day03(元组和函数)

函数

1. 函数定义

```
def 函数名():

    函数封装的代码
    ……
```

函数的规范

1. def 是英文 define 的缩写
2. **函数名称** 应该能够简单明确的表达 **函数功能**，以方便后续的调用
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**

案例

```python
"""---------------------- 第一个函数的实现 ----------------------"""
# 定义函数
def say_hello():
    print("你好啊！")
# 调用
say_hello()
```

```python
"""---------------------- 第一个函数的实现 ----------------------
# 定义函数
def say_hello():
    """ 这是给玩家看的 """
    print("你好啊！")
# 调用
say_hello()
```

Ctrl+Q

```
D:/Python_project/day03/06第一个函数的实现.py
def say_hello() -> None

这是给玩家看的
```

函数注释

由于这些函数的函数名并不能完全的表示出函数的含义,所以我们在定义函数的时候就需要给函数加上特定的注释功能,让别人看到这个注释就知道这个函数是干什么的.

这个函数的注释就是**文档注释**.

- 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**
- 在 **连续的三对引号** 之间编写对函数的说明文字
- 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

2. 函数嵌套函数

- 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**
- 如果函数func2中，调用了另外一个函数func1
  - 那么执行到调用 func1 函数时，会先把函数 func1 中的任务都执行完
  - 才会回到 func2 中调用函数 func1 的位置，继续执行后续的代码

函数参数

形参和实参

```python
"""---------------------- 函数传参 ----------------------"""
def say(name):
    """name为传入字符参数"""
    print("hello%s"%name)

say("张海方")
"""---------------------- 函数参数有多个 ----------------------"""
def Sum(a,b):
    SUM = a + b
    print(SUM)

Sum(1,3)
# 参数在定义时和调用时 要保持一致
"""---------------------- 最值练习 ----------------------"""
def Max_num(a):
    print(max(a))

b = [1, 3, 8, 9, 4, 6, 7]
Max_num(b)
```

- 函数的**参数**，可以传递数据给**函数内部**
- 参数的作用是增加函数的 **通用性**

形参

全称为"**形式参数**",由于它不是实际存在变量，所以又称**虚拟变量。**

是在定义函数名和函数体的时候使用的参数,目的是用来**接收调用该函数时传入的参数**.

一句话概括**形式参数**：形参是函数被调用时用于接收实参值的变量。

实参

全称为"**实际参数**",是在调用时传递给函数的参数.

作用域

作用域（scope），程序设计概念，通常来说，一段程序代码中所用到的名字并不总是有效/可用的，而限定这个名字的可用性的代码范围就是这个名字的*作用域*。

- 形参的作用域**只在定义函数的代码块** 中，一旦超出该范围就不能再使用了
- 不同的函数中,可以有**同名的形参**,不会相互影响
- 作用域的使用提高了程序逻辑的局部性，增强程序的可靠性，减少名字冲突。
- 函数间的形参是不能相互使用的

函数返回值

```python
"""---------------------- 练习1 ----------------------"""
"""
写一个函数求三个参数的和
"""
def Sum(a):
    print(sum(a))

a = [1, 2, 3]
Sum(a)
"""---------------------- 练习2 ----------------------"""
"""
写一个函数求三个参数的和
"""
def Sum(a):
    print(sum(a)/len(a))

a = [1, 2, 3]
Sum(a)
```

1.概念

- 开发中，有时会希望 **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
- **返回值** 是 **函数给调用方提供的结果**

2.设置返回值

- 在函数中使用 return 关键字可以返回结果
- 调用函数一方，可以 **使用变量** 来 **接收** 函数的返回结果

```python
def num(a,b):
    return a + b, a - b

print(type(num(10, 20)))
print(num(10, 20))
```

```
09多个返回值
D:\Develop\Python38\python.ex
<class 'tuple'>
(30, -10)
```

- 函数可以有多个return,只能执行一个return
- 函数也可以有多个返回值

```
SSum, _ = num(10,20)
print(SSum)

09多个返回值
D:\Develop\Python38\python.ex
<class 'tuple'>
(30, -10)
30
```

对第二个返回值不关心，用于划线接收，程序不占用空间

```
"""
需求1
定义一个 print_line 函数能够打印 * 组成的 一条分隔线,长度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,长度50
需求3
定义一个函数能够打印 任意长度 的分隔线
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
"""
"""---------------------- 1. ----------------------"""
def print_line(l):
    while l > 0:
        print("*", end="")
        l -= 1

print_line(50)
print()
"""---------------------- 2. ----------------------"""
import random
def print_lx(l):
    while l > 0:
        print("%c"%random.randint(100, 200), end="")
        l -= 1
print_lx(50)
print()
"""---------------------- 4. ----------------------"""
def print_15(l):
    i = 5
    while i > 0:
        j = l
        while j > 0:
            print("|", end="")
            j -= 1
        print()
        i -= 1
print_15(20)
```

局部变量和全局变量

- **局部变量**，就是在 **函数内部定义的变量**
- 不同的函数，可以定义相同的名字的局部变量，但是各用个的不会产生影响
- 局部变量的作用域只在函数内部
- 局部变量的目的是 **存储需要临时保存的数据**

心得

选择结构心得：假设一个最小值，然后将这个假设的最小值与后面的数值进行比较，然后得到最小值的角标，之后，将得到的最小值角标对应的最小值设置成最小值就ok了

选择排序

1. 原理

```
          9   8   7   6   5   4   3   2   1
选择排序:
第一轮:假定最小值角标:0
第一次:     比较角标0和角标1元素  将最小值的角标设置为1
第二次:     继续和最小值进行比较  将最小值的角标设置为2
..
第八次:     获取到最小值的角标:8
第一轮结束之后: 交换认为的最小值和真正的最小值
          1   8   7   6   5   4   3   2   9
第一轮之后:确定了最小值

第二轮:假定最小值角标1
第一次:     比较角标1和角标2的元素   最小值角标是2
..
第七次:     找到最小值的角标7
第二轮结束:交换角标为2以及角标为7的元素

          1   2   7   6   5   4   3   8   9

...
第八轮:
第一次:
          1   2   3   4   5   6   7   8   9
```

2. 代码

```python
"""---------------------- 选择排序 ----------------------"""
ll = [3, 6, 5, 8, 9, 7, 4, 2, 1]
i = 0
while i < len(ll):
    # 假定一个最小值
    minData = i
    # 第二层循环变量
    j = i + 1
    while j < len(ll):
        # 判断假定最小值是否真的是最小值
        if ll[minData] > ll[j]:
            minData = j
        j += 1
    # 得到最小值角标后，交换
    # 假定的最小值是i,而实际得到的最小值是minData
    # 交换两变量的值
    temp = ll[i]
    ll[i] = ll[minData]
    ll[minData] = temp
    # 外层循环自增
    i += 1
print(ll)
```

元组

元组的定义

Python的元组与列表类似，不同之处在于元组的**元素不能修改**

- 元组用 () 定义
- 用于存储 一串 数据，元素之间使用 , 分隔
- 元组的 索引 从 0 开始
- 元组的类型:**tuple**

元组的操作

注意：处理修改的操作之外的操作都可以，查询和遍历可以

元组使用场景

- 作为自动组包的默认类型
- 自动解包
- 交换数据
- 格式化输出
- 让列表不可被修改,保护数据安全

```python
"""---------------------- 元组的使用场景 ----------------------"""
# 自动组包 把多个数据自动组成一个包
a = 1, 2, 3, 4
print("自动组包", a)
print(type(a))
# 自动解包
t = (1, 2, 3, 4)
a, b, c, d = t
print("自动解包", a, b, c, d)
"""---------------------- 多个变量定义 ----------------------"""
# 先自动组包，在解包
a, b, c, d = 1, 2, 3, 4
"""---------------------- 交换 ----------------------"""
a = 10
b = 20
# 一般的方式
temp = a
a = b
b = temp
# 先组包再解包的过程（右边先组包，左边解包）
a, b = b, a
print(a, b)
"""---------------------- 格式化输出 ----------------------"""
a = 10
b = 20
t = (40, 50)
# 两个是一样的，都是一个组包的过程
print("a=%d,b=%d"%(a, b))
print("a=%d,b=%d"%t)
"""---------------------- 把列表变成元组，元素不可以修改 ----------------------
"""
ll = [1, 2, 3, 4]
t = tuple(ll)
print(type(t))
print(t)
t = list(ll)
print(type(t))
print(t)
```

```
"""---------------------- 元组最后一个或第一个元素 ----------
e, *args = b
print(e)
*args, e = b
print(e)

06第一个函数的实现
1
7
```

模板

```
"""---------------------- 模块定义 ----------------------"""
def myMax(a, b):
    if a >= b:
        return a
    else:
        return b
def myMin(a, b):
    if a >= b:
        return b
    else:
        return a
```

```
import tips
a = 10
b = 20
print(tips.myMax(a, b))
print(tips.myMin(a, b))
```

导入包

调用

```
from tips import myMax
# import tips
a = 10
b = 20
print(myMax(a, b))
# print(tips.myMin(a, b))
```