笔记

> 版权：智泊AI
>
> 作者：Jeff

一、变量的作用域

1.出现的原因

> 变量的作用域：变量可以被使用【被访问】的范围
>
> 程序中的变量并不是在任意的语句中都可以被访问，访问权限取决于这个变量被定义在哪个位置
>
> 在python中 分支语句 if..else if.. 和 循环for 等 是不存在变量作用域问题的.

```
if 5 > 4:
    a = 11
print(a)

for i in range(10):
    b = 32
print(b)
# 在 分支结构中if.. if..esle  和 循环中for in 不存在作用域的问题.
# 在他们的外面可以直接访问里面定义的变量.

def fn():
    c = 99
# print(c)  # name 'c' is not defined
```

2.作用范围划分

> 局部作用域：L【Local】
>
> 函数作用域：E【Enclosing】 将变量定义在闭包外的函数中
>
> 全局作用域：G【Global】
>
> 内建作用域：B【Built-in】
>
> 代码演示：

```
def fn():
    c = 99
# print(c)  # name 'c' is not defined
# 1.函数内部定义的变量在函数外部不能访问.

# 全局作用域:
num1 = 12     # 全局作用域,在函数内部和外部可以直接访问
def test():
    num2 = 87  # 函数作用域
```

```
    print(num1)

    # print(num3)   # name 'num3' is not defined
    def inner():
        num3 = 55   # 局部作用域
        # 在内部函数中,可以访问全局作用域\函数作用域\局部作用域的变量
        print(num1,num2,num3)
    return inner

test()
print(num1)
# print(num2)

fn = test()
fn()

n = int("28")   # builins 内置函数在调用的时候的作用域,这是python解释器自己定义的.
```

总结:

1. 在 分支结构中if.. if..esle 和 循环中for in 不存在作用域的问题.在他们的外面可以直接访问里面定义的变量.
2. 作用域主要体现在函数中, 函数内部定义的变量在函数外部不能访问.
3. 函数内部可以访问函数外部的变量.

3.全局变量和局部变量【掌握】

> 全局变量：将变量定义在函数的外面
>
> 局部变量：将变量定义在函数的内部
>
> 注意：局部变量只能在其被声明的当前函数中使用，而全局变量可以在整个程序中使用

4.global和nonlocal关键字的使用【掌握】

> 使用场景：当内部作用域【局部作用域，函数作用域】想要修改全局变量的作用域的时候

1.global

> 代码演示：

```
num = 11
def test():
  num = 78
  print(num)

test()    # 78
print(num)  # 11

# 若想在函数的内部,对全局变量进行修改,需要使用global关键字
num1 = 11
def test1():
```

```
    # 通过global关键字将函数内部声明变量变为了全局变量
    global num1
    num1 = 75
    print(num1)

test1()   # 75
print(num1)  # 75
```

2. nonlocal

代码演示：

```
# nolocal 关键字主要用于闭包函数中

# nolocal关键字用于闭包函数中
x = 15  # 全局变量
def outer():
    x = 19
    def inner():
        # x = 23
        # global x   # 使用的是 x = 15
        nonlocal x  # 这时候使用的变量是 x = 19
        x += 1
        print("inner:",x)
    return inner


# 闭包会保存住当前的运行环境
test = outer()
test()   # 20
test()   # 21
test()   # 22

num = 11
def demo():
    print(num)

demo()   # 11
demo()   # 11
demo()   # 11
```

二、闭包和装饰器【了解】

如果在一个函数的内部定义另外一个函数，外部的函数叫做外函数，内部的函数叫做内函数

如果在一个外部函数中定义一个内部函数，并且外部函数的返回值是内部函数，就构成了一个闭包，则这个内部函数就被称为闭包【closure】

实现函数闭包的条件:

1.必须是函数嵌套函数

2.内部函数必须引用一个定义在闭合范围内的外部函数的变量,----内部函数引用外部变量

3.外部函数必须返回内部的函数

代码演示:

1.闭包

```
# 闭包：如果在一个外部函数中定义一个内部函数，并且外部函数的返回值是内部函数，就构成了一个闭包，则这个内部函数就被称为闭包【closure】

# 最简单的闭包
# 外部函数
def outer():
     # 内部函数
     def inner():
         print("lala")
     return inner  # 将内部函数返回

fn = outer()    # fn =====> inner函数
fn()    # 相当于调用了inner函数    输出 lala


# 内部函数使用外部函数的变量
def outer1(b):
    a = 10
    def inner1():
        # 内部函数可以使用外部函数的变量
        print(a + b)
  return inner1

fun1 = outer1(12)
fun1()

'''
注意：
1.当闭包执行完毕后,仍然能够保存住当前的运行环境
2.闭包可以根据外部作用域的局部变量得到不同的效果,类似于配置功能,类似于我们可以通过修改外部变量,闭包根据变量的改变实现不同的功能.
应用场景：装饰器
'''
```

2 装饰器：

> 在代码运行期间，可以动态增加函数功能的方式，被称为装饰器【Decorator】 通过闭包函数实现
>
> 也就是说，在不修改原函数的基础上，给原函数增加功能
>
> 好处：在团队开发中，如果两个或者两个以上的程序员会用到相同的功能，但是功能又有细微的差别，采用装饰器：相互不影响，代码简化

2.1装饰器

> 代码演示：

```
# 原函数
def test():
  print("你好啊!")
# 需求：给上面的函数test增加一个功能，输出 我很好
# 第三种方式：通过装饰器的方式给函数追加功能    装饰器使用闭包实现
'''
闭包函数：
1.函数嵌套函数
2.内部函数使用外部函数的变量
3.外部函数中返回内部函数
'''

# a.书写闭包函数     此处的outer函数就是装饰器函数
def outer(fn):  # b. fn表示形参,  实际调用的时候传递的是原函数的名字
     def inner():
         fn()  # c.调用原函数
         # d. 给原函数添加功能,    注意:添加的功能可以写在原函数的上面也可以写在原函数的下面
         print("我很好")
     return inner

print("添加装饰器之前:",test,__name__)
test = outer(test)
print("添加装饰器之后:",test,__name__) #

test()

# 总结:
# 1.在装饰器中,给原函数添加的功能,可以写在原函数的上面,也可以写在原函数的下面
# 2.outer 函数就是我们的装饰器函数
```

2.2系统的简写

> 代码演示：

```
# a.书写闭包函数     此处的outer函数就是装饰器函数
def outer(fn):  # b. fn表示形参,  实际调用的时候传递的是原函数的名字
   def inner():
```

```
        fn()  # c.调用原函数
        # d. 给原函数添加功能,    注意:添加的功能可以写在原函数的上面也可以写在原函数的下面
        print("我很好")
    return inner

# test = outer(test)

# 装饰器的简写方式 @ + 装饰器名称
@outer    # 等价于  =====>test = outer(test)
def test():
    print("你好啊!")

test()

'''
注意:
1.在使用装饰器的简写方式的时候,原函数必须在装饰器函数的下面
2.outer就是装饰器函数.  @outer等价于  test = outer(test)
'''
```

2.3不定长参数的装饰器(通用装饰器)

代码演示:

```
# 同一个装饰器装饰多个函数
def jisuan(fn):
  def inner(*args):
      print("数学运算的结果是:",end=" ")
      fn(*args)
  return inner

@jisuan
def add(a,b):
  print(a+b)

add(12,34)


@jisuan
def cha(a,b,c):
  print(a-b-c)

cha(100,23,26)
```