# **字符串**

## 格式化

### format()

另一种格式化字符串的方法是使用字符串的format()方法，它会用传入的参数依次替换字符串内的占位符{0}、{1}……，不过这种方式写起来比%要麻烦得多：

>>> 'Hello, {0}, 成绩提升了 {1:.1f}%'.format('小明', 17.125)

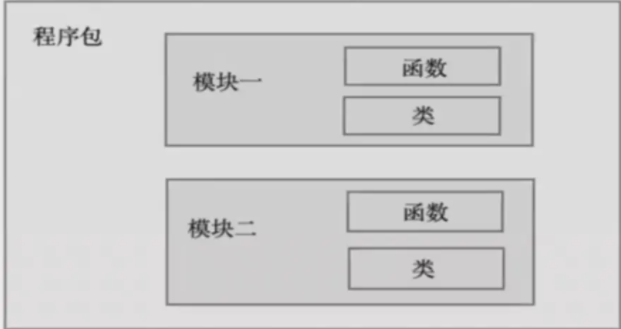
'Hello, 小明, 成绩提升了 17.1%'

# **模块**

## **Python的程序结构**

组成：

* package 包
* module 模块
* function 方法



#### 模块定义

* 包含Python定义和语句的文件
* .py 文件
* 可作为脚本运行

#### 不同方式导入模块

* import 模块名

使用模块内的方法、变量、类时，需要在<方法 | 变量 | 类>前面加上模块名，如：time.sleep()

* from <模块名> import <方法 | 变量 | 类>

可以直接调用<方法 | 变量 | 类>，如：sleep()

* from <模块名> import \*，可以直接调用该模块的所有<方法 | 变量 | 类>  
  - 同一个模块写多次，只被导入一次  
  - import应该放在代码的顶端

#### 模块分类

* 系统内置模块
* 第三方的开源模块
* 自定义模块

#### 如何使用模块

* 系统内置模块

Python安装好之后自带的一些非常有用的模块（sys、os、time、json模块等）

* 第三方开源模块

通过包管理工具pip完成，在命令行中输入pip install <模块名>；pycharm里面可以在setting->project->Python Interpreter->+输入模块名进行下载

## if \_\_name\_\_ == '\_\_main\_\_' 如何正确理解?

1. \_\_name\_\_是魔法函数的一种；

魔法函数相关的知识点击[这里](https://zhuanlan.zhihu.com/p/344951719)。

* 引言

学过Java、C、C++的程序员应该都知道，每次开启一个程序，都必须写一个主函数作为程序的入口，也就是我们常说的main函数。如下所示， main()就是Java中的一个main函数。

public class HelloWorld {

public static void main(String[] args) {

System.out.println("HelloWorld");

}

}

与Java、C、C++等几种语言不同的是，Python是一种解释型脚本语言，在执行之前不需要将所有代码先编译成中间代码，Python程序运行时是从模块顶行开始，逐行进行翻译执行，所以，最顶层（没有被缩进）的代码都会被执行，所以Python中并不需要一个统一的main()作为程序的入口。在某种意义上讲，“if \_\_name\_\_==’\_\_main\_\_:”也像是一个标志，象征着Java等语言中的程序主入口，告诉其他程序员，代码入口在此——这是“if \_\_name\_\_==’\_\_main\_\_:”这条代码的意义之一。

### 1. \_\_name\_\_的理解

#### 1.1 为什么使用\_\_name\_\_属性？

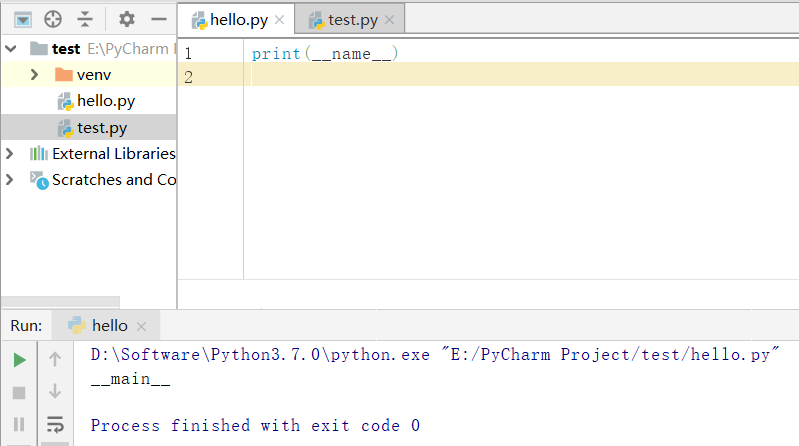
Python解释器在导入模块时，会将模块中没有缩进的代码全部执行一遍（模块就是一个独立的Python文件）。开发人员通常会在模块下方增加一些测试代码，为了避免这些测试代码在模块被导入后执行，可以利用\_\_name\_\_属性。

#### 1.2 什么是\_\_name\_\_属性。

\_\_name\_\_属性是Python的一个内置属性，记录了一个字符串。

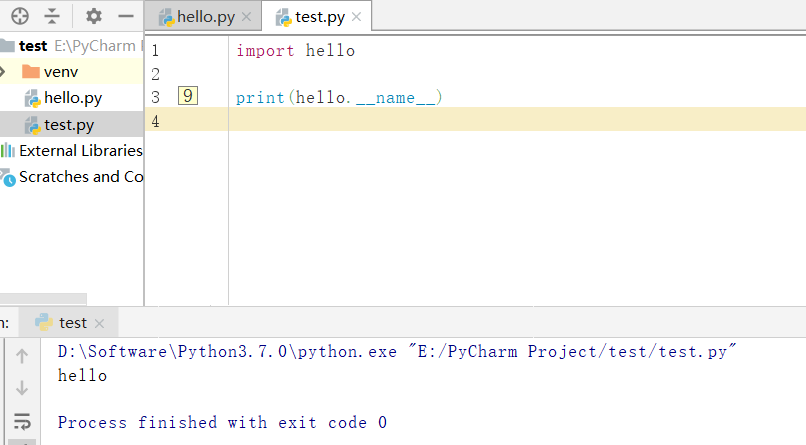
* 若是在当前文件，\_\_name\_\_ 是\_\_main\_\_

在hello文件中打印本文件的\_\_name\_\_属性值，显示的是\_\_main\_\_



* 若是导入的文件，\_\_name\_\_是模块名

test文件导入hello模块，在test文件中打印出hello模块的\_\_name\_\_属性值，显示的是hello模块的模块名。



因此\_\_name\_\_ == '\_\_main\_\_' 就表示在当前文件中，可以在if \_\_name\_\_ == '\_\_main\_\_':条件下写入测试代码，如此可以避免测试代码在模块被导入后执行。

### 2. 模块导入

我们知道，当我们把模块A中的代码在模块B中进行import A时，只要B模块代码运行到该import语句，模块A的代码会被执行。

模块A：

# 模块A

a = 100

print('你好，我是模块A……')

print(a)

模块B：

# 模块B

from package01 import A

b = 200

print('你好，我是模块B……')

print(b)

运行模块B时，输出结果如下：

你好，我是模块A……

100

你好，我是模块B……

200

如果在模块A中，我们有部分的代码不想在被导入到B时直接被运行，但在直接运行A时可直接运行，那该怎么做呢？那就可以用到“if \_\_name\_\_==’\_\_main\_\_:”这行代码了，我们队上面用到的A模块代码进行修改：

A模块代码修改为：

|  |
| --- |
| # 模块A  a = 100  print('你好，我是模块A……')  if \_\_name\_\_=='\_\_main\_\_':  print(a) |

B模块不做修改，直接执行B模块，输出结果如下：

|  |
| --- |
| 你好，我是模块A……  你好，我是模块B……  200 |

看到了吗，A模块中的a的值就没有再被输出了。所以，当你要导入某个模块，但又不想改模块的部分代码被直接执行，那就可以这一部分代码放在“if \_\_name\_\_=='\_\_main\_\_':”内部。

### 3.“\_\_name\_\_”与“\_\_main\_\_”

看到现在也许心中还是疑惑，那么现在我们来说一说“if\_\_name\_\_=='\_\_main\_\_':”的原理。

1. “\_\_name\_\_”是Python的内置变量，用于指代当前模块。我们修改上面用到的A模块和B模块，在模块中分别输出模块的名称：

模块A：

|  |
| --- |
| # 模块A  print('你好，我是模块A……')  print('模块A中\_\_name\_\_的值：{}'.format(\_\_name\_\_))  print('-------------------------') |

模块B：

|  |
| --- |
| # 模块B  from package01 import A  print('你好，我是模块B……')  print('模块B中\_\_name\_\_的值：{}'.format(\_\_name\_\_)) |

执行A模块时，输出结果：

|  |
| --- |
| 你好，我是模块A……  模块A中\_\_name\_\_的值：\_\_main\_\_  ------------------------- |

执行B模块时，输出结果：

|  |
| --- |
| 你好，我是模块A……  模块A中\_\_name\_\_的值：package01.A  -------------------------  你好，我是模块B……  模块B中\_\_name\_\_的值：\_\_main\_\_ |

发现神奇之处了吗？当哪个模块被直接执行时，该模块“\_\_name\_\_”的值就是“\_\_main\_\_”，当被导入另一模块时，“\_\_name\_\_”的值就是模块的真实名称。用一个类比来解释一下：记得小时候要轮流打算教室，轮到自己的时候（模块被直接执行的时候），我们会说今天是“我”（\_\_main\_\_）值日，称呼其他人时，我们就会直接喊他们的名字。所以，“\_\_main\_\_”就相当于当事人，或者说第一人称的“我”。

所以，当运行“if \_\_name\_\_=='\_\_main\_\_':”语句时，如果当前模块时被直接执行，\_\_name\_\_的值就是\_\_main\_\_，条件判断的结果为True，“if \_\_name\_\_=='\_\_main\_\_':”下面的代码块就会被执行。

# 面向对象编程

1.类的定义

|  |
| --- |
| class Student(object):  pass print(Student) *#<class '\_\_main\_\_.Student'>* |

(1)class后面紧接着是类名，即Student，类名通常是大写开头的单词  
(2)类名后面紧接着是(object)，表示该类是从哪个类继承下来的,通常，如果没有合适的继承类，就使用object类，这是所有类最终都会继承的类。如果是object可以省略（object）

(3)当一个类定义完之后，就产生了一个类对象。类对象支持两种操作：引用和实例化。

引用操作是通过类对象去调用类中的属性或者方法;

实例化是产生出一个类对象的实例，称作实例对象。比如定义了一个people类：

2.创建实例

|  |
| --- |
| # 创建实例是通过类名+()实现的 bart = Student() print(bart) #<\_\_main\_\_.Student object at 0x0000020685207490> |

3.实例对象动态绑定属性

|  |
| --- |
| bart.name = 'Bart Simpson' print(bart.name) |

在类中可以不声明属性，在对象创建好之后，给对象绑定属性。

4.实例对象静态绑定属性  
1.每个实例的必要属性需要在定义类的时候绑定  
2.通过定义一个特殊的\_\_init\_\_方法，在创建实例的时候，就把name，score等属性绑上去：  
3.\_\_init\_\_方法的第一个参数永远是self，表示创建的实例本身，因此，在\_\_init\_\_方法内部，就可以把各种属性绑定到self，因为self就指向创建的实例本身。

|  |
| --- |
| class Student(object):  def \_\_init\_\_(self, name, score):  self.name = name  self.score = score |

5.定义了\_\_init\_\_方法的类创建实例

当类中有了\_\_init\_\_方法，在创建实例的时候，就不能传入空的参数了，必须传入与\_\_init\_\_方法匹配的参数，但self不需要传，Python解释器自己会把实例变量传进去：

|  |
| --- |
| bart = Student('Bart Simpson', 59) print(bart.name) print(bart.score) |

6.类中的函数

|  |
| --- |
| (1)和普通的函数相比，在类中定义的函数只有一点不同，就是第一个参数永远是实例变量self, 但是在调用时，不用传递该参数。  除此之外，类的方法和普通函数没有什么区别，所以，你仍然可以用默认参数、可变参数、关键字参数和命名关键字参数。 |