

函数

学习目标

1. 函数的快速体验
2. 函数的基本使用
3. 函数的参数
4. 函数的返回值
5. 函数的嵌套调用
6. 在模块中定义函数

一、函数的快速体验

所谓**函数**，就是把 **具有独立功能的代码块** 组织为一个小模块，在需要的时候 **调用**

函数的使用包含两个步骤：

- 1 定义函数 —— **封装** 独立的功能
- 2 调用函数 —— 享受 **封装** 的成果

函数的作用，在开发程序时，使用函数可以提高编写的效率以及代码的 **重用**

举例：多次使用九九乘法表

二、函数的基本使用

2.1 定义函数的格式如下：

```
def 函数名():
```

```
    函数封装的代码
```

1. **def** 是英文 **define** 的缩写
2. **函数名称** 应该能够表达 **函数封装代码** 的功能，方便后续的调用
3. 函数名称 的命名应该 **符合 标识符的命名规则**

可以由 字母、下划线 和 数字 组成

不能以数字开头

不能与关键字重名

2.2 函数的调用

调用函数很简单的，通过 **函数名()** 即可完成对函数的调用

2.3 第一个函数演练

需求

编写一个打招呼 `say_hello` 的函数，封装三行打招呼的代码

在函数下方调用打招呼的代码

```
name = "小明"

# 解释器知道这里定义了一个函数

def say_hello():

    print("hello 1")

    print("hello 2")

    print("hello 3")

print(name)

# 只有在调用函数时，之前定义的函数才会被执行

# 函数执行完成之后，会重新回到之前的程序中，继续执行后续的代码

say_hello()

print(name)
```

注意：

定义好函数之后，只表示这个函数封装了一段代码而已

如果不主动调用函数，函数是不会主动执行的

思考：

问：能否将 函数调用 放在 **函数定义** 的上方？

不能！因为在 使用函数名 调用函数之前，必须要保证 Python 已经知道函数的存在

否则控制台会提示 `NameError: name 'say_hello' is not defined` (名称错误：**say_hello** 这个名字没有被定义)

2.4 函数的文档注释

1. 在开发中，如果希望给函数添加注释，应该在 **定义函数** 的下方，使用 **连续的三对引号**
2. 在 **连续的三对引号** 之间编写对函数的说明文字
3. 在 **函数调用** 位置，使用快捷键 **CTRL + Q** 可以查看函数的说明信息

注意：

因为 函数体相对比较独立，函数定义的上方，应该和其他代码（包括注释）保留 **两个空行**

三、函数参数

演练需求

开发一个 `sum_2_num` 的函数

函数能够实现 **两个数字的求和** 功能

演练代码如下：

```
def sum_2_num():  
  
    num1 = 10  
  
    num2 = 20  
  
    result = num1 + num2  
  
    print("%d + %d = %d" % (num1, num2, result))  
  
sum_2_num()
```

***思考一下当前代码存在的问题？

函数只能处理 **固定数值** 的相加

***如何解决？

如果能够把需要计算的数字，在调用函数时，传递到函数内部就好了！

3.1 函数参数的使用

在函数名的后面的小括号内部填写 **参数**

多个参数之间使用 **,** 分隔

```
def sum_2_num(num1, num2):  
  
    result = num1 + num2  
  
    print("%d + %d = %d" % (num1, num2, result))  
  
sum_2_num(50, 20)
```

****注意参数的作用：

函数，把 具有独立功能的代码块 组织为一个小模块，在需要的时候 调用。

函数的参数，增加函数的 通用性，针对 相同的数据处理逻辑，能够 适应更多的数据

在函数 内部，把参数当做 变量 使用，进行需要的数据处理

函数调用时，按照函数定义的**参数顺序**，把 希望在函数内部处理的数据，通过参数 传递

****形参和实参

形参：定义 函数时，小括号中的参数，是用来接收参数用的，在函数内部 作为变量使用

实参：调用 函数时，小括号中的参数，是用来把数据传递到 函数内部 用的

四、函数返回值

1. 在程序开发中，有时候，会希望 一个函数执行结束后，告诉调用者一个结果，以便调用者针对具体的结果做后续的处理

2. **返回值** 是函数 完成工作后，最后 给调用者的 一个结果

3. 在函数中使用 **return** 关键字可以返回结果

4. 调用函数一方，可以 使用变量 来 接收 函数的返回结果

注意：return 表示返回，后续的代码都不会被执行

```
def sum_2_num(num1, num2):  
  
    """对两个数字的求和"""  
  
    return num1 + num2  
  
# 调用函数，并使用 result 变量接收计算结果  
  
result = sum_2_num(10, 20)  
  
print("计算结果是 %d" % result)
```

五、函数的嵌套使用

1. 一个函数里面 又调用 了 另外一个函数，这就是 函数嵌套调用

2. 如果函数 test2 中，调用了另外一个函数 test1

那么执行到调用 test1 函数时，会先把函数 test1 中的任务都执行完
才会回到 test2 中调用函数 test1 的位置，继续执行后续的代码

```
def test1():
```

```
    print("*" * 50)
```

```
    print("test 1")
```

```
    print("*" * 50)
```

```
def test2():
```

```
    print("-" * 50)
```

```
    print("test 2")
```

```
    test1()
```

```
    print("-" * 50)
```

```
test2()
```

函数的嵌套演练：

需求 1

定义一个 print_line 函数能够打印 * 组成的 一条分隔线

```
def print_line():
```

```
    print("*" * 50)
```

需求 2

定义一个函数能够打印 由任意字符组成 的分隔线

```
def print_line(char):
```

```
    print(char * 50)
```

需求 3

定义一个函数能够打印 **任意重复次数** 的分隔线

```
def print_line(char, times):  
  
    print(char * times)
```

需求 4

定义一个函数能够打印 **5 行** 的分隔线，分隔线要求符合需求 3

```
def print_line(char, times):  
  
    print(char * times)  
  
def print_lines(char, times):  
  
    row = 0  
  
    while row < 5:  
  
        print_line(char, times)  
  
        row += 1
```

六、使用模块中的函数

模块是 **Python** 程序架构的一个核心概念

模块就好比是 **工具包**，要想使用这个工具包中的工具，就需要 **导入 import** 这个模块

每一个以扩展名 **.py** 结尾的 **Python** 源代码文件都是一个 **模块**

在模块中定义的 **全局变量**、**函数** 都是模块能够提供给外界直接使用的工具

第一个 模块演练

新建 **py_分隔线模块.py**

复制 **py_打印多条分隔线.py** 中的内容，最后一行 **print** 代码除外

增加一个字符串变量

总结：

1. 可以在一个 **Python** 文件中 **定义变量** 或者 **函数**
2. 然后在 **另外一个文件**中 使用 **import** 导入这个模块
3. 导入之后，就可以使用 **模块名.变量** / **模块名.函数** 的方式，使用这个模块中定义的变量或者函数

注意：模块名也是一个标识符

标识符可以由 字母、下划线 和 数字 组成

不能以数字开头

不能与关键字重名

注意：如果在给 Python 文件起名时，以数字开头 是无法在 PyCharm 中通过导入这个模块的