课程：函数

目标

- 变量作用域
- 多函数程序执行流程
- 函数的返回值
- 函数的参数
- 拆包和交换两个变量的值
- 引用
- 可变和不可变类型

一. 变量作用域

变量作用域指的是变量生效的范围，主要分为两类：局部变量和全局变量。

- 局部变量

所谓局部变量是定义在函数体内部的变量，即只在函数体内部生效。

```
def testA():
    a = 100

    print(a)


testA()  # 100
print(a)  # 报错：name 'a' is not defined
```

> 变量a是定义在`testA`函数内部的变量，在函数外部访问则立即报错。

局部变量的作用：在函数体内部，临时保存数据，即当函数调用完成后，则销毁局部变量。

- 全局变量

所谓全局变量，指的是在函数体内、外都能生效的变量。

思考：如果有一个数据，在函数A和函数B中都要使用，该怎么办？

答：将这个数据存储在一个全局变量里面。

```
# 定义全局变量a
a = 100


def testA():
```

```
    print(a)  # 访问全局变量a，并打印变量a存储的数据


def testB():
    print(a)  # 访问全局变量a，并打印变量a存储的数据


testA()  # 100
testB()  # 100
```

思考：`testB`函数需求修改变量a的值为200，如何修改程序?

```
a = 100


def testA():
    print(a)


def testB():
    a = 200
    print(a)


testA()  # 100
testB()  # 200
print(f'全局变量a = {a}')  # 全局变量a = 100
```

思考：在`testB`函数内部的`a = 200`中的变量a是在修改全局变量`a`吗?

答：不是。观察上述代码发现，15行得到a的数据是100，仍然是定义全局变量a时候的值，而没有返回`testB`函数内部的200。综上：`testB`函数内部的`a = 200`是定义了一个局部变量。

思考：如何在函数体内部修改全局变量?

```
a = 100


def testA():
    print(a)


def testB():
    # global 关键字声明a是全局变量
    global a
    a = 200
    print(a)

```

```
14
15 testA()  # 100
16 testB()  # 200
17 print(f'全局变量a = {a}')  # 全局变量a = 200
```

二. 多函数程序执行流程

一般在实际开发过程中，一个程序往往由多个函数（后面知识中会讲解类）组成，并且多个函数共享某些数据，如下所示：

- 共用全局变量

```
1  # 1. 定义全局变量
2  glo_num = 0
3
4
5  def test1():
6      global glo_num
7      # 修改全局变量
8      glo_num = 100
9
10
11 def test2():
12     # 调用test1函数中修改后的全局变量
13     print(glo_num)
14
15
16 # 2. 调用test1函数，执行函数内部代码：声明和修改全局变量
17 test1()
18 # 3. 调用test2函数，执行函数内部代码：打印
19 test2()  # 100
```

- 返回值作为参数传递

```
1  def test1():
2      return 50
3
4
5  def test2(num):
6      print(num)
7
8
9  # 1. 保存函数test1的返回值
10 result = test1()
11
12
13 # 2.将函数返回值所在变量作为参数传递到test2函数
14 test2(result)  # 50
```

三. 函数的返回值

思考：如果一个函数如些两个return (如下所示)，程序如何执行?

```
def return_num():
    return 1
    return 2


result = return_num()
print(result)  # 1
```

答：只执行了第一个return，原因是因为return可以退出当前函数，导致return下方的代码不执行。

思考：如果一个函数要有多个返回值，该如何书写代码?

```
def return_num():
    return 1, 2


result = return_num()
print(result)  # (1, 2)
```

> 注意：
>
> 1. `return a, b`写法，返回多个数据的时候，默认是元组类型。
> 2. return后面可以连接列表、元组或字典，以返回多个值。

四. 函数的参数

4.1 位置参数

位置参数：调用函数时根据函数定义的参数位置来传递参数。

```
def user_info(name, age, gender):
    print(f'您的名字是{name}, 年龄是{age}, 性别是{gender}')


user_info('TOM', 20, '男')
```

> 注意：传递和定义参数的顺序及个数必须一致。

4.2 关键字参数

函数调用，通过“键=值”形式加以指定。可以让函数更加清晰、容易使用，同时也清除了参数的顺序需求。

```
def user_info(name, age, gender):
    print(f'您的名字是{name}, 年龄是{age}, 性别是{gender}')


user_info('Rose', age=20, gender='女')
user_info('小明', gender='男', age=16)
```

注意：**函数调用时，如果有位置参数时，位置参数必须在关键字参数的前面，但关键字参数之间不存在先后顺序。**

4.3 缺省参数

缺省参数也叫默认参数，用于定义函数，为参数提供默认值，调用函数时可不传该默认参数的值（注意：所有位置参数必须出现在默认参数前，包括函数定义和调用）。

```
def user_info(name, age, gender='男'):
    print(f'您的名字是{name}, 年龄是{age}, 性别是{gender}')


user_info('TOM', 20)
user_info('Rose', 18, '女')
```

> 注意：函数调用时，如果为缺省参数传值则修改默认参数值；否则使用这个默认值。

4.4 不定长参数

不定长参数也叫可变参数。用于不确定调用的时候会传递多少个参数(不传参也可以)的场景。此时，可用包裹(packing)位置参数，或者包裹关键字参数，来进行参数传递，会显得非常方便。

- 包裹位置传递

```
def user_info(*args):
    print(args)


# ('TOM',)
user_info('TOM')
# ('TOM', 18)
user_info('TOM', 18)
```

> 注意：传进的所有参数都会被args变量收集，它会根据传进参数的位置合并为一个元组(tuple)，args是元组类型，这就是包裹位置传递。

- 包裹关键字传递

```
def user_info(**kwargs):
    print(kwargs)


# {'name': 'TOM', 'age': 18, 'id': 110}
user_info(name='TOM', age=18, id=110)
```

> 综上：无论是包裹位置传递还是包裹关键字传递，都是一个组包的过程。

五. 拆包和交换变量值

5.1 拆包

- 拆包：元组

```
def return_num():
    return 100, 200


num1, num2 = return_num()
print(num1)  # 100
print(num2)  # 200
```

- 拆包：字典

```
dict1 = {'name': 'TOM', 'age': 18}
a, b = dict1

# 对字典进行拆包，取出来的是字典的key
print(a)  # name
print(b)  # age

print(dict1[a])  # TOM
print(dict1[b])  # 18
```

5.2 交换变量值

需求：有变量`a = 10`和`b = 20`，交换两个变量的值。

- 方法一

借助第三变量存储数据。

```
# 1. 定义中间变量
c = 0

# 2. 将a的数据存储到c
c = a

# 3. 将b的数据20赋值到a，此时a = 20
a = b

# 4. 将之前c的数据10赋值到b，此时b = 10
b = c

print(a)  # 20
print(b)  # 10
```

- 方法二

```
a, b = 1, 2
a, b = b, a
print(a)  # 2
print(b)  # 1
```

六. 引用

6.1 了解引用

在python中，值是靠引用来传递来的。

我们可以用`id()`来判断两个变量是否为同一个值的引用。 我们可以将id值理解为那块内存的地址标识。

```
# 1. int类型
a = 1
b = a

print(b)  # 1

print(id(a))  # 140708464157520
print(id(b))  # 140708464157520

a = 2
print(b)  # 1,说明int类型为不可变类型

```

```
print(id(a))  # 140708464157552, 此时得到是的数据2的内存地址
print(id(b))  # 140708464157520


# 2. 列表
aa = [10, 20]
bb = aa

print(id(aa))  # 2325297783432
print(id(bb))  # 2325297783432


aa.append(30)
print(bb)  # [10, 20, 30], 列表为可变类型

print(id(aa))  # 2325297783432
print(id(bb))  # 2325297783432
```

6.2 引用当做实参

代码如下：

```
def test1(a):
    print(a)
    print(id(a))

    a += a

    print(a)
    print(id(a))


# int：计算前后id值不同
b = 100
test1(b)

# 列表：计算前后id值相同
c = [11, 22]
test1(c)
```

效果图如下：

```
C:\Users\黑马程序员\AppData\Local\Programs\Python\Python37\python3.exe
100
140708464160688
200
140708464163888
[11, 22]
2008286519944
[11, 22, 11, 22]
2008286519944

Process finished with exit code 0
```

七. 可变和不可变类型

所谓可变类型与不可变类型是指：数据能够直接进行修改，如果能直接修改那么就是可变，否则是不可变.

- 可变类型
  - 列表
  - 字典
  - 集合
- 不可变类型
  - 整型
  - 浮点型
  - 字符串
  - 元组

八. 总结

- 变量作用域
  - 全局：函数体内外都能生效
  - 局部：当前函数体内部生效
- 函数多返回值写法

```
return 表达式1，表达式2...
```

- 函数的参数
  - 位置参数
    - 形参和实参的个数和书写顺序必须一致
  - 关键字参数
    - 写法：`key=value`
    - 特点：形参和实参的书写顺序可以不一致；关键字参数必须书写在位置参数的后面
  - 缺省参数

        - 缺省参数就是默认参数
        - 写法：`key=vlaue`
    - 不定长位置参数
        - 收集所有位置参数，返回一个元组
    - 不定长关键字参数
        - 收集所有关键字参数，返回一个字典
- 引用：Python中，数据的传递都是通过引用