4.函数

定义函数和return

1.定义函数

def func_name(形参):

  函数体

2.函数默认返回none，自定义返回值用return

3.函数结束两种情况：a. 遇到return直接结束　b.函数内部代码执行完

返回值实例 Python

```python
def my_len():
    s = 'hello world'
    length = 0
    for i in s:
        length = length + 1
    return length

str_len = my_len()
print(str_len)
```

使用多个值接收(值的数量必须跟返回值数量对应) Python

```python
def ret_demo():
    return 1,2,'a',['hello','world']

ret1,ret2,ret3,ret4 = ret_demo()
print(ret1,ret2,ret3,ret4)

# Output:
1 2 a ['hello', 'world']
```

函数中的参数

1.形参是占位置的

实参可以是：常量、变量、表达式或其他函数的返回值

1.位置参数

2.默认参数

3.关键字参数

4.动态参数：*args接受除关键字外的其他类型参数->组成元组、*kwargs接受关键字参数->组成字典

动态参数 Python

```python
def func(*args):
    print(args)
func(1,2,3,4,[1,2,3],(1,2,3),{'name':'nls'})

def func2(**kwargs):
    for key,value in kwargs.items():
      print(f"{key}:{value}")
func2(name='nls')
```

可以在同一个函数中同时使用 *args 和 **kwargs，但 *args 必须在 **kwargs 之前。

函数中同时使用 *args 和 **kwargs Python

```python
def display_info(*args, **kwargs):
    print("位置参数:", args)  # args 是一个元组
    print("关键字参数:", kwargs)  # kwargs 是一个字典

display_info(1, 2, 3, name="Alice", age=30)
```

*和**后面的字是可以重新命名的，但是*表示元组**表示字典是不变的

命名空间和作用域

命名空间：用于存储变量名（或标识符）与对象（值）之间的映射关系。

类型：**内置命名空间**、**全局命名空间**、**局部命名空间**

作用域：定义了变量的可访问性或可见性。

类型：局部、全局、内置

globals和locals方法

globals() 和 locals() 是两个内置函数，用于访问全局命名空间和当前所在的命名空间。

```python
x = 10
y = 20
print(globals())
print(locals())

def func():
    a = 12
    b = 20
    print(globals())  #可以打印x,y
    print(locals())   #只打印a,b

func()
```

global关键字

在函数内部声明一个变量是全局变量，从而允许你在函数内部对其进行修改。

```python
x = 10  # 全局变量
def modify_global():
    global x  # 声明 x 为全局变量
    x = 20    # 修改全局变量

modify_global()
print(x)  # 输出: 20
```

对可变数据类型（list，dict，set）可以直接引用不用通过global

```python
li = [1,2,3]
dic = {'name':'aaron'}

def change():
    li.append(4)
    dic['age'] = 18
    print(dic)
    print(li)

change()
print(dic)
print(li)
```

nolocal关键字

它允许内层函数修改外层函数的变量。仅在嵌套函数中有效，不能用于声明全局变量。

```python
def outer_function():
    x = 10  # 外层函数的局部变量

    def inner_function():
        nonlocal x  # 声明 x 为外层函数的局部变量
        x = 20      # 修改外层函数的变量

    inner_function()
    print(x)  # 输出: 20

outer_function()
```

嵌套

声明

```python
def f1():
    print('in f1')
    def f2():
        print('in f2')
    f2()
f1()
```

嵌套调用

数字比较大小 Python

```python
def max_num(x,y):
    if x>y:
        return x
    else:
        return y
def number(a,b,c,d):
    res1=max_num(a,b)
    res2=max_num(res1,c)
    res3=max_num(res2,d)
    return res3

ret = number(10,100,-2,40)
print(ret)
```

作用域链

作用域链是指在查找变量时，Python 按照一定的顺序查找变量的规则。它遵循 LEGB 规则：

Local (局部作用域)：当前函数内定义的变量。

Enclosing (包围作用域)：外层函数中的局部变量。

Global (全局作用域)：模块级别定义的变量。

Built-in (内置作用域)：Python 内置的名字，如 len()、print() 等。

函数名本质

函数名的本质可以理解为一个指向函数对象的引用，是个指针。

可以被赋值给变量 Python

```python
def greet():
    print('hello world')
a=greet
a()
```

可以被当作容器类型的元素 Python

```python
def f1():
  print('f1')
def f2():
  print('f2')
def f3():
  print('f3')
list1 =[f1,f2,f3]
list1[0]()   #list1[0]=f1,list1[0]()=f1()

db={'a':f1,'b':f2,'c':f3}
db['a']()
```

闭包

内部函数包含对外部作用域而非全剧作用域变量的引用，该内部函数称为闭包函数

举例闭包 Python

```python
def func():
    name = '张三'
    def inner():
        print(name)
    return inner

f = func()    #前面有return innner所以func()=inner,f=inner
f()           #f()=innner()
```

作用：

1.可以提前在函数的局部空间中封装一些预设的值

2.可以规定获取某个数值的方式

闭包保持计数器状态 Python

```python
def make_counter():
    count = 0  # 外部变量

    def counter():  # 嵌套函数
        nonlocal count  # 声明使用外层变量
        count += 1
        return count

    return counter  # 返回嵌套函数

# 创建一个计数器
my_counter = make_counter()

# 测试计数器
print(my_counter())  # 输出: 1
print(my_counter())  # 输出: 2
print(my_counter())  # 输出: 3
```

判断闭包函数

__closure__或者查看函数的__code__.co_freevars属性

Python

```python
def func():
    name = 'aaron'
    def inner():
        print(name)
    return inner

f = func()
print(f.__code__.co_freevars)   # 可以通过f.__code__.co_freevars属性来查看到该函数是否应用了外部作用域的变量
print(f.__closure__)    # 如果打印出的内容非空，说明该函数是闭包函数
```