

day10

今日内容

1. 参数
2. 作用域
3. 函数嵌套

内容回顾

1. 面试题相关

- 线上操作系统：centos
- py2和py3的区别？
- 每种数据类型，列举你了解的方法。
- 3 or 9 and 8
- 字符串的反转
- is 和 == 的区别？
- $v1 = (1) / v2 = 1$
- 深浅拷贝
- 文件操作，大文件如何读取内容 [50G的日志文件]

```
v = open(...)\n\nfor line in v:\n    print(line)\n\nv.close()
```

- 一行写出：9*9乘法表【不会】
- git流程【不会】

2. 练习题

2.1 知识点回顾

1. 函数基本结果

```
def func(name,age,email):
    # 函数体(保持缩进一致)
    a = 123
    print(a)
    return 1111
    b = 456
    print(b)

result = func(1,2,3)

# 函数默认返回值: None
```

2. 参数

```
def func(n1,n2):
    print(n1,n2)

func(1,2)
func(1,[11,22,3])
func({'k1':'k'},[11,22,3])

# 严格按照顺序传参数: 位置方式传参。
# 实际参数可以是任意类型。
```

3. 返回值

- 函数没有返回值，默认返回: None
- 函数内部执行过程中遇到return，就终止。

```
def func1():
    return "完成" # 函数每次执行到此，就返回；所以下面代码永远不执行。
    for i in range(10):
        print(i)
func1()

def func2():
    for i in range(10):
        print(i)
        return "完成"
    print(666)
func2()
```

- return 可以返回任意类型

```
def func():
    return (1,2,3)

v = func()
print(v)

# 特殊:返回元组
def func():
    return 5,8,"alex"

v = func()
print(v)
```

2.2 作业题

```
def func(data_list):
    val = data_list[1::2]
    return list(val)

v1 = func([1,2,3,4,5,5])
print(v1)
v2 = func((1,2,3,4,5,5))
print(v2)

# 其他类型可以通过强制转换为列表
# v1 = (1,2,3,4)
# v1 = {1,2,3,4}
# v1 = {'k1':'v1','k2':'v2'}
v1 = "asdfasdfasdf"
v2 = list(v1)
print(v2)
```

```
def func(arg):
    if len(arg) > 5:
        return True
    else:
        return False

data = func([1111,22,3,42,12])
data = func((1111,22,3,42,12))
```

```
def func(a1,a2):
    if a1 > a2:
        return a1
    else:
        return a2
v1 = func(1,2)
v2 = func(11,2)

def func(a1,a2):
    return a1 if a1 > a2 else a2
v1 = func(1,2)
v2 = func(11,2)
```

```
def func(name,gender,age,edu):
    # template = "%s*%s*%s*%s" %(name,gender,age,edu,)
    # return template
    data_list = [name,gender,age,edu]
    return "".join(data_list)

n1 = input('>')
n2 = input('>')
n3 = input('>')
n4 = input('>')
result = func(n1,n2,n3,n4)
print(result)
```

```
def func(max_range):
    result = [1,1]
    while True:
        val = result[-1] + result[-2]
        if val > max_range:
            break
        result.append(val)
    return result

v = func(100)
print(v)
```

```
def func(name):
    with open('data.txt',mode='r',encoding='utf-8') as obj:
        # 方式一
        content = obj.read()
        flag = False
        row_list = content.split('\n') #
        ['1|alex|123123','2|eric|rwerwe','3|wupei qi|pppp']
        for row in row_list:
            v= row.split('|')
            if v[1] == name:
```

```

        flag = True
        break

    return flag

func('alex')
func('eric')

# #####
def func(name):
    with open('data.txt',mode='r',encoding='utf-8') as obj:
        # 方式一
        content = obj.read()
        row_list = content.split('\n') #
        ['1|alex|123123','2|eric|rwerwe','3|wupeiqi|pppp']
        for row in row_list:
            v= row.split('|')
            if v[1] == name:
                return True

v1 = func('alex')
if v1:
    print('存在')
else:
    print('不存在')

```

内容详细

1. 参数

1. 基本参数知识

- 任意个数
- 任意类型

```

def func(a1,a2,a3):
    print(a1,a2,a3)

func(1,"asdf",True)

```

2. 位置传参（调用函数并传入参数）【执行】

```

def func(a1,a2):
    print(a1,a2)

func(1,3)

```

3. 关键字传参【执行】

```
def func(a1, a2):  
    print(a1, a2)  
  
func(a2=99, a1=2)  
  
# 关键字传参数和位置传参可以混合使用 (位置传入的参数 > 关键字参数在后 = 总参数个数)  
def func1(a1, a2, a3):  
    print(a1, a2, a3)  
  
# func(1, 2, a3=9)  
# func(1, a2=2, a3=9)  
# func(a1=1, a2=2, a3=9)  
# func(a1=1, 2, 3) # 错误
```

4. 默认参数【定义】

```
def func(a1, a2, a3=9, a4=10):  
    print(a1, a2, a3, a4)  
  
func(11, 22)  
func(11, 22, 10)  
func(11, 22, 10, 100)  
func(11, 22, 10, a4=100)  
func(11, 22, a3=10, a4=100)  
func(11, a2=22, a3=10, a4=100)  
func(a1=11, a2=22, a3=10, a4=100)
```

5. 万能参数 (打散)

- *args
 - 可以接受任意个数的位置参数，并将参数转换成元组。
 - 调用函数无 *

```
def func(*args):  
    print(args)  
  
func(1, 2, 3, 4)
```

- 调用函数有 *

```
def func(*args):  
    print(args)  
  
func(*(1, 2, 3, 4))  
func(*[1, 2, 3, 4])
```

- 只能用位置传参

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

# func(1)
# func(1,2)
func(1,2) # args=(1, 2)
func((11,22,33,44,55)) # args=((11,22,33,44,55),)
func(*(11,22,33,44,55)) # args=(11,22,33,44,55)
```

o **kwargs

- 可以接受任意个数的关键字参数，并将参数转换成字典。
- 调用函数无 **

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

func(k1=1, k2="alex")
```

- 调用函数有**

```
def func(**kwargs):
    print(kwargs)
func(**{'k1': 'v2', 'k2': 'v2'}) # kwargs={'k1': 'v2', 'k2': 'v2'}
```

- 只能用关键字传参
- 综合应用：无敌 + 无敌 => 真无敌

```
def func(*args, **kwargs):
    print(args, kwargs)

# func(1,2,3,4,5,k1=2,k5=9,k19=999)
func(*[1,2,3], k1=2, k5=9, k19=999)
func(*[1,2,3], **{'k1':1, 'k2':3})
func(111,222, *[1,2,3], k11='alex', **{'k1':1, 'k2':3})
```

参数相关重点：

1. 定义函数

```
def func1(a1,a2):
    pass

def func2(a1,a2=None):
    pass

def func3(*args,**kwargs):
    pass
```

2. 调用函数 位置参数 > 关键字参数

2. 作用域

python中:

- py文件: 全局作用域
- 函数: 局部作用域

```
a = 1
def s1():
    x1 = 666
    print(x1)
    print(a)
    print(b)

b = 2
print(a)
s1()
a = 88888
def s2():
    print(a,b)
    s1()

s2()
```

- 总结:
 - 一个函数是一个作用域

```
def func():
    x = 9
    print(x)
func()
print(x)
```

- 作用域中查找数据规则: 优先在自己的作用域找数据, 自己没有就去 "父级" -> "父级" -> 直到全局, 全部没有就报错。注意: 父级作用域中的值到底是什么?


```
x = 10
def func():
    x = 9
    print(x)

func()
```

o 练习题:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

# x = 10
# def func():
#     x = 9
#     print(x)
#     def x1():
#         x = 999
#         print(x)
#
# func()

# x = 10
# def func():
#     x = 9
#     print(x)
#     def x1():
#         x = 999
#         print(x)
#     x1()
#
# func()

# x = 10
# def func():
#     x = 9
#     print(x)
#     def x1():
#         x = 999
#         print(x)
#     print(x)
#     x1()
#
# func()

# x = 10
# def func():
#     x = 8
#     print(x)
```

```
#     def x1():
#         x = 999
#         print(x)
#     x1()
#     print(x)
#
# func()
```

```
# x = 10
# def func():
#     x = 8
#     print(x)
#     def x1():
#         print(x)
#     x1()
#     print(x)
#
# func()
```

```
# x = 10
# def func():
#     x = 8
#     print(x)
#     def x1():
#         print(x)
#     x = 9
#     x1()
#     x = 10
#     print(x)
#
# func()
```

```
#
# x = 10
# def func():
#     x = 8
#     print(x)
#     def x1():
#         print(x)
#
#     x1()
#     x = 9
#     x1()
#     x = 10
#     print(x)
#
# func()
```

- 子作用域中只能 找到父级中的值，默认无法重新为父级的变量进行赋值。(global/nonlocal可以强制做)

```

# #####
name = 'oldboy'
def func():
    name = 'alex' # 在自己作用域再创建一个这样的值。
    print(name)
func()
print(name)

# #####
name = [1,2,43]
def func():
    name.append(999)
    print(name)
func()
print(name)

# ##### 如果非要对全局的变量进行赋值
# 示例一
name = ["老男孩", 'alex']
def func():
    global name
    name = '我'
func()
print(name)
# 示例一
name = "老男孩"
def func():
    name = 'alex'
    def inner():
        global name
        name = 999
    inner()
    print(name)
func()
print(name)

name = "老男孩"
def func():
    name = 'alex'
    def inner():
        global name
        name = 999
    inner()
    print(name)
func()
print(name)

# ##### nonlocal
name = "老男孩"
def func():

```

```
name = 'alex'
def inner():
    nonlocal name # 找到上一级的name
    name = 999
inner()
print(name)
func()
print(name)
```

总结

- 参数
 - 调用（执行）函数时，传参：位置参数 > 关键字参数
 - 定义函数：
 - def func(a)
 - def func(a,b=None) # 对于默认值，如果是可变类型，----> 坑。
 - def func(*args,**kwargs)
- 作用域
 - 函数为作用域
 - 自己 > 父级 > 父级 > 全局 【读/修改（可变）】
 - 重新赋值：
 - global
 - nonlocal
- 例题

补充

1. 全部变量以后必须全部是大写

```
USER_LIST = [11,22,3]

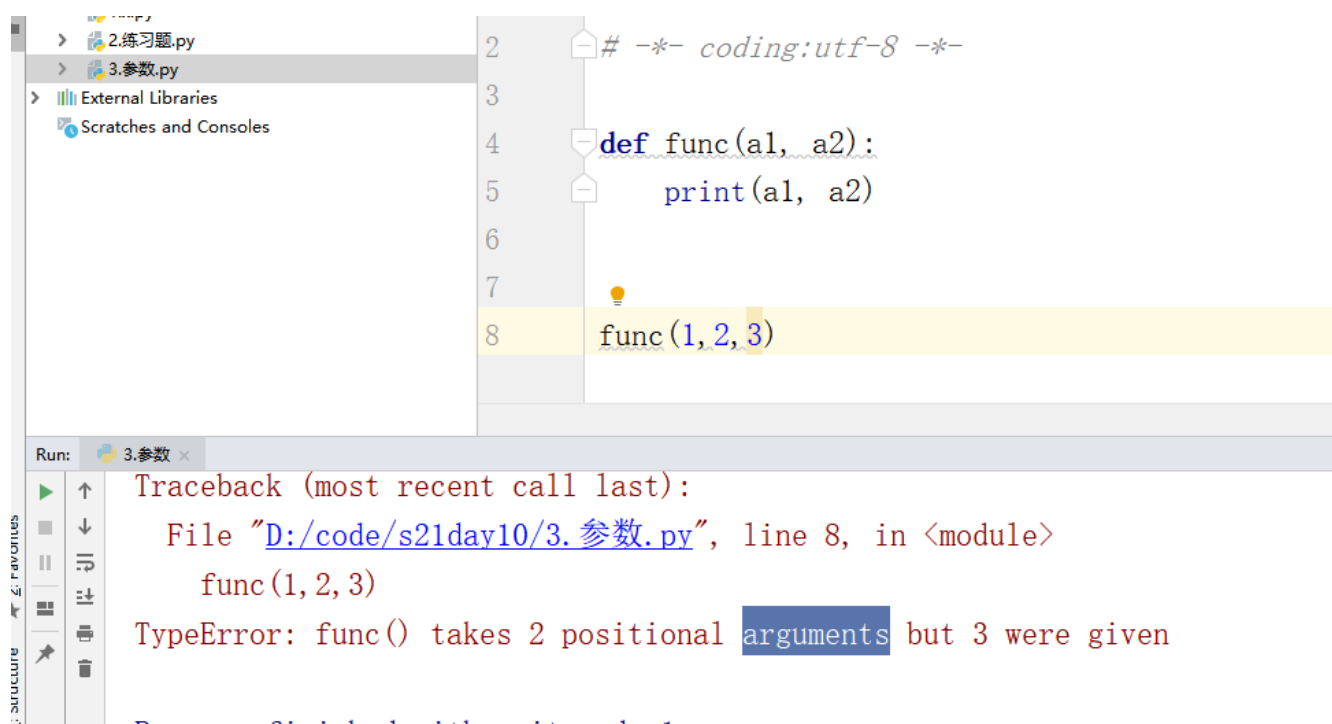
def func():
    name = 'asdf'
    USER_LIST.append(12)
    USER_LIST.append(name)

func()
print(USER_LIST)
```

强调

- 一类：
 - 笔记
 - 作业
 - 面试真题（预习）
- 二类：
 - 看老师笔记 + 自己笔记 + 记忆 ==> 笔记
 - 做题 + 2难题 => 笔记
- 三类：
 - 看视频 + 练习题 ==> 笔记
 - 作业（放弃2道题）
 - 时间：4

错误



```
2 # -*- coding:utf-8 -*-
3
4 def func(a1, a2):
5     print(a1, a2)
6
7
8 func(1, 2, 3)
```

Run: 3.参数 ×

Traceback (most recent call last):

File "D:/code/s21day10/3. 参数.py", line 8, in <module>

func(1, 2, 3)

TypeError: func() takes 2 positional arguments but 3 were given