Week 03

1. 文件操作：

读操作：

1. 格式：f = open("文件路径"，mode = "r",encoding = "utf-8")

   f : 代表文件句柄

2. 文件路径：

   1. 绝对路径：从根文件夹下查找
   2. 相对路径： 相对于某个文件进行查找

```
f = open("D:\Python_s25\day08\小姐姐电话号",mode="r",encoding="utf-8")#
print(f.read())   # 全部读取
(1)print(f.read(3))  # 按照字符读取
(2)print(f.readline())  # 默认尾部有一个\n# (3)print(f.readline().strip())
# 读取一行# (4)print(f.readline().strip())  # 将\n去除#
(5)print(f.readlines()) #一行一行读取,全部存储在列表中
```

3. 读字节：rb

   1. 读取图像，视频时使用的的操作
   2. 字节操作，不能执行encoding

```
f = open("timg.jpg",mode="rb")# print(f.read())    # 全部读取
(1) print(f.read(3))     # 按照字节读取
(2)(f.readline())      # 按照行进行读取
(3)print(f.readlines())
```

4. r 和 rb 的区别：

   1. r 需要指定encoding，rb 不需要
   2. r模式中的read（数字）代表按照字符读取

      rb模式中的read（数字）代表按照字节读取

   3. read 和 readlines 如果文件较大时，会出现内存溢出

```
面试题  ****
当文件较大时，会出现内存溢出现象
解决办法：
使用for循环进行读取
```

写操作

1. 只写，写字符：w

   1. 特性：写操作--清空写

      - 先清空文件（打开文件时就清空）
      - 在写入内容

2. 当文件模式为w 和 a 模式时，有文件就使用当前文件，没有问价那就创建一个新的文件
3. 写入的内容必须是字符串

```
ff = open("a1",mode="w",encoding="utf-8")# ff.write("[1,2,3,4]\n")  # 写的内容必须是字符串
ff.write('1111\n')  # 写的内容必须是字符串 ff.write('2222\n')  # 写的内容必须是字符串
```

2. 只写，写字节：wb
    1. 清空写，写的是字节

```
ff = open("a1",mode="w",encoding="utf-8")
ff.write("[1,2,3,4]\n")  # 写的内容必须是字符串# ff.write('1111\n')  # 写的内容必须是字符串# ff.write('2222\n')  # 写的内容必须是字符串
```

3. 追加操作
    1. 追加写，写文本（a）

```
f = open("b1",mode="a",encoding="utf-8")
f.write("你好啊\n")
f.write("我好啊\n")
f.write("他好啊\n")
f.write("大家好啊\n")
```

    2. 追加写，写字节（ab）
4. 其他操作：
    1. 读写：r+

```
# 坑 -- 使用方式是错误
# f = open("b1",mode="r+",encoding="utf-8")
# f.write("今天是周一")
# print(f.read())


# 正确的操作:
# f = open("b1",mode="r+",encoding="utf-8")
# print(f.read())
# f.write("今天是周一")
```

    2. 写读：w+

```
# w+ 写读 (有点用)
# f = open("b1",mode="w+",encoding="utf-8")
# f.write("今天是周一")
# f.seek(0)  # 移动光标
# print(f.read())
```

    3. 追加读：a +

```
# a+ 追加读  # 坑
f = open("b1",mode="a+",encoding="utf-8")
f.write("今天是周一")
f.seek(0)  # 移动光标
f.write("啊啊啊啊")
print(f.read())
```

4. 其他操作：

```
# seek() 移动光标
# f.seek(0,0)  # 移动光标到文件的头部
# f.seek(0,1)  # 移动光标到当前位置
# f.seek(0,2)  # 移动光标到文件末尾
# f.seek(6)   # 光标是按照字节移动
```

```
# 查看光标：
# tell 查光标
# f = open("c1","r",encoding="gbk")
# print(f.read(3))
# print(f.tell())  # 按照字节进行计算
```

5. 修改文件：

   - import os 操作系统交互的接口

```
# f = open('a2',"r",encoding="utf-8")
# f1 = open("a1","w",encoding="utf-8")
# for i in f:
#     i = i.replace("日","天")
#     f1.write(i)
#
# f.close()
# f1.close()
# os.remove("a2")   # 删除不能找回
# os.rename("a1","a2")
```

```
 考点：
# import os  # 操作系统交互的接口
# f = open('a2',"r",encoding="utf-8")
# f1 = open("a1","w",encoding="utf-8")
# i = f1.read().replace("天","日")    # 将文件中全部内容读取 容易导致内存溢出
# f1.write(i)
#
# f.close()
# f1.close()
# os.rename("a2","a3")
# os.rename("a1","a2")
```

6. with open：

   1. 自动关闭文件
   2. 同一时间操作多个文件

```
# with open("a3","r",encoding="utf-8")as f,\
#         open('a2',"r",encoding="utf-8")as f1:
#     print(f.read())
#     print(f1.read())
```

文件操作的目的

持久化，永久储存

2. 函数初识：

定义：

1. 将某个功能封装到一个空间中就是一个函数
2. 减少重复代码

```
lst = [1,2,3,4,5]   #列表，元祖，字典，字符串都可以求长度

n = 0

for i in list:
    n += 1
print(list)
```

定义函数：def——python关键字

```
def ():
    函数体
    # def-python 关键字，len-函数名--变量名一样，（）必须要写的，格式规定，：代表语句结束
```

函数的调用：

- 函数名+（）
    1. 在调用函数
    2. 接收返回值

```
def yue():
    print("掏出手机")
    print("打开微信")
    print("聊天")
    print("约会")
yue()
```

```
#面向函数编程：
def yue():
    print("掏出手机")
    print("打开微信")
    print("聊天")
    print("约会")
yue()
print("上班")
yue()
print("吃饭")
```

函数的返回值（return）：

1. return 值 ==返回值
2. 可以返回任意类型数据
3. return返回多个内容是元祖的形式
4. return下方不执行，并且会终止当前函数
5. return不写或者写了return后面不写值都会返回None
6. 函数的返回值返回给函数的调用者
7. 函数的返回值可以有多个结果，

```
def yue():
    print("掏出手机")
    print("打开微信")
    print("聊天")
    print("约会")
    print("......")
    return"女朋友" #函数中遇到return，此函数结束，不再继续执行
girl = yue()
print(girl)
```

函数的参数：

1. 参数分类：
    1. 位置参数：一一对应
    2. 默认参数：参数定义时括号中写好的就是默认参数
        - 不进行传参使用默认参数，使用传参时使用传递的参数
    3. 关键字参数（默认参数）：按照名字进行传参
    4. 位置参数必须放在默认参数之前
    5. 混合·参数：未知参数和关键字参数一起传参

```
def yue(a,app1="微信"):
    print("掏出手机")
    print(f"打开{a}{app1}")
    print("聊天")
    print("约会")
yue("探探")
```

2. 形参：函数定义阶段括号中的参数叫做形参
3. 实参：函数调用阶段括号中的参数叫做实参
4. 传参：将实参传递给形参的过程叫传参

```
#例：
def yue(app):        #形参
    print("掏出手机")
    print("打开"+app)
    print("聊天")
    print("约会")
yue("app名字")#    实参   输出结果是yue（）里面的内容，一个萝卜一个坑，括号里面的内容必须和前面def yue（）数量一一对应
```

传参：将实参传递给形参的过程叫传参

三元运算（三目运算）：

1. 条件成立的结果 条件   条件不成立的结果（只可以使用if   else）

```
def func(a,b):
    return a if a > b else b
print(func(6,9))
```

函数是一种编程思维

3.动态位置参数：

```
def eat(*args): #函数的定义阶段     *聚合（打包）
    print(args) #元祖
    print(*args)#函数体中的*，打散（解包）
```

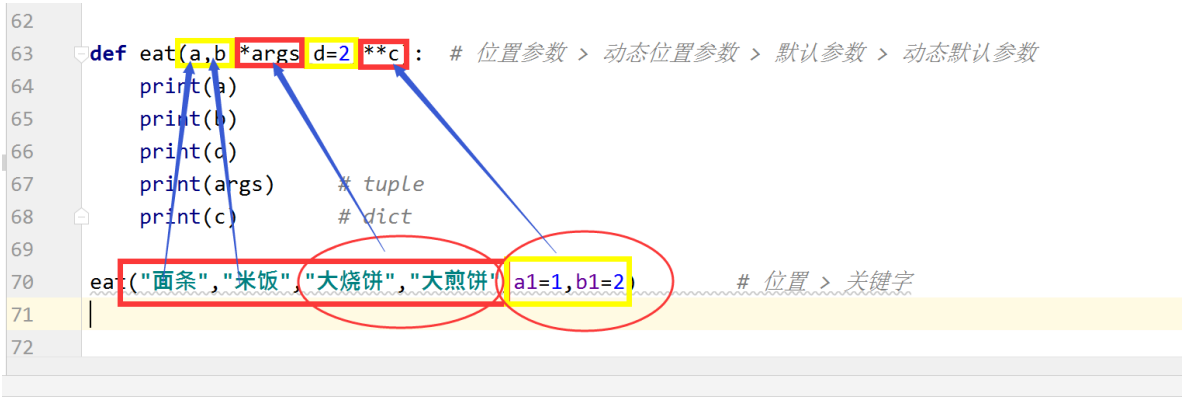
```
def eat(a,b,*c):     #a，b为位置参数>带*号的为动态位置参数
    print(a)
    print(b)
    print(c)     #元祖、
eat("面条", "米饭","馒头","大饼")
#输出结果：面条，米饭，('馒头', '大饼')
```

确定思想：位置参数永远大于默认参数，动态参数也是一样

```
# def eat(a,b,*args,d=2,**c):  # 位置参数 > 动态位置参数 > 默认参数 > **c是动态默认参数
#     print(a)
#     print(b)
#     print(d)
#     print(args)      # tuple
#     print(c)         # dict
# eat("面条","米饭","大烧饼","大煎饼",a1=1,b1=2)
```

输出结果：

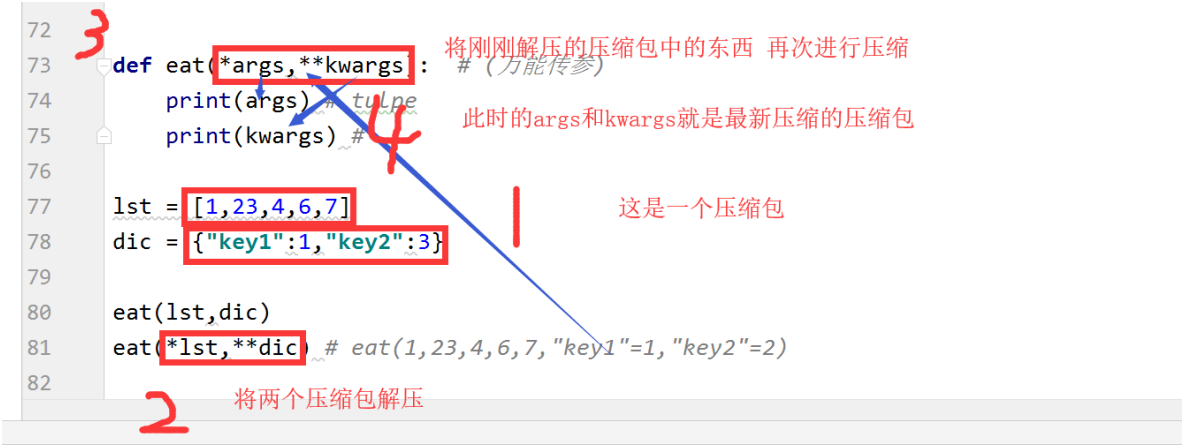
```
# 输出结果：
# 面条
# 米饭
# 2
# ('大烧饼', '大煎饼')
# {'a1': 1, 'b1': 2}
```




```
thon.exe "D:/Python_s25/day10/04 函数的动态参数.py"
```

```
# def eat(*args,**kwargs):  # (万能传参)
#     print(args) # tulpe
#     print(kwargs) #dic

# lst = [1,23,4,6,7]
# dic = {"key1":1,"key2":3}
# eat(*lst,**dic)
输出结果:
(1, 23, 4, 6, 7)
{'key1': 1, 'key2': 3}
```




```
ython.exe "D:/Python_s25/day10/04 函数的动态参数.py"
, 7], {'key1': 1, 'key2': 3})

 7)
```

*args：大家伙都用的名字，可以进行修改但是不建议

**kwargs(聚合关键字参数)大家伙都用的名字，可以进行修改但是不建议

定义时的优先级：

位置参数 > 动态位置参数 > 默认参数 > 动态默认参数

函数体中的 * ：

- 第一个代表：聚合
- 第二个代表：打散

```
def eat(a,b,*args):
    print(a)    #面条
    print(b)    #米饭
    print(*c)   #元祖，(馒头，大饼)
eat("面条","米饭","馒头","大饼")
```

形参：

- 位置参数：定义在函数体开头的时候
- 动态位置参数：先执行位置参数，位置参数接收后额外的参数动态位置参数进行接收，获取到的是一个元祖
- 默认参数：函数接收体接收到的函数
- 动态关键字参数（默认）：先执行默认参数，默认参数接收后，额外的默认参数动态进行接收，获取到的是一个字典

实参和函数体：

- *打散
- **实参能够使用

动态关键字参数：

在函数的定义阶段*和**都是聚合

函数体中的 * 就是打散，* args将元祖中的元素进行打散，**kwargs将字典的键获取

```
def eat(a,b,*args,**kwargs):
    print(a)    #面条
    print(b)    #米饭
    print(*args)    #元祖，(馒头，大饼)
    print(**kwargs) #字典{'a1':1,'a2':4}
eat("面条","米饭","馒头","大饼",a1=1,a2 = 4)
```

形参：

位置参数：

动态位置参数：先执行位置参数，未知参数接收完后额外的参数动态位置参数进行接收，获取到的是一个元祖

默认参数

动态关键字参数：先执行默认参数，默认参数接收后额外的默认参数动态默认参数进行接收，获取的是一个字典

实参和函数体：

*打散

**实参时能够使用

4. 函数的注释

查看函数名注释：*.doc*

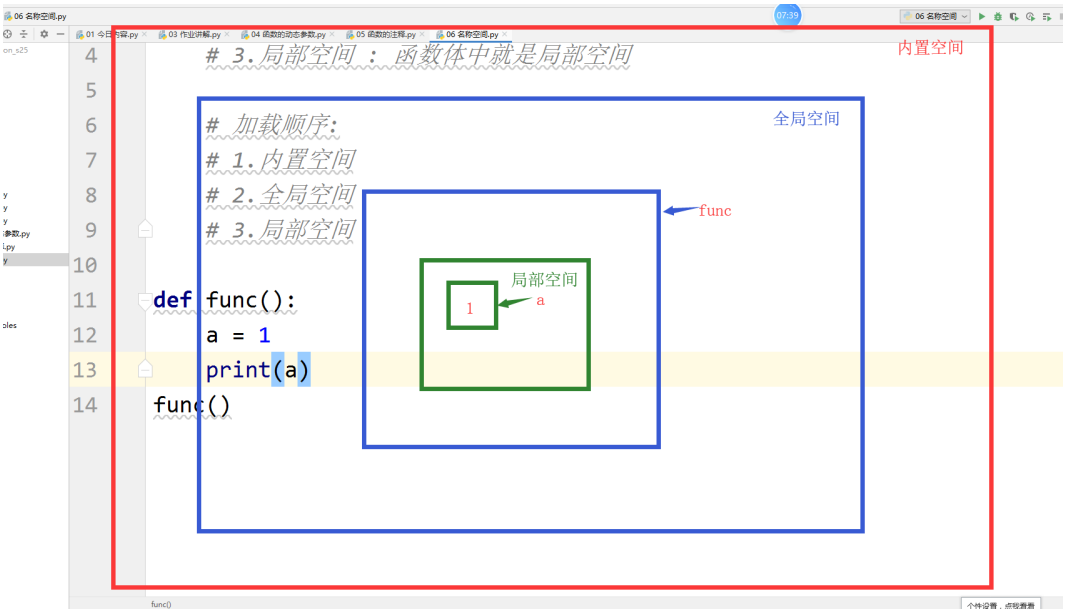
```
# def a(a:int,b:int):
#     """
#     “声明注释内容，例如此函数的意思就是进行加运算”
#     :param a: int
#     :param b: int
#     :return:  int
#     """
#     return a + b
```

查看函数的名字：*.name*

5. 名称空间：

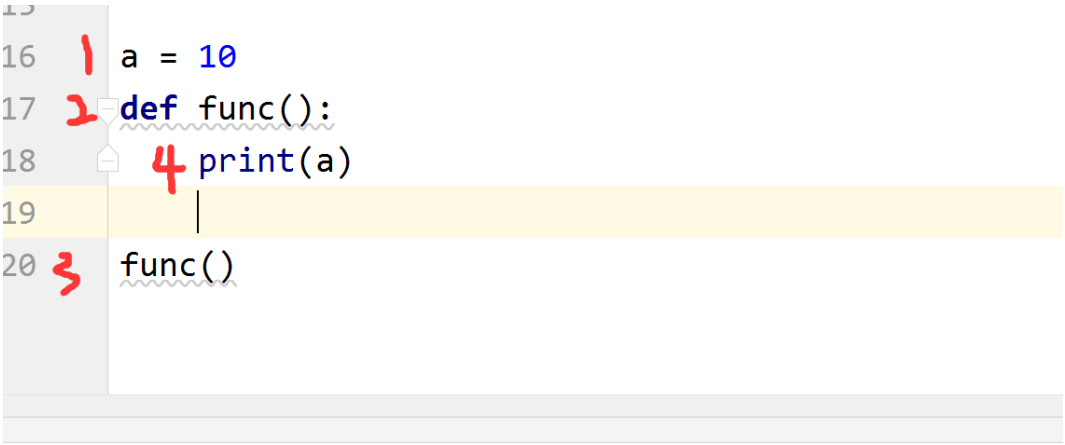
1. 内置空间：python解释器自带的一块空间
2. 全局空间：py文件中顶格写的就是全局空间
3. 局部空间：函数体中就是局部空间
4. 加载顺序：
    1. 内置空间
    2. 全局空间
    3. 局部空间

```
# def func():
#     a = 1
#     print(a)
# func()
```



5. 取值顺序：
    1. 局部空间
    2. 全局空间
    3. 内置空间

```
a = 10
def func()
    print(a)
func()
```




```
hon.exe "D:/Python_s25/day10/06 名称空间.py"
```

6. 作用域：
    1. 全局作用域：全局+内置
    2. 局部作用域：局部

6.函数的嵌套

不管在什么位置，只要是函数名（）就是在调用一个函数。

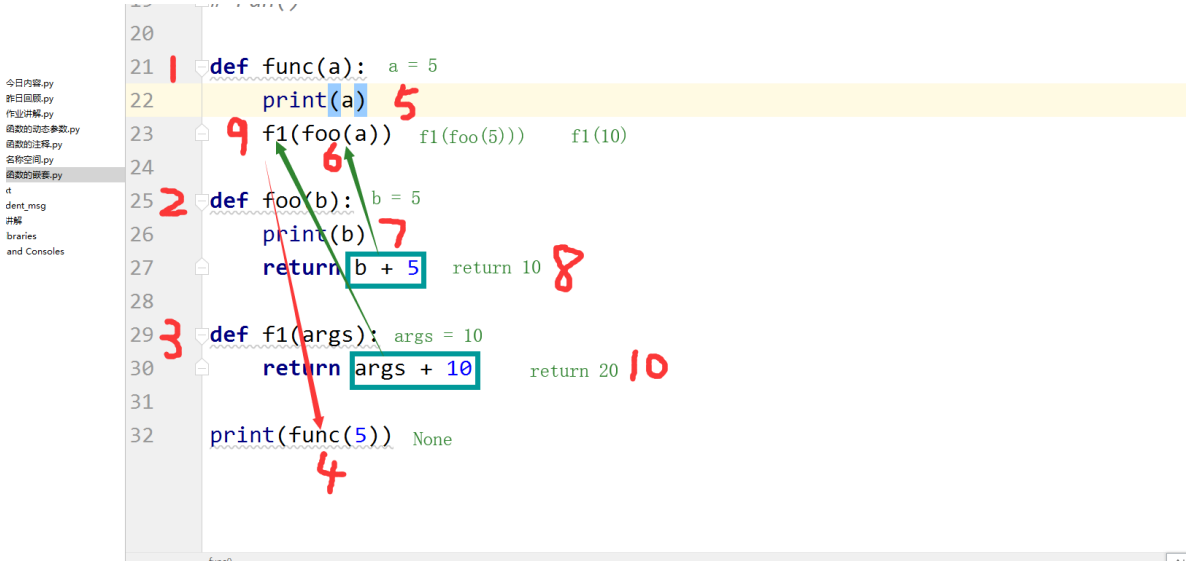
```
# 混合嵌套：
# def f1():
#     print(11)
#
# def f2():
#     print(22)
#     f1()
#
# def f3():
#     print(33)
#     f1()
#
# def run():
#     f3()
#     f2()
#     f1()
# run()
```

```
30
31  def func():
32      print("这是一个生成器")
33      yield "生成器"
34
35  g = func()
36  print(next(g))
```

例2：

```
# def func(a):
#     print(a)
#     return f1(foo(a))
#
# def foo(b):
#     print(b)
#     return b + 5
#
# def f1(args):
#     return args + 10
#
# print(func(5))
```

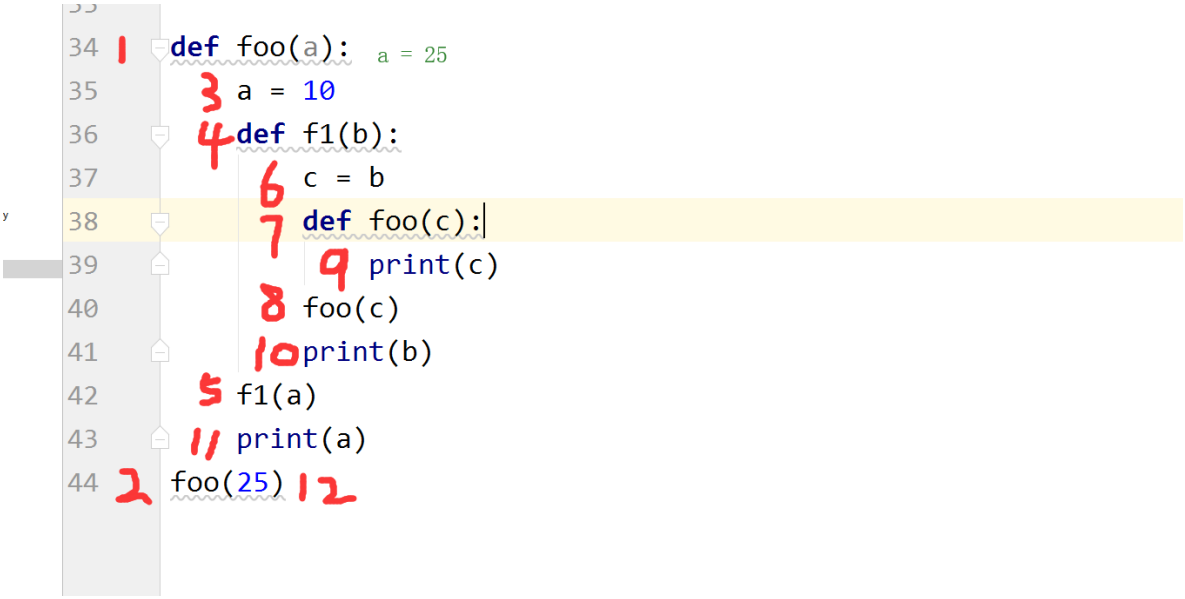
输出结果：5，5，20




```
# def foo(a):
#     a = 10
#     def f1(b):
#         c = b
#         def foo(c):
#             print(c)
#             print(foo.__doc__)
#         foo(c)
#         print(b)
#     f1(a)
```

```
#     print(a)
# foo(25)
```

输出结果：10，10，10



7.修改全局：

global，只修改全局

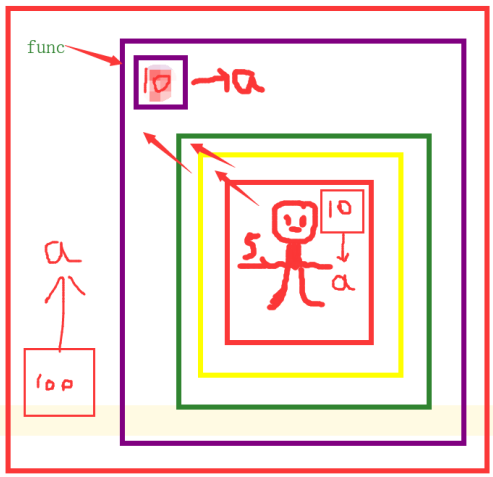
nonlocal：修改局部，修改离他最近的一层，上一层没有继续向上层查找，只限局部

```
# a = 10
# def func():
#     global a
#     a = a - 6
#     print(a)
# fun ()
```

```
# a = 100
# def func():
#     b = 10
#     def foo():
#         b = a
#         def f1():
#             nonlocal b
#             b = b + 5
#             print(b)  # 105
#         f1()
#         print(b)  # 105
#     foo()
#     print(b) # 10
# func()
# print(a) # 100
```

```
# print(a)

a = 100
def func():
    a = 5
    def foo():
        def f1():
            nonlocal a
            a = a + 5
            print(a)
        f1()
        print(a)
    foo()
    print(a)
func()
print(a)
```



8.函数名的使用以及第一类对象：

函数名的第一类对象（概述）：

使用方式：

- 函数名可以当做值赋值给变量

```
def func():
    print(1)
print (func)     #查看函数的内存地址
a = func
print (a)        #
```

- 函数名可以当做容器中的元素

```
dic = {"1":login,"2":register,"3":index}
msg = """
1 登录
2 注册
3 主页
"""
choose = input(msg)   # 1
if choose.isdecimal():
     if dic.get(choose):
         dic[choose]()
     else:
         print("请正确输入!")
```

- 函数名可以当做函数的参数

```
def fuc(a):
    a()
    print(111)
def foo():
    print(222)
    def f1()
        print(333)
    fun(f1)
foo()
```

- 函数名可以当函数的返回值

```
def func():
    def foo():
        print(111)
    return foo
a = func()
a()
func()()        #foo()
输出结果：111
        111
```

```
def func():
    def foo():
        print(111)
    return foo

a = func()
a()

func()()  # foo()
```

进阶题：

```
def foo(a):
    def func(a):
        def f1(a):
            print(a)
            return "aelx"
        return f1(a)
    return func(a)
print(foo(5))
```

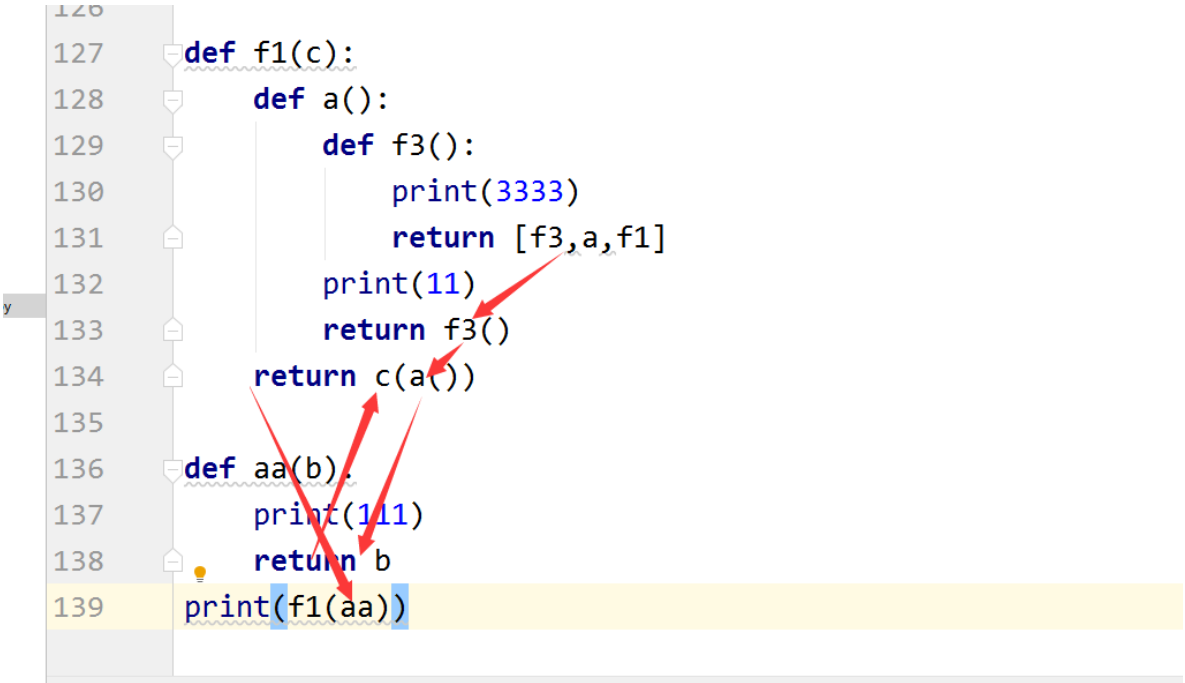
输出结果： 5 ， alex

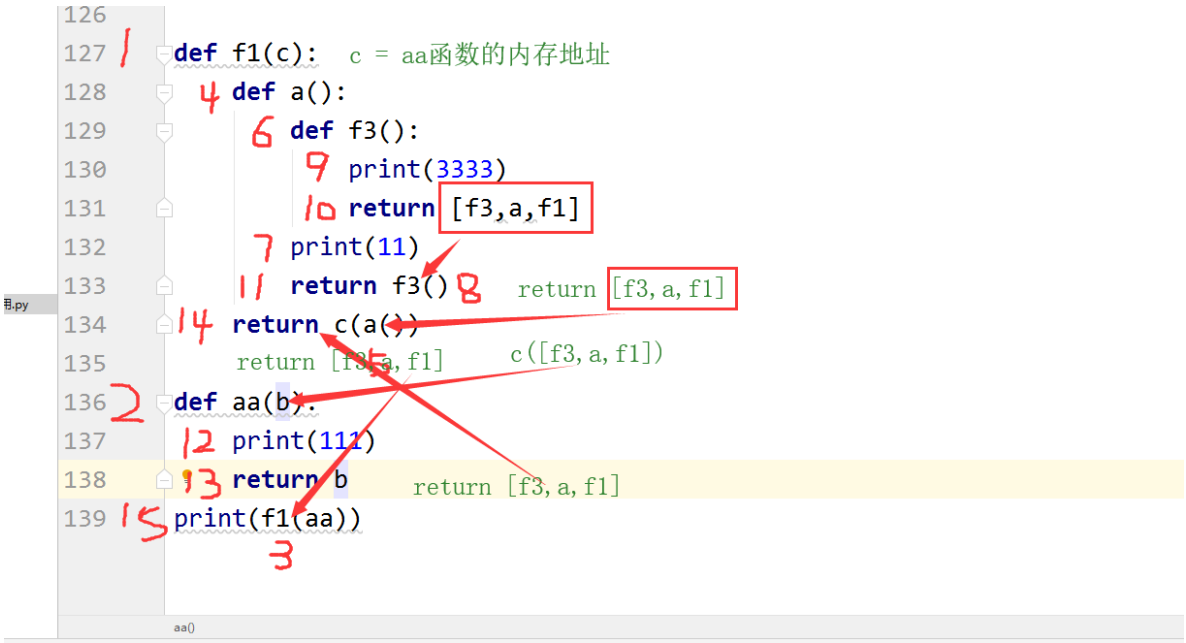
```
def func(a):
    a()

def foo(b):
    return b()
def f1(c):
    def a():
        def f3():
            print(3333)
            return [f3,a,f1]
        print(11)
        return f3()
    return c(a())
def aa(b):
    print(111)
    return b
print(f1(aa))
```

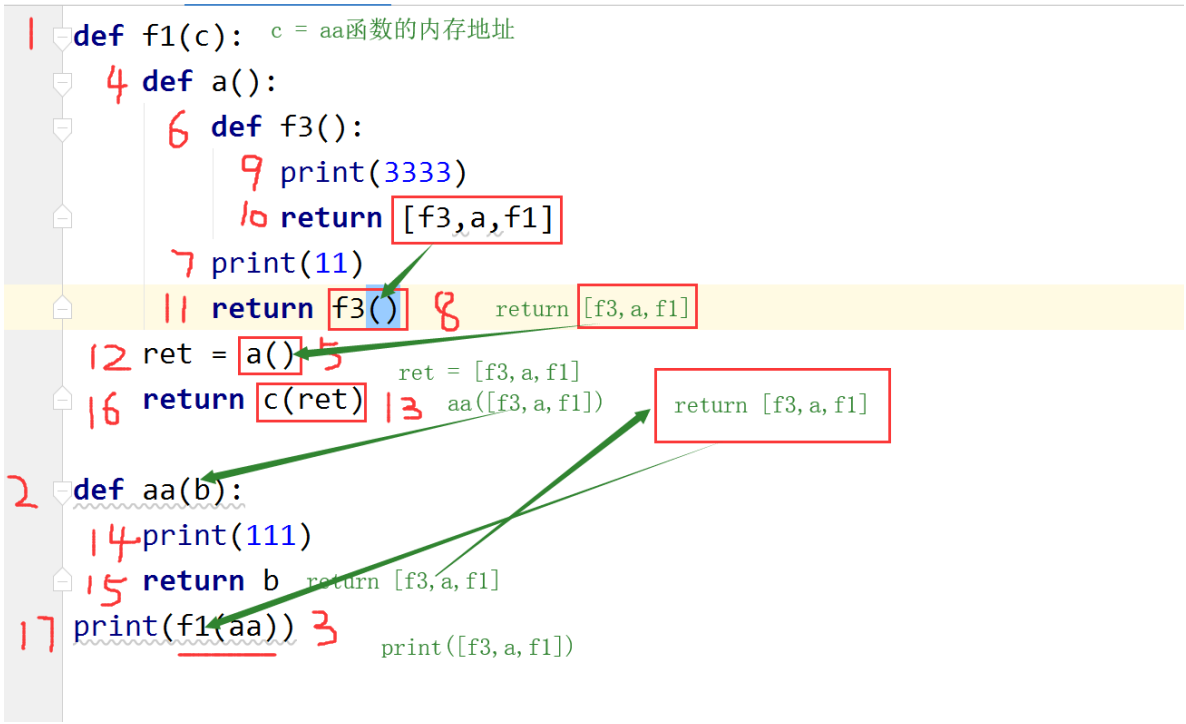
输出结果：

```
11
3333
111
[<function f1.<locals>.a.<locals>.f3 at 0x00000187D0649C80>, <function f1.
<locals>.a at 0x00000187D0649BF8>, <function f1 at 0x00000187D0649AE8>]
```






```
def f1(c):
    def a():
        def f3():
            print(3333)
            return [f3,a,f1]
        print(11)
        return f3()
    ret = a()
    return c(ret)
def aa(b):
    print(111)
    return b
print(f1(aa))
```



9.f-strings

f"{变量名}"

F"{变量名}"

f"""{变量名}"""

```
print(F"姓名:{input('name:')} 年龄:{input('age:')}")
```

```
def foo():
    print("is foo")
```

```
lst = [1,2,3,4]
dic = {"key1":23,"key2":56}
print(f"""{dic['key1']}""")
```

```
print(f"{3+5}")
print(f"{3 if 3>2 else 2}")
```

```
print(f"""{':'}""")

msg = f"""{{{{'alex'}}}}"""     #必须是偶数
print(msg)

name = "alex"
print(f"{name.upper()}")

print(f"{':'}")
```

10.迭代器：是基于上一次停留的位置，进行取值

1. 可迭代对象：
    1. list，tuple，str，set，dict取值方式只能直接看、
    2. 只要具有__ iter __()方法就是一个可迭代对象

```
lst = [1,23,4,5]
for i in lst:
    print(i)
```

```
lst = [1,2,3,4]
lst.__iter__()
dict.__iter__()
```

2. 迭代器：工具

- 具有__ iter __ () 和 __ next __ () 两个方法才是迭代器
- ```
  lst = [1,2,3,4,5]
  l = lst.__iter__()  # 将可迭代对象转换成迭代器

  l.__iter__()  # 迭代器指定__iter__()还是原来的迭代器
  print(l.__next__())   # 1
  print(l.__next__())   # 2
  ```

3. for循环的本质（重点）：

```
while True:
    try:        #异常处理机制
        print(l.__next__())
    except StopIteration:  #接收到异常提醒就终止
        break
```

```
lst = []
for i in range(10000):
    lst.append(i)

l = lst.__iter__()

for i in range(16):
    print(l.__next__())

print(''.center(50,"*"))

for i in range(16):
    print(l.__next__())

print(''.center(50,"*"))
for i in range(16):
    print(l.__next__())
```

4. 优点：
   - 惰性机制：节省空间
5. 缺点：
   1. 不能直接查看值 -- 迭代器查看到的是一个迭代器的内存地址
   2. 一次性，用完就没了
   3. 不能逆行，后退，只能继续执行下一条
6. 空间换时间：容器存储大量的数据，取值快，占用空间大
7. 时间换空间：迭代器就是一个节省了空间，但是取值慢
8. 可迭代对象：具有 iter() 方法的就是一个可迭代对象
9. 迭代器：具有 iter（）和next（）方法就是一个迭代器
10. python2和python3中的区别：
    1. python 3
```

- iter （）和 __ iter __() 都有
- next （） 和 __ next __ () 都有

2. Python 2

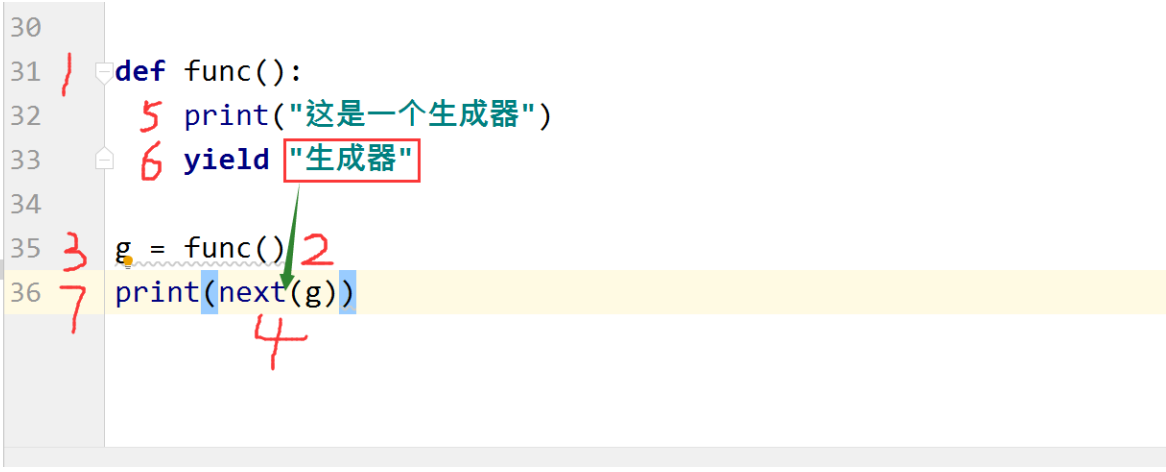
- iter （）和 __ iter __()
- next （）

11.什么是生成器

1. 核心：生成器的本质就是一个迭代器
    1. 迭代器是python自带的的
    2. 生成器是程序员自己写的一种迭代器
2. 编写方式：
    1. 基于函数编写
    2. 推导式编写

```
def func ():
    print("这是一个函数")
    return“函数”
func()
```

```
def func():
    print("这是一个生成器")
    yield"生成器"
#func()      生成一个生成器
print(func().__next__)  #启动生成器
输出结果：获取到的是一个生成器的
```



3. 内存地址函数体中出现yield代表要声明一个生成器，generator -- 生成器
   获取到的是一个生成器的内存地址

```
# 获取到的是一个生成器的内存地址
# <generator object func at 0x00000087C2A10CA8>
```

```
# def func():
#     msg = input("请输入内容")
#     yield msg
#     print("这是第二次启动")
#     yield "生成器2"
#     yield "生成器3"
```

```
#     yield "生成器4"
#
# g = func()
# print(next(g))
# print(next(g))
# print(next(g))
# print(next(g))

# 执行结果：
# 请输入内容>>>>你好
# 你好
# 这是第二次启动
# 生成器2
# 生成器3
# 生成器4
```

一个yield 必须对应一个next

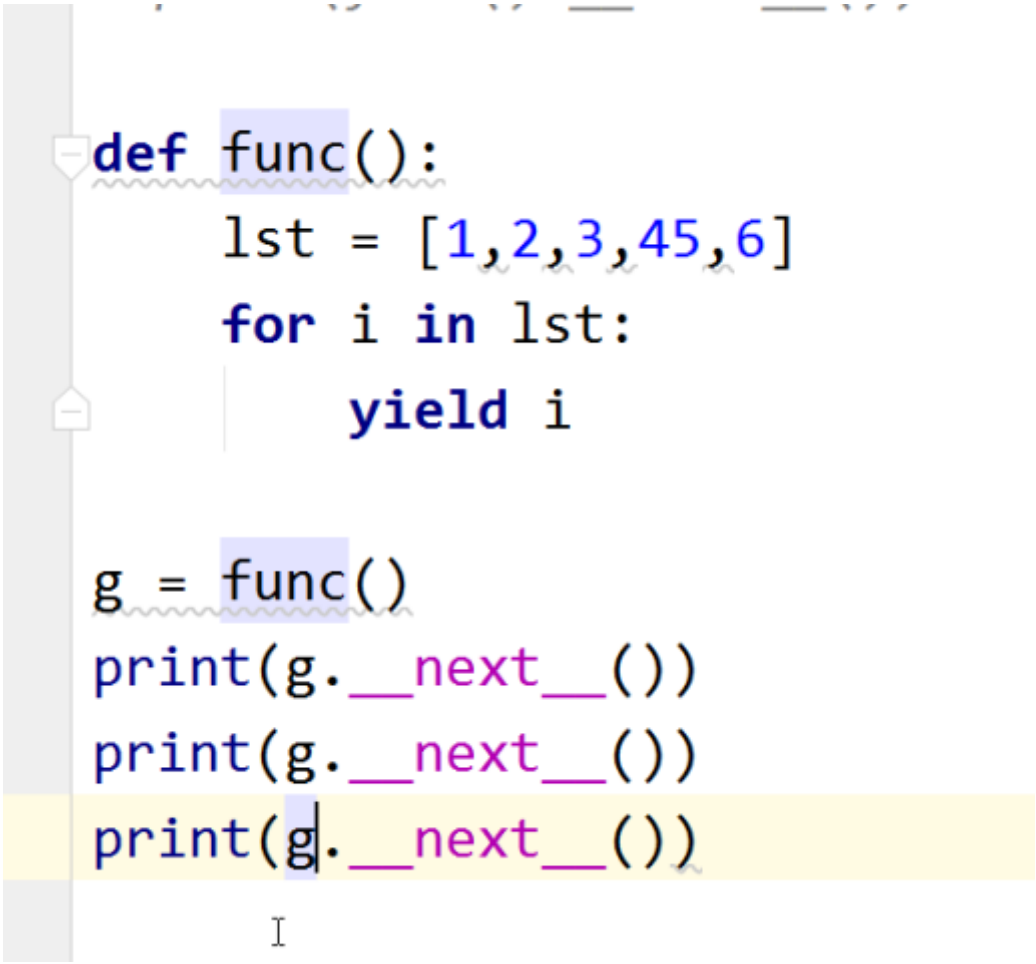
```
# def func():
#     lst =[]
#     for i in range(100000):
#         lst.append(i)
#     return lst
# print(func())
# 输出结果：[0~100000]
```

4. 用途：节省空间，例题：吃包子问题
5. 使用场景：
    1. 当文件或容器中数据量较大时，建议使用生成器
6. 可迭代对象：（python 3 ）
    - 优点：list，tuple，str 节省时间，取值方便，使用灵活（具有自己的私有办法）
    - 缺点：消耗内存
7. 迭代器：
    - 优点：节省空间
    - 缺点：不能直接查看值，使用不灵活，消耗时间，一次性，不可逆
8. 生成器：
    - 优点：节省空间，人为定义
    - 缺点：不能直接查看值，消耗时间，一次性，不可逆行
9. yield 和 return 区别：
    1. 相同点
        - 都是返回内容
        - 都可以返回多次，但是return写多个只会执行一个
    2. 不同点
        - return会终止函数，yield是暂停生成器
        - yield能够记录当前执行位置
10. 区别：通过send区别什么是迭代器，什么是生成器
    - 迭代器的地址<list_iterator>
    - 生成器的地址

```
# 数据类型 (pyhton3: range() | python2 :xrange()) 都是可迭代对象 __iter__()
# 文件句柄是迭代器  __iter__() __next__()
```

没有send方法就是一个迭代器，具有send方法就是一个生成器

```
# def func():
#     lst = [1,2,3,45,6]
#     lst1 = ["alex","wusir","taibi","baoyuan"]
#     yield from lst
#     yield from lst1
# g = func()
# print(g)
# 输出结果：<generator object func at 0x0000010DA038CF68>
```



```python
def func():
    lst = [1,2,3,45,6]
    for i in lst:
        yield i

g = func()

for i in g:
    print(i)
```

10. yield from将可迭代对象逐个返回

    yield 将可迭代对象一次性返回

12. 什么是推导式

1. 列表推导式：
    1. 循环模式：（[变量 for循环]）
        - print([i for i in range(10)])
    2. 筛选模式：[加工后的变量 for循环 加工条件]
        - print （[i for i in range（10） if i > 2])
        - print （[i for i in range（10） if i % 2 == 0])
2. 集合推导式：
    1. 普通循环：{变量 for循环}
        - print （ {i for i in range（10）} ）
    2. 筛选模式：{加工后的变量for循环 加工条件}
        - print({i for i in range (10) if i % 2 == 1})
3. 字典推导式：
    1. 普通：{键：值 for循环}
        - print {"key":"value" for i in range (10)}
    2. 筛选：{加工后的键：值 for循环 加工条件}
        - print({i : i+1 for i in range(10) if i % 2 == 0})
4. 生成器推导式（面试必问）：
    1. 普通模式：（变量 for 循环）
        - tu = （i for i in range（10）） 这是生成器，切记
    2. 筛选模式：（加工后的变量 for 循环 加工条件）

```
tu = （i for i in range（10）if  i > 5）
    for i in tu :
        print(i)
```

13. 内置函数一：

1. eval：执行字符串类型的代码
2. exac：执行字符串社类型的代码

   eval与exac 禁止使用
3. hash（）作用就是区分可变数据类型与不可变数据类型

```
# print(hash("123"))
# print(hash(12))
# print(hash(-1))
# print(hash(-10))
# print(hash((2,1)))

# dic = {[1,2,3]:2}
# print(hash([1,2,3]))
```

4. help（）：查看帮助信息
5. callable（）：查看对象是否可以调用，

```
# def func():
#     print(1)
# lst = [1,23,4,]
# print(callable(lst))   # 查看对象是否可调用
```

6. int（）：将字符串或数字转换成整型
7. float（）：转换成浮点数
8. complex（）：复数
9. bin（）：十进制转二进制
10. oct（）：十进制转八进制
11. hex（）：十进制转十六进制
12. divmod（）：计算除数与被除数结果，包含一个商和余数的元祖
13. round（）：保留浮点数的小数位数，可以设定保留位数，默认保留整数
14. pow（）：求x ** y次幂（三个参数的时候为x ** y的结果对第三个参数取余）
15. bytes（）：用于不同编码之间的转换，建议使用encode
16. ord（）：通过元素获取当前表位编码位置
17. chr（）：通过表位序号查找对应的元素
18. repr（）：查看数据的原生态（给程序员使用的）
19. all（）：判断容器汇总的元素是否都为真，返回true
20. any（）：判断容器中的元素有一个为真，就是True