

# 函数与面向对象编程

整理 by Orash  
For 2020 Summer Session

# 函数

函数是可重用的程序代码块。函数的作用，不仅可以实现代码的复用，更能实现代码的一致性。一致性指的是，只要修改函数的代码，则所有调用该函数的地方都能得到体现。

在编写函数时，函数体中的代码写法和我们前面讲述的基本一致，只是对代码实现了封装，并增加了函数调用、传递参数、返回计算结果等内容。

为了让大家更容易理解，掌握的更深刻。我们也要深入内存底层进行分析。绝大多数语言内存底层都是高度相似的，这样大家掌握了这些内容也便于以后学习其他语言。

## 5.1.1 函数的基本概念

1. 一个程序由一个个任务组成；函数就是代表一个任务或者一个功能。
2. 函数是代码复用的通用机制。

## 5.1.2 Python函数的分类

Python中函数分为如下几类：

1. 内置函数我们前面使用的str()、list()、len()等这些都是内置函数，我们可以拿来直接使用。
2. 标准库函数我们可以通过import语句导入库，然后使用其中定义的函数
3. 第三方库函数

Python社区也提供了很多高质量的库。下载安装这些库后，也是通过import语句导入，然后可以使用这些第三方库的函数。

4. 用户自定义函数 (\*)

用户自己定义的函数，显然也是开发中适应用户自身需求定义的函数。今天我们学习的就是如何自定义函数。

## 5.2.2 函数的定义和调用

### 核心要点

Python中，定义函数的语法如下：

```
def 函数名([参数列表]):  
    '''文档字符串'''  
    函数体/若干语句
```

要点：

1. 我们使用def来定义函数，然后就是一个空格和函数名称；  
Python执行def时，会创建一个函数对象，并绑定到函数名变量上。
2. 参数列表
  - (1) 圆括号内是形式参数列表，有多个参数则使用逗号隔开
  - (2) 形式参数不需要声明类型，也不需要指定函数返回值类型
  - (3) 无参数，也必须保留空的圆括号
  - (4) 实参列表必须与形参列表一一对应
3. return返回值
  - (1) 如果函数体中包含return语句，则结束函数执行并返回值；

(2) 如果函数体中不包含return语句，则返回None值。

4. 调用函数之前，必须要先定义函数，即先调用def创建函数对象

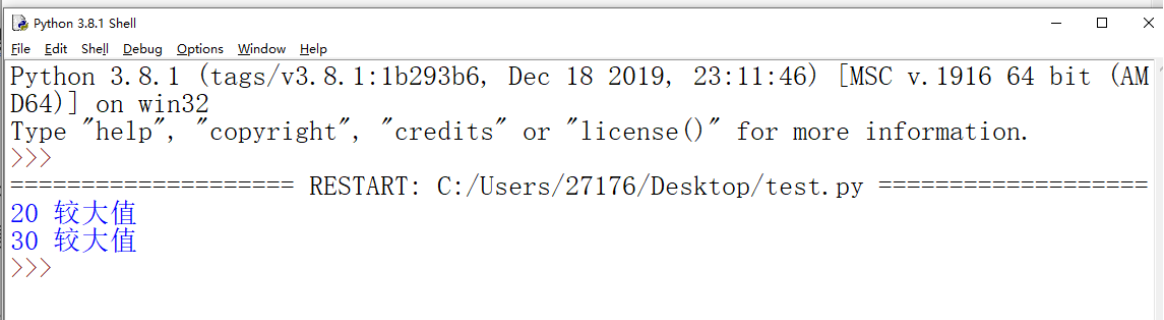
(1) 内置函数对象会自动创建

(2) 标准库和第三方库函数，通过import导入模块时，会执行模块中的def语句

## 形参和实参

咱们先来看个例子：

```
def printMax(a, b):  
    #实现两个数的比较，并返回较大的值  
    if a>b:  
        print(a, '较大值')  
    else:  
        print(b, '较大值')  
  
printMax(10, 20)  
printMax(30, 5)
```



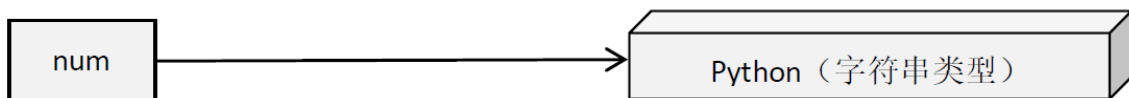
```
Python 3.8.1 Shell  
File Edit Shell Debug Options Window Help  
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:/Users/27176/Desktop/test.py =====  
20 较大值  
30 较大值  
>>>
```

上面的printMax函数中，在定义时写的printMax(a,b)。a和b称为“形式参数”，简称“形参”。也就是说，形式参数是在定义函数时使用的。形式参数的命名只要符合“标识符”命名规则即可。

在调用函数时，传递的参数称为“实际参数”，简称“实参”。上面代码中printMax(10,20)，10和20就是实际参数。

## 对象引用

变量是用来存储不同类型数据的，数据类型可以包括很多种，例如列表、数组、字符串、整数等。变量本身是没有任何数据类型的。例如，num = "python"，其中“python”是字符串类型，num 是对字符串类型的引用，num本身没有数据类型。学过C语言的同学应该知道指针的概念，原理类似，都是变量对某个数据类型的引用而已，如图所示。当num变量引用其他数据类型对象时，其引用的数据类型随之也发生变化，例如引用对象的数据类型可以是String、数值型、布尔型等。



## 返回值

return返回值要点：

1. 如果函数体中包含return语句，则结束函数执行并返回值；
2. 如果函数体中不包含return语句，则返回None值。
3. 要返回多个返回值，使用列表、元组、字典、集合将多个值“存起来”即可。

## 5.3函数也是对象，内存底层分析

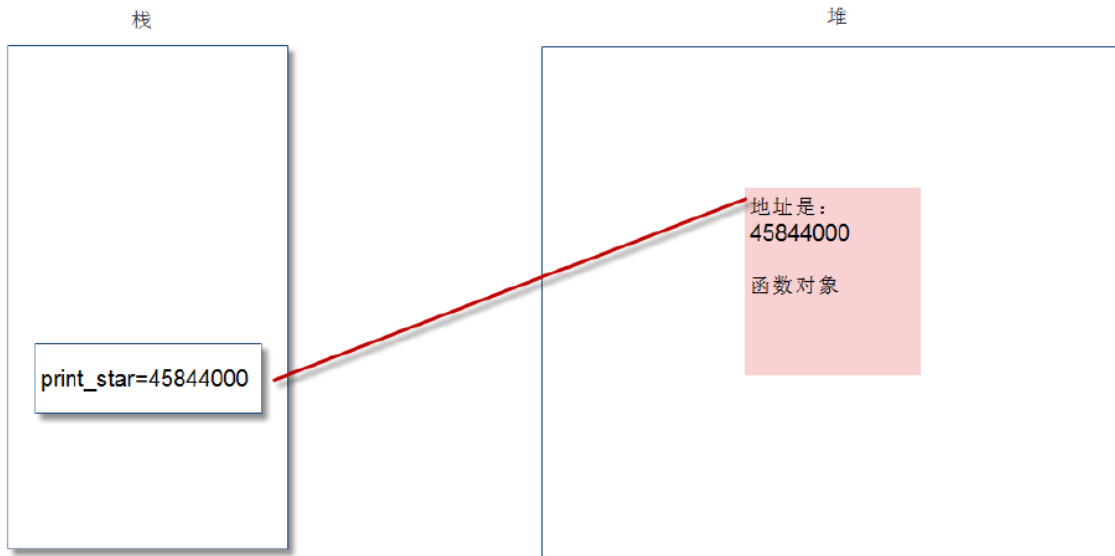
Python中，“一切都是对象”。实际上，执行def定义函数后，系统就创建了相应的函数对象。我们执行如下程序，然后进行解释：

```
1. def print_star(n):
2.     print("***n")
3.
4. print(print_star)
5. print(id(print_star))
6. c = print_star
7. c(3)
```

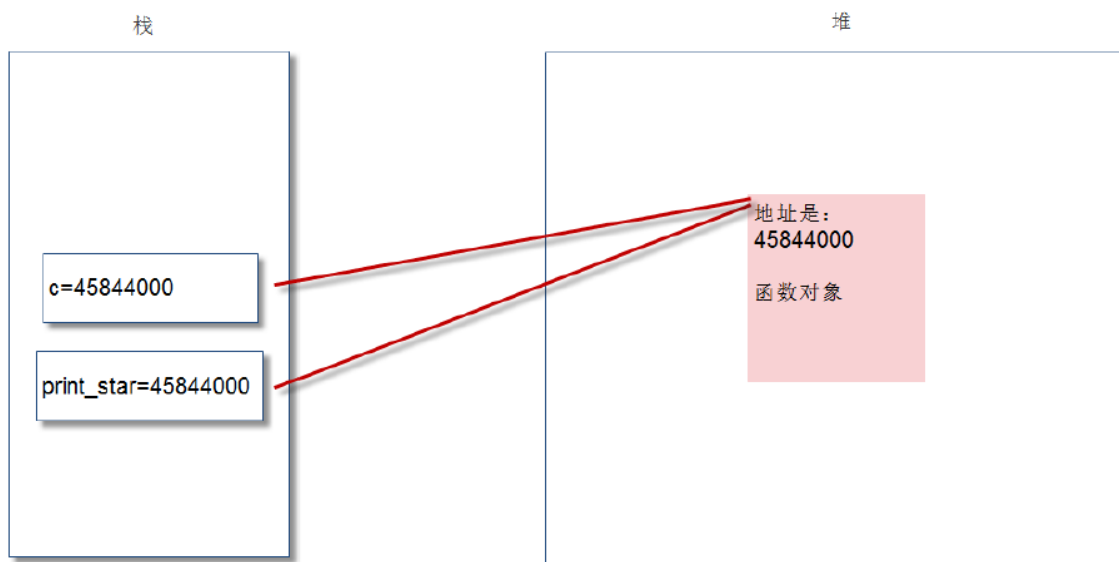
执行结果：

```
1. <function print_star at 0x000000002BB8620>
2. 45844000
3. ***
```

上面代码执行def时，系统中会创建函数对象，并通过print\_star这个变量进行引用：



我们执行“c=print\_star”后，显然将print\_star变量的值赋给了变量c，内存图变成了：



显然，我们可以看出变量c和print\_star都是指向了同一个函数对象。因此，执行c(3)和执行print\_star(3)的效果是完全一致的。Python中，圆括号意味着调用函数。在没有圆括号的情况下，Python会把函数当做普通对象。

与此核心原理类似，我们也可以做如下操作：

```
shixiang = int
shixiang("234")
```

显然，我们将内置函数对象int()赋值给了变量shixiang，这样shixiang和int都是指向了同一个内置函数对象。当然，此处仅限于原理性讲解，实际开发中没必要这么做。

## 5.4变量的作用域(全局变量和局部变量)

变量起作用的范围称为变量的作用域，不同作用域内同名变量之间互不影响。变量分为：全局变量、局部变量。

### 全局变量：

1. 在函数和类定义之外声明的变量。作用域为定义的模块，从定义位置开始直到模块结束。
2. 全局变量降低了函数的通用性和可读性。应尽量避免全局变量的使用。
3. 全局变量一般做常量使用。
4. 函数内要改变全局变量的值，可以使用global声明一下。

### 局部变量：

1. 在函数体中（包含形式参数）声明的变量。
2. 局部变量的引用比全局变量快，优先考虑使用。
3. 如果局部变量和全局变量同名，则在函数内全局变量用global声明一下，只使用同名的局部变量。

## \*5.5局部变量和全局变量效率测试

局部变量的查询和访问速度比全局变量快，优先考虑使用，尤其是在循环的时候。

在特别强调效率的地方或者循环次数较多的地方，可以通过将全局变量转为局部变量提高运行速度。

```
1. import math
2. import time
3. def test01():
4.     start = time.time()
5.     for i in range(10000000):
6.         math.sqrt(30)
7.     end = time.time()
8.     print("耗时{0}".format((end-start)))
9. def test02():
10.    b = math.sqrt
11.    start = time.time()
12.    for i in range(10000000):
13.        b(30)
14.    end = time.time()
15.    print("耗时{0}".format((end-start)))
16. test01()
17. test02()
```

## 5.6 参数的传递

函数的参数传递本质上就是：从实参到形参的赋值操作。所以，Python中参数的传递都是“引用传递”，不是“值传递”。具体操作时分为两类：

1. 对“可变对象”进行“写操作”，直接作用于原对象本身。
2. 对“不可变对象”进行“写操作”，会产生一个新的“对象空间”，并用新的值填充这块空间。（起到其他语言的“值传递”效果，但不是“值传递”）

可变对象有：

字典、列表、集合、自定义的对象等

不可变对象有：

数字、字符串、元组、function等

### 5.6.1 传递可变对象的引用

传递参数是可变对象（例如：列表、字典、自定义的其他可变对象等），实际传递的还是对象的引用。在函数体中不创建新的对象拷贝，而是可以直接修改所传递的对象。

```
1. a = 100
2. def f1(n):
3.     print("n:", id(n))          #传递进来的是 a 对象的地址
4.     n = n+200                  #由于 a 是不可变对象，因此创建新的对象 n
5.     print("n:", id(n))          #n 已经变成了新的对象
6.     print(n)
7. f1(a)
8. print("a:", id(a))
```

n: 1663816464

n: 46608592

300

### 5.6.3 浅拷贝和深拷贝

为了更深入的了解参数传递的底层原理，我们需要讲解一下“浅拷贝和深拷贝”。我们可以使用内置函数：copy(浅拷贝)、deepcopy(深拷贝)。

浅拷贝：不拷贝子对象的内容，只是拷贝子对象的引用。

深拷贝：会连子对象的内存也全部拷贝一份，对子对象的修改不会影响源对象。

```
1. import copy
2. def testCopy():
3.     #测试浅拷贝
4.     a = [10, 20, [5, 6]]
5.     b = copy.copy(a)
6.
7.     print("a", a)
8.     print("b", b)
9.     b.append(30)
10.    b[2].append(7)
11.    print("浅拷贝.....")
12.    print("a", a)
13.    print("b", b)
14.
15. def testDeepCopy():
16.     #测试深拷贝
17.     a = [10, 20, [5, 6]]
18.     b = copy.deepcopy(a)
19.     print("a", a)
```

```

20.     print("b", b)
21.     b.append(30)
22.     b[2].append(7)
23.     print("深拷贝.....")
24.     print("a", a)
25.     print("b", b)
26.
27. testCopy()
28. print("*****")
29. testDeepCopy()

```

## 5.7 参数的几种类型

### 位置参数

函数调用时，实参默认按位置顺序传递，需要个数和形参匹配。按位置传递的参数，称为：“位置参数”。

### 默认值参数

我们可以为某些参数设置默认值，这样这些参数在传递时就是可选的。称为“默认值参数”。默认值参数放到位置参数后面。

```

8 9 10 20
8 9 19 20
8 9 19 29

```

test.py - C:/Users/27176/Desktop/test.py (3.8.1)

File Edit Format Run Options Window Help

```

def f1(a, b, c=10, d=20):    #默认值参数必须位于普通位置参数后面
    print(a, b, c, d)
f1(8, 9)
f1(8, 9, 19)
f1(8, 9, 19, 29)

```

### 命名参数

我们也可以按照形参的名称传递参数，称为“命名参数”，也称“关键字参数”。

```

8 9 19
20 30 10
>>>

```

```

def f1(a, b, c):
    print(a, b, c)
f1(8, 9, 19)           #位置参数
f1(c=10, a=20, b=30)  #命名函数

```

## 5.8 嵌套函数(内部函数)

嵌套函数：

在函数内部定义的函数！

一般在什么情况下使用嵌套函数？

1. 封装-数据隐藏

外部无法访问“嵌套函数”。

2. 贯彻DRY(Don't Repeat Yourself)原则

嵌套函数，可以让我们在函数内部避免重复代码。

3. 闭包

后面会详细讲解。

## 5.9 nonlocal关键字

nonlocal用来声明外层的局部变量。

global用来声明全局变量。

## 5.10 LEGB规则

Python在查找“名称”时，是按照LEGB规则查找的：

Local-->Enclosed-->Global-->Built in

Local	指的就是函数或者类的方法内部
Enclosed	指的是嵌套函数（一个函数包裹另一个函数，闭包）
Global	指的是模块中的全局变量
Built in	指的是Python为自己保留的特殊名称。

如果某个name映射在局部(local)命名空间中没有找到，接下来就会在闭包作用域(enclosed)进行搜索，如果闭包作用域也没有找到，Python就会到全局(global)命名空间中进行查找，最后会在内建(built-in)命名空间搜索（如果一个名称在所有命名空间都没有找到，就会产生一个NameError）。



# Python内功-面向对象思想

## 6.1 面向对象简介

面向对象 (Object oriented Programming, OOP) 编程的思想主要是针对大型软件设计而来的。面向对象编程使程序的扩展性更强、可读性更好，使的编程可以像搭积木一样简单。

面向对象编程将数据和操作数据相关的方法封装到对象中，组织代码和数据的方式更加接近人的思维，从而大大提高了编程的效率。

Python完全采用了面向对象的思想，是真正面向对象的编程语言，完全支持面向对象的基本功能，例如：继承、多态、封装等。

Python中，一切皆对象。我们在前面学习的数据类型、函数等，都是对象。

\*Python是一种面向对象的高级程序设计语言。面向对象的概念在C++、Java语言中体现的淋漓尽致，如果你没有学习过Java等面向对象语言，也不要太紧张，我们把你按照零基础来考虑的。本节会涉及面向对象最基础的概念，后续章节会对Python面向对象进一步深入的展开。

注：Python支持面向过程、面向对象、函数式编程等多种编程范式。

### 1. 类

类定义了事物的属性和行为。面向对象术语中，成员函数称为方法，成员变量称为属性。类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。

### 2. 对象

类是对一类事物的归纳抽象，这类事物具有共同的属性与行为。从抽象到具体，引出了对象的概念。对象是类的实例。一个类可以产生无数个对象。例如现实生活当中，同一个班级当中女生是一类人群的总称，具体到对象个体，有张红、翠花等等不同的女生。

## 6.2 面向对象和面向过程区别

### •面向过程(Procedure Oriented)思维

面向过程编程更加关注的是“程序的逻辑流程”，是一种“执行者”思维，适合编写小规模

的程序。  
面向过程思想思考问题时，我们首先思考“怎么按步骤实现？”并将步骤对应成方法，一步一步，最终完成。这个适合简单任务，不需要过多协作的情况下。比如，如何开车？我们很容易就列出实现步骤：

1. 发动车
2. 挂挡
3. 踩油门
4. 走你

面向过程适合简单、不需要协作的事务。但是当我们思考比较复杂的问题，比如“如何造车？”，就会发现列出1234这样的步骤，是不可能的。那是因为，造车太复杂，需要很多协作才能完成。此时面向对象思想就应运而生了。

### •面向对象(Object Oriented)思维

面向对象更加关注的是“软件中对象之间的关系”，是一种“设计者”思维，适合编写大规模的程序。

面向对象(Object)思想更契合人的思维模式。我们首先思考的是“怎么设计这个事物？”比如思考造车，我们会先思考“车怎么设计？”，而不是“怎么按步骤造车的问题”。这就是思维方式的转变。

**面向对象方式思考造车，发现车由如下对象组成：**

1. 轮胎 2. 发动机 3. 车壳 4. 座椅 5. 挡风玻璃

为了便于协作，我们找轮胎厂完成制造轮胎的步骤，发动机厂完成制造发动机的步骤；这样，发现大家可以同时进行车的制造，最终进行组装，大大提高了效率。但是，具体到轮胎厂的一个流水线操作，仍然是有步骤的，还是离不开面向过程思想！

因此，面向对象可以帮助我们宏观上把握、从整体上分析整个系统。但是，具体到实现部分的微观操作（就是一个个方法），仍然需要面向过程的思路去处理。

我们千万不要把面向过程和面向对象对立起来。他们是相辅相成的。面向对象离不开面向过程！

### 面向对象思考方式

遇到复杂问题，先从问题中找名词（面向过程更多的是找动词），然后确立这些名词哪些可以作为类，再根据问题需求确定的类的属性和方法，确定类之间的关系。

\*中美如何面对处理这次疫情

## 6.3 对象的进化

随着编程面临的问题越来越复杂，编程语言本身也在进化，从主要处理简单数据开始，随着数据变多进化“数组”；数据类型变复杂，进化出了“结构体”；处理数据的方式和逻辑变复杂，进化出了“对象”。

### 1. 简单数据

像30,40, 50.4等这些数字，可以看做是简单数据。最初的计算机编程，都是像这样的数字。

### 2. 数组

将同类型的数据放到一起。比如：整数数组[20,30,40]，浮点数数组[10.2, 11.3, 12.4]，字符串数组：["aa","bb","cc"]

### 3. 结构体

将不同类型的数据放到一起，是C语言中的数据结构。比如：

```
1. struct resume{
2.     int age;
3.     char name[10];
4.     double salary;
5. };
```

### 4. 对象

将不同类型的数据、方法（即函数）放到一起，就是对象。比如：

```
1. class Student:
2.     school = "JDFZ"
3.     count = 0
4.     def __init__(self,name,score):
```

```

5.         self.name = name
6.         self.score = score
7.         Student.count = Student.count + 1
8.     def say_score(self):
9.         print("我的学校是: ", Student.school)
10.        print(self.name, "的分数是", self.score)

```

我们前面学习的数字也是对象。比如：整数9，就是一个包含了加法、乘法等方法的对象。

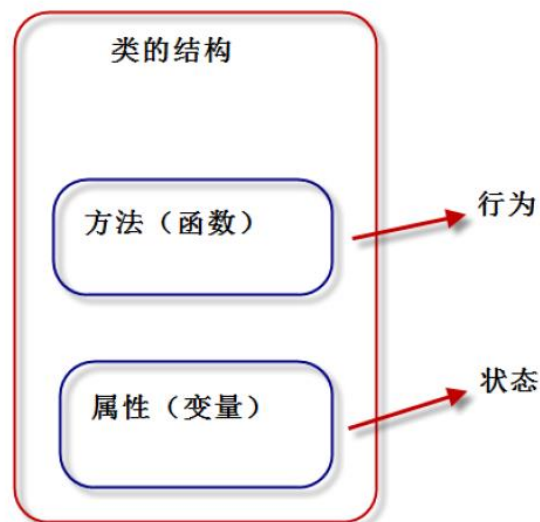
## 6.4 类的定义

我们把对象比作一个“饼干”，类就是制造这个饼干的“模具”。

我们通过类定义数据类型的属性（数据）和方法（行为）

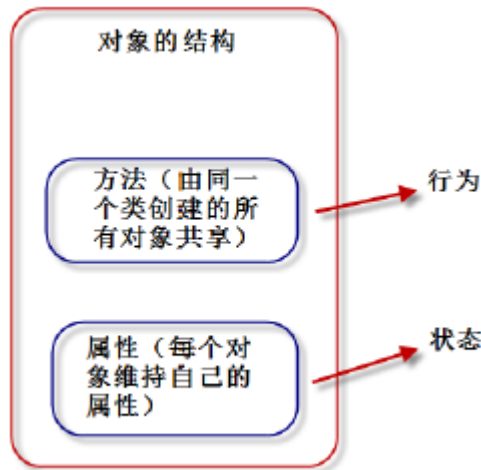


我们通过类定义数据类型的属性（数据）和方法（行为），也就是说，“类将行为和状态打包在一起”。



对象是类的具体实体，一般称为“类的实例”。类看做“饼干模具”，对象就是根据这个“模具”制造出的“饼干”。

从一个类创建对象时，每个对象会共享这个类的行为（类中定义的方法），但会有自己的属性值（不共享状态）。更具体一点：“方法代码是共享的，属性数据不共享”。



Python中，“一切皆对象”。类也称为“类对象”，类的实例也称为“实例对象”。

定义类的语法格式如下：

```
class 类名:
```

类体

要点如下：

1. 类名必须符合“标识符”的规则；一般规定，首字母大写，多个单词使用“驼峰原则”。
2. 类体中我们可以定义属性和方法。
3. 属性用来描述数据，方法(即函数)用来描述这些数据相关的操作。

## 6.5 \_\_init\_\_ 构造方法和 \_\_new\_\_ 方法

类是抽象的，也称之为“对象的模板”。我们需要通过类这个模板，创建类的实例对象，然后才能使用类定义的功能。

我们前面说过一个Python对象包含三个部分：id (identity识别码)、type (对象类型)、value (对象的值)。

现在，我们可以更进一步的说，一个Python对象包含如下部分：

1. id (identity识别码)
2. type (对象类型)
3. value (对象的值)
  - (1) 属性 (attribute)
  - (2) 方法 (method)

创建对象，我们需要定义构造函数\_\_init\_\_()方法。构造方法用于执行“实例对象的初始化工作”，即对象创建后，初始化当前对象的相关属性，无返回值。

\_\_init\_\_()的要点如下：

1. 名称固定，必须为：\_\_init\_\_()
2. 第一个参数固定，必须为：self。self指的就是刚刚创建好的实例对象。
3. 构造函数通常用来初始化实例对象的实例属性，如下代码就是初始化实例属性：name和score。
4. 通过“类名(参数列表)”来调用构造函数。调用后，将创建好的对象返回给相应的变量。

比如: `s1 = Student('张三', 80)`

5. `__init__()`方法: 初始化创建好的对象, 初始化指的是: “给实例属性赋值”

6. `__new__()`方法: 用于创建对象, 但我们一般无需重定义该方法。

7. 如果我们不定义`__init__`方法, 系统会提供一个默认的`__init__`方法。如果我们定义了带参的`__init__`方法, 系统不创建默认的`__init__`方法。

注: Python中的`self`相当于C++中的`self`指针, JAVA和C#中的`this`关键字。Python中, `self`必须为构造函数的第一个参数, 名字可以任意修改。但一般遵守惯例, 都叫做`self`。

## 6.6 实例属性和实例方法

### 实例属性

实例属性是从属于实例对象的属性, 也称为“实例变量”。他的使用有如下几个要点:

1. 实例属性一般在`__init__()`方法中通过如下代码定义: `self.实例属性名 = 初始值`
2. 在本类的其他实例方法中, 也是通过`self`进行访问:

`self.实例属性名`

3. 创建实例对象后, 通过实例对象访问:

`obj01 = 类名()`      #创建对象, 调用`__init__()`初始化属性

`obj01.实例属性名 = 值` #可以给已有属性赋值, 也可以新加属性

### 实例方法

实例方法是从属于实例对象的方法。实例方法的定义格式如下:

`def 方法名(self[, 形参列表]):`

函数体

方法的调用格式如下:

`对象.方法名([实参列表])`

要点:

1. 定义实例方法时, 第一个参数必须为`self`。和前面一样, `self`指当前的实例对象。
2. 调用实例方法时, 不需要也不能给`self`传参。`self`由解释器自动传参。

#### •函数和方法的区别

1. 都是用来完成一个功能的语句块, 本质一样。
2. 方法调用时, 通过对象来调用。方法从属于特定实例对象, 普通函数没有这个特点。
3. 直观上看, 方法定义时需要传递`self`, 函数不需要。

#### •实例对象的方法调用本质:

`a = Student()`

解释器翻译:

`a.say_score()`



`Student.say_score(a)`

其他操作:

1. `dir(obj)`可以获得对象的所有属性、方法
2. `obj.__dict__` 对象的属性字典
3. `pass` 空语句
4. `isinstance (对象,类型)` 判断“对象”是不是“指定类型”

## 6.7 类对象、类属性、类方法

### 类对象

我们在前面讲的类定义格式中，“`class 类名:`”。实际上，当解释器执行`class`语句时，就会创建一个类对象。

### 类属性

类属性是从属于“类对象”的属性，也称为“类变量”。由于，类属性从属于类对象，可以被所有实例对象共享。

类属性的定义方式：

```
class 类名:
    类变量名 = 初始值
```

在类中或者类的外面，我们可以通过：“类名.类变量名”来读写。

### 类方法

类方法是从属于“类对象”的方法。类方法通过装饰器`@classmethod`来定义，格式如下：

`@classmethod`

```
def 类方法名(cls [, 形参列表]) :
    函数体
```

要点如下：

1. `@classmethod`必须位于方法上面一行
2. 第一个`cls`必须有；`cls`指的就是“类对象”本身；
3. 调用类方法格式：“类名.类方法名(参数列表)”。参数列表中，不需要也不能给`cls`传值。
4. 类方法中访问实例属性和实例方法会导致错误
5. 子类继承父类方法时，传入`cls`是子类对象，而非父类对象