境中运行）：

```
1. class Test:
2.     def prt(showmeai):
3.         print(showmeai)
4.         print(showmeai.__class__)
5.
6. t = Test()
7. t.prt()
```

以上实例执行结果为：

```
1. <__main__.Test instance at 0x100724179>
2. __main__.Test
```

类的方法

在类的内部，我们可以使用 **def** 关键字来定义类方法，类方法必须包含参数**self**, 且其为第一个参数，**self**代表的是类的实例。例如（代码可以在[在线python3](#)环境中运行）

```
1. class Person:
2.     #定义基本属性
3.     name = "
4.     age = 0
5.     #定义私有属性,私有属性在类外部无法直接进行访问
6.     __weight = 0
7.     #定义构造方法
8.     def __init__(self,n,a,w):
9.         self.name = n
10.        self.age = a
11.        self.__weight = w
12.    def talk(self):
13.        print("%s的年龄是 %d 岁。" %(self.name,self.age))
14.
15. # 实例化类
16. p = Person('ShowMeAI',30,30)
17. p.talk()
```

执行以上程序输出结果为：

```
1. ShowMeAI的年龄是 30 岁。
```

继承

Python 同样支持类的继承。派生类的定义如下所示:

```
1. class DerivedClass(BaseClass):
2.     <statement-1>
3.     .
4.     .
5.     .
6.     <statement-N>
```

子类（派生类/DerivedClass）会继承父类（基类/BaseClass）的属性和方法。

BaseClassName（实例中的基类名）必须与派生类定义在一个作用域内。除了类，还可以用表达式，基类定义在另一个模块中时这一点非常有用：

```
1. class DerivedClassName(modname.BaseClassName):
```

```
1. #类定义
2. class person:
3.     #定义基本属性
4.     name = "
5.     age = 0
6.     #定义私有属性,私有属性在类外部无法直接进行访问
7.     __weight = 0
8.     #定义构造方法
9.     def __init__(self,n,a,w):
10.         self.name = n
11.         self.age = a
12.         self.__weight = w
13.    def talk(self):
14.        print("%s的年龄是 %d 岁。" %(self.name,self.age))
15.
16. #单继承示例
17. class student(person):
18.     grade = "
19.     def __init__(self,n,a,w,g):
20.         #调用父类的构造函数
21.         people.__init__(self,n,a,w)
22.         self.grade = g
23.     #覆写父类的方法
24.     def talk(self):
25.         print("%s的年龄是 %d 岁， 目前正在读 %d 年级"%(self.name,self.age,self.grade))
26.
27. s = student('小Show',12,60,5)
28. s.talk()
```

执行以上程序输出结果为：

```
1. 小Show的年龄是 12 岁， 目前正在读 5 年级
```

多继承

Python同样支持多继承形式。多继承的类定义形如下例:

```
1. class DerivedClassName(Base1, Base2, Base3):
2.     <statement-1>
3.     .
4.     .
5.     .
6.     <statement-N>
```

需要注意圆括号中父类的顺序，若是父类中有相同的方法名，而在子类使用时未指定，python从左至右搜索 即方法在子类中未找到时，从左到右查找父类中是否包含方法。

```
1. #类定义
2. class person:
3.     #定义基本属性
4.     name = "
5.     age = 0
6.     #定义私有属性,私有属性在类外部无法直接进行访问
7.     __weight = 0
8.     #定义构造方法
9.     def __init__(self,n,a,w):
10.         self.name = n
11.         self.age = a
12.         self.__weight = w
13.     def speak(self):
14.         print("%s的年龄是 %d 岁。" %(self.name,self.age))
15.
16. #单继承示例
17. class student(person):
18.     grade = "
19.     def __init__(self,n,a,w,g):
20.         #调用父类的构造函
21.         people.__init__(self,n,a,w)
22.         self.grade = g
23.     #覆写父类的方法
24.     def talk(self):
25.         print("%s的年龄是 %d 岁， 目前正在读 %d 年级"%(self.name,self.age,self.grade))
26.
27. #另一个类，多重继承之前的准备
28. class speaker():
29.     topic = "
30.     name = "
31.     def __init__(self,n,t):
32.         self.name = n
33.         self.topic = t
34.     def talk(self):
35.         print("%s是一个演说家，今天ta演讲的主题是 %s"%(self.name,self.topic))
36.
37. #多重继承
38. class sample(speaker,student):
39.     a ="
40.     def __init__(self,n,a,w,g,t):
41.         student.__init__(self,n,a,w,g)
42.         speaker.__init__(self,n,t)
43.
44. test = sample("ShowMeAI",25,80,4,"Python")
45. test.talk() #方法名同，默认调用的是在括号中排前地父类的方法
```

执行以上程序输出结果为：

```
1. ShowMeAI是一个演说家，今天ta演讲的主题是 Python
```

方法重写

如果你的父类方法的功能不能满足你的需求，你可以在子类重写你父类的方法，实例如下：

```
1. class Parent:      # 定义父类
2.     def my_method(self):
3.         print ('调用父类方法')
4.
5. class Child(Parent): # 定义子类
6.     def my_method(self):
7.         print ('调用子类方法')
8.
9. c = Child()         # 子类实例
10. c.my_method()       # 子类调用重写方法
11. super(Child,c).my_method() #用子类对象调用父类已被覆盖的方法
```

super()函数是用于调用父类(超类)的一个方法。

执行以上程序输出结果为：

```
1. 调用子类方法
2. 调用父类方法
```

类属性与方法

类的私有属性

__private_attrs：由两个下划线开头，声明为私有属性，不能在类的外部被使用或直接访问。在类内部的方法中可以使用，使用方法为 **self.__private_attrs**。

类的方法

在类的内部定义的成员方法，必须包含参数 **self**，且为第一个参数，**self** 代表的是类的实例。

self 的名字并不是规定死的，也可以使用 **this**，但建议还是按照约定使用 **self**。

类的私有方法

__private_method：由两个下划线开头，声明为私有方法，只能在类的内部调用，使用方法为 **self.__private_methods**。

类的私有属性示例代码如下：

```
1. class NewCountry:
```



```
1. class NewCounter:
2.     __secret_count = 0 # 私有变量
3.     public_count = 0  # 公开变量
4.
5.     def count(self):
6.         self.__secret_count += 1
7.         self.public_count += 1
8.         print (self.__secret_count)
9.
10. counter = NewCounter()
11. counter.count()
12. counter.count()
13. print (counter.public_count)
14. print (counter.__secret_count) # 报错，实例不能访问私有变量
```

执行以上程序输出结果为：

```
1. 1
2. 2
3. 2
4. Traceback (most recent call last):
5.   File "test.py", line 16, in <module>
6.     print (counter.__secret_count) # 报错，实例不能访问私有变量
7. AttributeError: 'NewCounter' object has no attribute '__secret_count'
```

类的私有方法实例如下：

```
1. class WebSite:
2.     def __init__(self, name, url):
3.         self.name = name    # public
4.         self.__url = url    # private
5.
6.     def who(self):
7.         print('name : ', self.name)
8.         print('url : ', self.__url)
9.
10.    def __foo(self):        # 私有方法
11.        print('这是私有方法')
12.
13.    def foo(self):          # 公共方法
14.        print('这是公共方法')
15.        self.__foo()
16.
17. x = WebSite('ShowMeAI知识社区', 'www.showmeai.tech')
18. x.who()                  # 正常输出
19. x.foo()                  # 正常输出
20. x.__foo()                # 报错
```

类的专有方法

- `__init__`：构造函数，在生成对象时调用
- `__del__`：析构函数，释放对象时使用
- `__repr__`：打印，转换
- `__setitem__`：按照索引赋值
- `__getitem__`：按照索引获取值
- `__len__`：获得长度
- `__cmp__`：比较运算
- `__call__`：函数调用
- `__add__`：加运算
- `__sub__`：减运算
- `__mul__`：乘运算
- `__truediv__`：除运算
- `__mod__`：求余运算
- `__pow__`：乘方

运算符重载

Python同样支持运算符重载，我们可以对类的专有方法进行重载，实例如下：

```
1. class MyVector:
2.     def __init__(self, a, b):
3.         self.a = a
4.         self.b = b
5.
6.     def __str__(self):
7.         return 'Vector (%d, %d)' % (self.a, self.b)
8.
9.     def __add__(self, other):
10.        return Vector(self.a + other.a, self.b + other.b)
11.
12. v1 = MyVector(2,10)
13. v2 = MyVector(5,-2)
14. print(v1 + v2)
```

以上代码执行结果如下所示：

```
1. Vector(7,8)
```

视频教程

也可以点击 [这里](#) 到B站查看有【中英字幕】的版本

【双语字幕+资料下载】Python 3...

00:00




一键运行所有代码

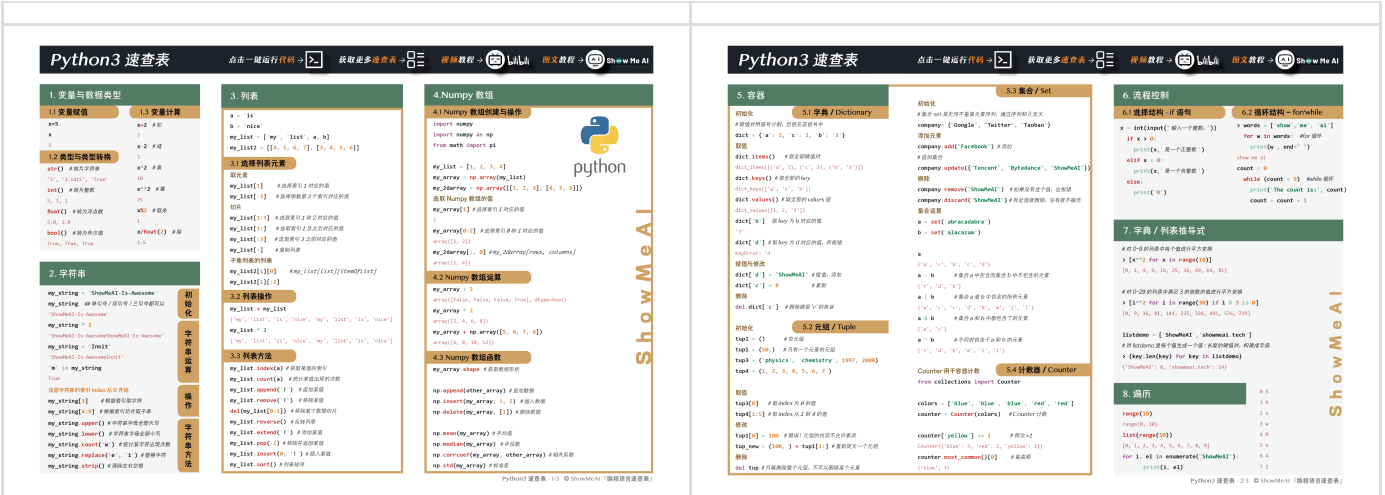
图解Python编程系列 配套的所有代码，可前往ShowMeAI 官方 **GitHub**，下载后即可在本地 Python 环境中运行。能访问 Google 的宝宝也可以直接借助 Google Colab一键运行与交互学习！

下载Python要点速查表

Awesome cheatsheets | **ShowMeAI**速查表大全 系列包含『编程语言』『AI技能知识』『数据科学工具库』『AI垂直领域工具库』四个板块，追平到工具库当前最新版本，并跑通了所有代码。点击 [官网](#) 或 [GitHub](#) 获取~

 ShowMeAI速查表大全

 Python 速查表（部分）



拓展参考资料

- Python教程 - Python3文档
- Python教程 - 廖雪峰的官方网站

ShowMeAI图解Python编程系列推荐（要点速查版）

- ShowMeAI 图解 Python 编程(1) | 介绍
- ShowMeAI 图解 Python 编程(2) | 安装与环境配置
- ShowMeAI 图解 Python 编程(3) | 基础语法
- ShowMeAI 图解 Python 编程(4) | 基础数据类型
- ShowMeAI 图解 Python 编程(5) | 运算符
- ShowMeAI 图解 Python 编程(6) | 条件控制与if语句
- ShowMeAI 图解 Python 编程(7) | 循环语句
- ShowMeAI 图解 Python 编程(8) | while循环
- ShowMeAI 图解 Python 编程(9) | for循环
- ShowMeAI 图解 Python 编程(10) | break语句
- ShowMeAI 图解 Python 编程(11) | continue语句
- ShowMeAI 图解 Python 编程(12) | pass语句
- ShowMeAI 图解 Python 编程(13) | 字符串及操作
- ShowMeAI 图解 Python 编程(14) | 列表
- ShowMeAI 图解 Python 编程(15) | 元组
- ShowMeAI 图解 Python 编程(16) | 字典
- ShowMeAI 图解 Python 编程(17) | 集合
- ShowMeAI 图解 Python 编程(18) | 函数
- ShowMeAI 图解 Python 编程(19) | 迭代器与生成器
- ShowMeAI 图解 Python 编程(20) | 数据结构
- ShowMeAI 图解 Python 编程(21) | 模块
- ShowMeAI 图解 Python 编程(22) | 文件读写
- ShowMeAI 图解 Python 编程(23) | 文件与目录操作
- ShowMeAI 图解 Python 编程(24) | 错误与异常处理
- ShowMeAI 图解 Python 编程(25) | 面向对象编程
- ShowMeAI 图解 Python 编程(26) | 命名空间与作用域
- ShowMeAI 图解 Python 编程(27) | 时间和日期

ShowMeAI系列教程精选推荐

- 大厂技术实现：推荐与广告计算解决方案
- 大厂技术实现：计算机视觉解决方案
- 大厂技术实现：自然语言处理行业解决方案
- 图解Python编程：从入门到精通系列教程
- 图解数据分析：从入门到精通系列教程
- 图解AI数学基础：从入门到精通系列教程
- 图解大数据技术：从入门到精通系列教程

- [图解机器学习算法：从入门到精通系列教程](#)
- [机器学习实战：手把手教你玩转机器学习系列](#)
- [深度学习教程：吴恩达专项课程 · 全套笔记解读](#)
- [自然语言处理教程：斯坦福CS224n课程 · 课程带学与全套笔记解读](#)
- [深度学习与计算机视觉教程：斯坦福CS231n · 全套笔记解读](#)

图解 Python 编程(25) | 面向对象编程

[< 上一篇](#)

[图解 Python 编程\(24\) | 错误与异常处理](#)

[下一篇 >](#)

[图解 Python 编程\(26\) | 命名空间与作用域](#)