

# Python基础Day10

## 一、函数的动态位置参数

- 约定用\*args表示动态位置参数,但是也可以用其他变量名,不推荐
- `def func(* args):`      形参位置上的\*表示聚合  
    `print(* args)`      函数体内的\* 表示打散
- 可以接收多个参数,以元组形式接收
- 作用
  - 1.能够接收不固定长度的参数
  - 2.位置参数过多时可以使用

## 二、函数的动态关键字参数

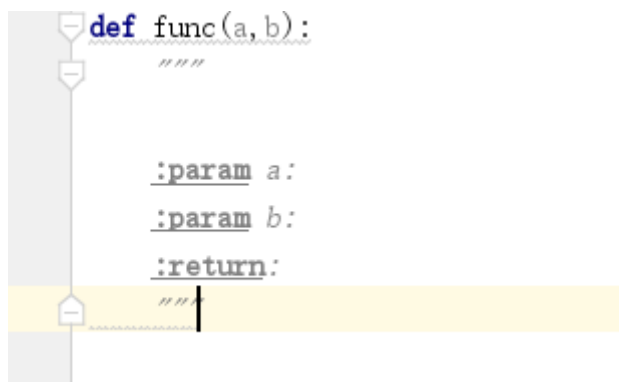
- 约定用\*\* kwargs表示动态关键字参数,与动态位置参数类似
- 动态关键字参数的打散只能\* kwargs,结果围殴字典的键
- \*\* kwargs可以接收多个关键字参数,以字典形式接收

## 三、参数位置总结

- 形参中:位置参数>动态位置参数>默认参数>动态关键字参数
- 实参中:位置参数>关键字参数

## 四、函数的注释

- 作用:解释函数的功能,使用方法,以及对参数,返回值的要求和解释
- 使用方法,在函数体最前面使用""" """注释,之后会自动生成一段注释内容



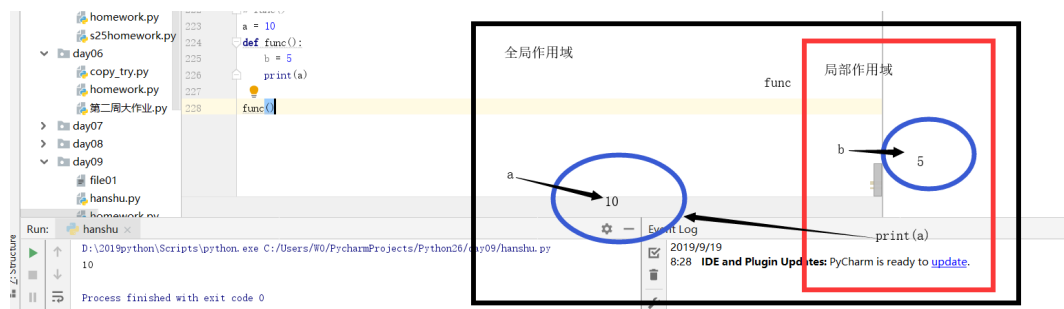
```
def func(a, b):  
    """  
  
    :param a:  
    :param b:  
    :return:  
    """
```

- 使用""" """不报错,但是不推荐
- 新式写法:
  - `def add(a:int,b:int):`  
    `print(a+b)`  
    `add(1,2)`  
  
    ◦ `a:int`只是对参数的提示,并没有实际的约束作用
- 查看注释内容: `add.__doc__`

- 查看函数名字: `add.__name__`

## 五、命名空间和作用域

- 命名空间:
  - 内置空间:存放python'自带的一些函数
  - 全局空间:py文件中所有顶格声明的变量
  - 局部空间:函数中声明的变量
  - 定义:名称到对象的映射关系,是一个字典
  - 取值顺序:局部空间-全局空间-内置空间
  - 加载顺序:内置空间-全局空间-局部空间
  - 取值时先在局部空间找,找不到后再到外面的全局空间找,最后再到内置空间



- 
- 作用域:
  - 定义:变量的作用范围
  - 全局作用域:全局空间+内置空间
  - 局部作用域:局部空间
  - 查看全局作用域:`print(globals)`
  - 查看局部作用域:`print(locals)`
  - 局部空间内可以调用全局空间和内置空间的变量,但反之则不行

## 六、函数名的第一类对象及使用

- 总结:
  - 函数名可以当作值(该函数的内存地址),并赋值给其他变量

### ■ 示例:

```

# def func():
#     print(1)

# a = func
# print(func) # 函数的内存地址
# print(a)
# a()

```

- 函数名可以当作其他函数的参数使用

### ■ 示例

```
def func():
    print("1")

def f(b):
    print(b)
    f(func)
```

- 函数名可以当作另一个函数的返回值来使用
  - 示例

```
def func():
    print("1")

def f():
    print("2")
    return func
print(f())
```

- 函数可以当作元素存储在容器中
  - 示例

```
msg = """
1. 登录
2. 注册
3. 购买
请输入序号：
"""

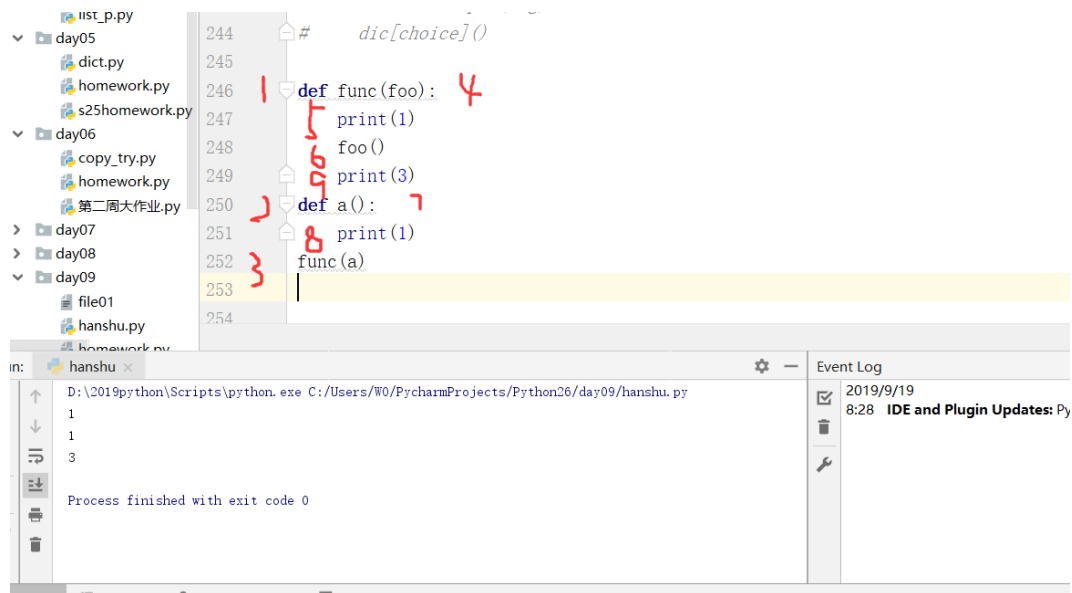
def login():
    print("登录")
def register():
    print("注册")
def buy():
    print("购买")
dic = {"1":login,"2":register,"3":buy}
while True:
    choice = input(msg)
    dic[choice]()
```

## 七、函数的嵌套

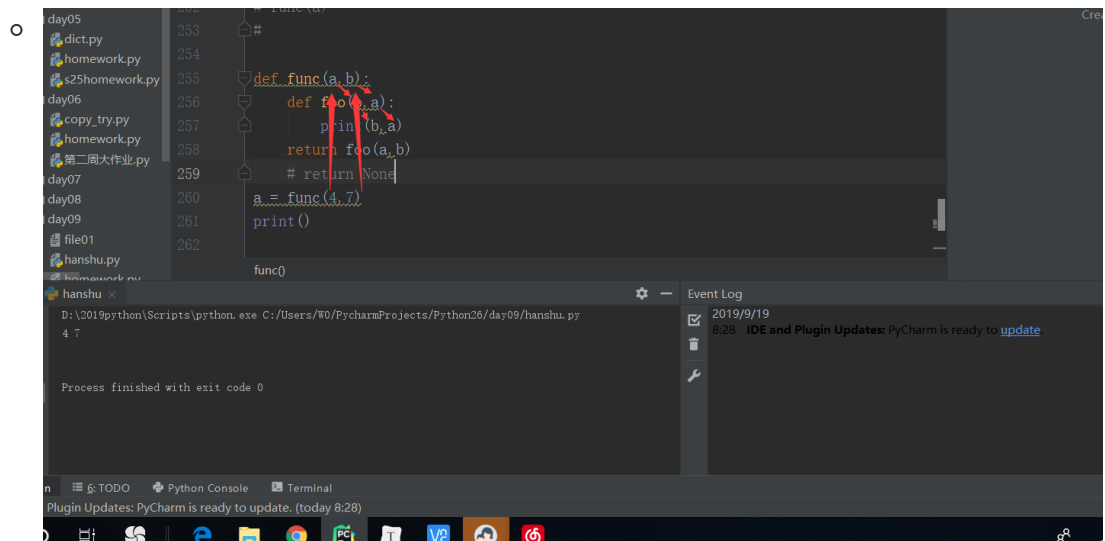
- 交叉嵌套

- ```
def func(foo):
    print(1)
    foo()
    print(3)
def a():
    print(1)
    func(a)
```

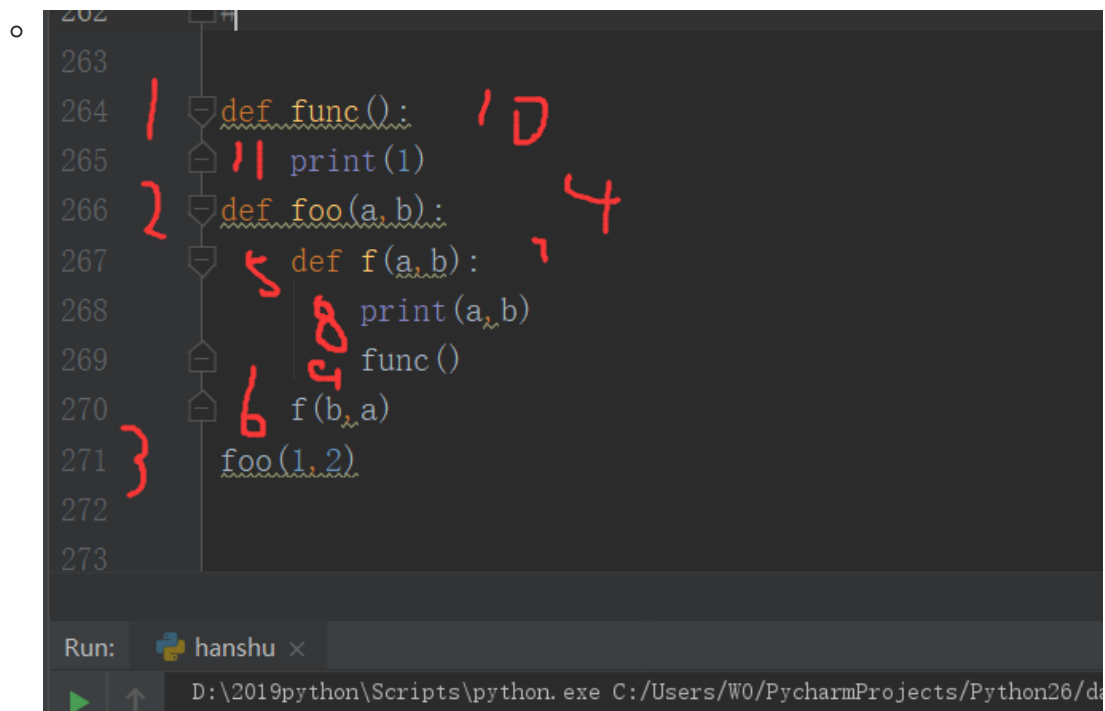
执行顺序:



### 嵌套:



不要被名字迷惑,这是位置参数,只要位置是对应的就可以,要注意返回值,若没有返回值默认返回None



注意执行顺序,还有参数的究竟是给了谁

```
def func():  
    print(1)  
def foo(a, b):  
    def f(a, b):  
        print(a, b)  
        func()  
    f(b, a)  
foo(1, 2)
```

## 八、global与nonlocal

- global:
  - 在局部空间不可以改变全局变量,用global声明就可以对其进行修改
  - 当变量存在时,global声明该变量就是要修改该变量
  - 当变量不存在时,就是要在全局范围内新建该变量
  - global只修改全局变量
  - global的使用:

```
1 # foo(1, 2)
2
3 a = 10
4 def func():
5     global a
6     a += 1
7     print(a)
8 func()
9 print(a)
```

Output: 11, 11

- nonlocals:
  - 只修改局部空间中的变量,最多到最外层函数
  - nonlocals不进行创建
- 应用:

```
o a = 15
  def func():
    def foo():
      def f():
        nonlocal a
        a = 1
        print(a)
      print(a)
      f()
    foo()
    print(a)
  func()
  print(a)
```

#由于nonlocal不能创建,所以会报错

执行顺序:

```

# 1 a = 15
# 2 def func():
# 3     def foo():
# 4         def f():
# 5             nonlocal a
# 6             a = 1
# 7             print(a)
# 8             print(a)
# 9             f()
# 10         foo()
# 11     print(a)
# 12 func()
# 13 print(a)

```

```

181 1 a=15
182 2 def func():
183     3 a=1
184     4 def foo():
185         5 a=5
186         6 def f():
187             7 nonlocal a
188             8 a=a+1
189             9 print(a)
190             10 f()
191             11 print(a)
192         12 foo()
193     13 print(a)
194 14 func()
195 15 print(a)
196
197
Run: hanshu x
D:\2019python\Scripts\python.exe C:/Users/W0/PycharmProjects/Python26/day09/han
6

```

分析:第一个打印的是f()执行的结果,f()里面没有a,到外层函数寻找,找到a=5,进行加1操作,所以打印6;

第二个打印的是foo()执行的结果,由于f()将foo()中的a修改为6,所以foo()也打印6;

第三个打印的是func()的执行结果,由于nonlocal在foo()中就已经找到了a,所以没有改变的a,func()的a依然是1;

最后是最外层的执行结果,这里找到的是全局变量a

- ```
a = 10
def func():
    a = 5
    def foo():
```

```

a = 3
def f():
    nonlocal a
    a = a + 1
    def aa():
        a = 1
        def b():
            global a
            a = a + 1
            print(a) # 11
        b()
        print(a) # 1
    aa()

    print(a) # 4
f()
print(a) #4
foo()
print(a) #5
func()
print(a)

```

#这里需要注意的地方是**b()**里面是**global**,所以改变的是全局变量