基础学习 变量进阶

一、变量的引用

程序运行时，变量和数据 都是保存再内存

1、引用的概念：

在 python 中，

（1）**变量** 和 **数据**是分开存储的

（2）**数据** 保存在**内存**中的一个位置，数据中存储的就是数据

（3）**变量** 中保存着**数据在内存中的地址**，变量 中 **记录数据的地址叫做 引用**

（4）使用 **id()** 函数可以查看变量直接查询数据所在的 内存地址

变量 相当于便签值(盒子上的标签)，数据 就是数据值(标签)

python 中函数传值，是传递引用

函数返回值，返回的也是引用

```
def test(num):

 print("在函数内部 %d 对应的数据内存地址是 %d " %(num,id(num)))

 # 1> 定义一个字符串变量
 result = "hello"

 print("函数要返回数据的内存地址是 %d" % id(result))

 # 2> 将字符串变量返回,函数返回的是数据的引用，而不是数据本身
 return result

# 1、定义一个数字的变量
a = 10

print("a 变量保存数据的内存地址是 %d" % id(a))

# 2、调用 test 函数,本质上传递的是实参保存数据的引用，而不是实参保存的数据（传引用）

# 注意：如果函数有返回值，但是没有定义变量接收，程序不会报错，但是无法获得返回结果
r = test(a)

print("%s 的内存地址是 %d" % (r,id(r)))
```

运行结果如下：

```
a 变量保存数据的内存地址是 140730827789632
在函数内部 10 对应的数据内存地址是 140730827789632
函数要返回数据的内存地址是 2121534620144
hello 的内存地址是 2121534620144
```

2、可变和不可变类型

（1）变量的数据类型：五种（数字类型、字符串(str)、元组(tuple)、列表(list)、字典(dict)）

可变类型(内存中的数据不允许被修改)：数字类型、字符串、元组

不可变类型(内存中的数据可以修改)：列表、字典(**字典的 key 只能使用不可变类型的数据，对比之前在字典定义时的要求学习**)

（2）可变类型的数据：在使用 **方法** 修改时，数据的引用不发生改变，

一旦发生 **赋值语句** ，一定会改变数据本身的引用

```
>>> a = [1,2,3]
>>> id(a)
2468731141256
>>> a.append(999)
>>> a
[1, 2, 3, 999]
>>> id(a)
2468731141256
>>> a.remove(2)
>>> a
[1, 3, 999]
>>> id(a)
2468731141256
>>> a.insert(0)
Traceback (most recent call
  File "<pyshell#16>", line
    a.insert(0)
TypeError: insert() takes ex
>>> a.insert(0,0)
>>> a
[0, 1, 3, 999]
>>> a.clear()
>>> a
[]
>>> id(a)
2468731141256
>>> a = []
>>> id(a)
2468731488584
```

```
>>> d = {"name":"xiaomei"}
>>> d
{'name': 'xiaomei'}
>>> id(d)
2468731785648
>>> d["age"] = 18
>>> d
{'name': 'xiaomei', 'age': 18}
>>> id(d)
2468731785648
>>> d.pop("age")
18
>>> id(d)
2468731785648
>>> d.clear()
>>> d
{}
>>> id(d)
2468731785648
>>> d = {}
>>> id(d)
2468731786728
```

（3）哈希函数：有一个 hash(0) 的内置函数

接收一个 **不可变类型** 的数据作为参数，**返回** 结果是一个 **整数**

hash(0)：传入相同的值，得到相同的整数

传入不同的值，得到不同的整数

3、局部变量和全部变量

局部变量，定义在函数内部，只能在该函数 内部 使用

全部变量，定义在函数外部，所有函数 内部 都可以使用这个变量

（1）局部变量

生命周期：

所谓 生命周期 就是变量从 被创建 到 被系统回收 的过程

局部变量在 被执行时 才会被创建，在函数执行结束后 被系统回收

局部变量在生命周期 内，可以用来存储 函数内部临时用到的数据

（2）全局变量：

在python 中，是不允许 **直接** 修改全局变量的值，若是全局变量在函数内部使用了赋值语句，会在函数内部，定义一个局部变量

若是非要休要全局变量 —— 使用 global 声明一下变量即可

```python
# 全局变量
num = 10

def demo():

 # 修改全局变量，使用 global 声明一下变量
 global num
 num = 99
```

```
9
10   print("demo ==> %d" % num)
11
12 print("函数前的全局变量 num = %d" % num)
13
14 demo()
15
16 print("函数后的全局变量 num = %d" % num)
```

```
函数前的全局变量 num = 10
demo ==> 99
函数后的全局变量 num = 99
```

（3）全局变量的位置：

在开发时，应该把模块中的所有全局变量定义在所有函数上方，就可以保证所有的函数都能够正常的访问到每一个全局变量了

```
1  num = 10
2
3  def demo():
4
5    print(num)
6    print(title)
7    print(name)
8
9  title = "其实我这会想去看小说"
10 demo()
11 name = "王盼盼"
```

运行结果：

```
10
Traceback (most recent call last):
其实我这会想去看小说
  File "E:/Python 代码/07 语法进阶/sin 04 全局变量的位置.py", line 10, in <module>
    demo()
  File "E:/Python 代码/07 语法进阶/sin 04 全局变量的位置.py", line 7, in demo
    print(name)
NameError: name 'name' is not defined

进程已结束,退出代码1
```

代码结构示意图如下：

shebang(标识用哪个解释器解释当前的python程序)

import 模块
全局变量
函数定义
执行代码

4、函数的返回值（有多个返回值）

```python
def measure():
    """测量温度和湿度"""

    print("测量前……")
    tmp = 39
    wetness = 50
    print("测量结束……")

    # 元组 - 可以半酣多个数据，因此可以使用元组让函数一次返回多个值
    # return (tmp,wetness)
    # 如果函数返回的数据类型是元组，小括号可以省略
    return tmp,wetness

# result 元组
result = measure()
print(result)

# 若需要单独处理温度或湿度
print(result[0])
print(result[1])

# 如果函数返回值类型是元组，同时希望单独的处理元组中的元素
# 可以使用多个变量，一次接收函数的返回结果
# 注意：使用多个变量接收结果是，变量的个数应该和元组中元素的个数保持一致
# 在变量名前用 gl 一般表示全局变量
gl_tmp , gl_wetness = measure()
print(gl_tmp)
print(gl_wetness)
```

面试题 —— 交换两个数字

```python
a = 6
```

```python
b = 100

# 解法1：使用其他变量
# tmp = a
# a= b
# b = tmp

# 解法2：  不使用其他变量
 # a = a + b
 # b = a - b
 # a = a - b

 # 解法3 ：python 专享
 #a,b = (b,a)
 # 提示：等号右边是一个元组，只是把小括号省略了
 a,b = b,a

 print("a = %d" % a)
 print("b = %d" %b )
```

5、函数的参数

（1）、在函数内部，针对参数使用的 **赋值语句** ，会不会**影响调用函数时传递的 实参变量**？ **——不会**！！

```python
def demo(num,num_list):

 print("函数内部的代码")

 # 在函数内部，针对参数使用赋值语句,不会修改到外部的实参变量
 num_list = [1,2,3]
 num = 100

 print(num)
  print(num_list)
  print("函数执行完成")

gl_num = 99
gl_num_list = [4,5,6]
demo(gl_num,gl_num_list)
print(gl_num)
print(gl_num_list)
```

（2）、如果传递的参数是 **可变类型** ，在函数内部，使用 **方法** 修改了数据的内容，同样**会影响到外部的数据**

```python
def demo(num_list):

 print("函数内部的代码")

 num_list.append(9) # 注意在函数内部，参数在调用方法时，没有智能提示
 print(num_list)

 print("函数执行完成")

gl_list = [1,2,3]
demo(gl_list)
print(gl_list)
```

（3）面试题：+=

在 python 中，列表变量调用 += 本质上是在执行列表变量的 extend 方法，不会修改变量的引用

```python
def demo(num,num_list):

 print("函数开始")

 num += num
 # 列表变量使用 += 不会做相加再赋值的操作！
 # 本质上是在调用 extend 方法
 num_list += num_list
 print(num)
  print(num_list)

  print("函数完成")

gl_num = 9
gl_list = [1,2,3]
demo(gl_num,gl_list)
print(gl_num)
print(gl_list)
```

6、缺省参数

定义函数时，可以给 某个参数 指定一个 默认值，具有默认值的参数叫做 缺省参数

调用函数时，如果没有传入 缺省参数 的值，则在函数内部使用定义函数时指定的 参数默认值

```
# 在自己的函数中定义缺省参数的默认值时，在函数的形参后边写上 = (默认值)
#
def print_info(name,gender=True):
 """

 :param name: 同学姓名
 :param gender: True:男生 False:女生
 """

 gender_text = "男生"

 if not gender:
 gender_text = "女生"

 print("%s 是 %s" % (name,gender_text))

print_info("小明")
print_info("小美",False)
```

缺省参数的注意事项：

（1）必须保证 带有默认值的缺省参数 在参数列表末尾

（2）在 调用函数时，如果有 多个缺省参数，需要指定参数名，这样解释器才能指定参数的对应关系

7、多值参数

（1）使用：

```
def demo(num,*args,**kwargs):

 print(num)
 print(args) # 一个 * 表示元组，函数的形参通常表示为 *args
 print(kwargs) #一个 * 表示元组，函数的形参通常表示为 *args

demo(1,2,3,4,5,6,name="王盼盼",age=18,sex="female")
```

（2）多值累加求和

```
def sum_numbers(*args):

 num = 0
 print(args)
```

```
5
6   # 累加求和 —— 循环遍历
7   for n in args:
8   num += n
9
10   return num
11
12  result1 = sum_numbers(1,2,3,4,5,6,7)
13  result2 = sum_numbers(1,2)
14  print(result1)
15  print(result2)
```

(3) 拆包：

```
1  def demo(*args,**kwargs):
2
3   print(args)
4   print(kwargs)
5
6  args = (1,2,3,4,5,6)
7  kwargs = {"name":"王盼盼","age":18,"sex":"female"}
8
9  # 多值函数的错误传递
10  # demo(args,kwargs)
11  # 正确传递，拆包语法，简化元组变量 / 字典变量的传递
12  demo(*args,**kwargs)
```