day 08 函数高级

一函数的参数

区别：赋值和修改

1、传递不可变类型参数，不会影响参数本身

2、传递可变类型参数，如果直接重新赋值，那么也不会影响参数本身，如果直接在原对象基础上进行修改会影响。

案例

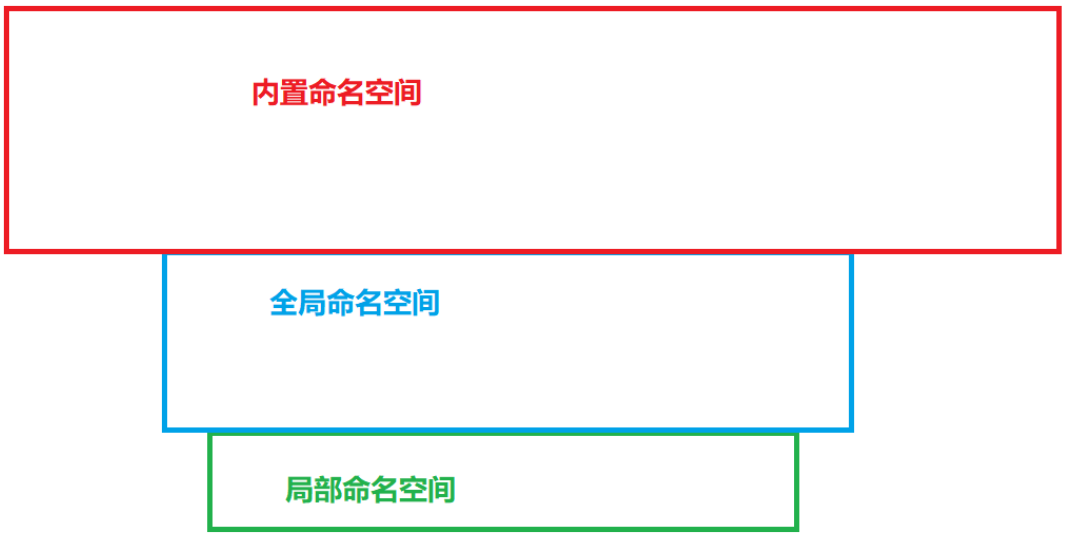
```
def f(n=[]):
    n.append(3)

    print(f"第二次n的值是{n}")

# 第一次调用
f()   # [3]

# 第二次调用使用默认参数:第一次调用会影响第二次
# f()  #  [3,3]

# 第二次调用不使用默认参数：第一次调用不会影响第二次
f([10])  # [10,3]
```

二、命名空间

1 概念：保存对象和值的字典

2 分类

- 局部命名空间：函数内定义内容。
- 全局命名空间：模块级别，当前所有的py文件
- 内置命名空间：解释器级别，内置的方法等



3 访问命名空间

- locals():访问局部命名空间：根据调用的位置有关系
- globals():访问全局命名空间：与调用位置无关

4 加载顺序：从大到小

5 查找顺序：从小到大

```
# 情形1
id = 10

def func():
    id = 20
    print(id)  # 20

func()  #



# 情形2
id = 10

def func():
    # id = 20
    print(id)  # 10

func()


# 情形3
# id = 10

def func():
    # id = 20
    print(id)  # id函数

func()
```

三、 作用域，局部变量和全局变量

1 概念

作用域：一个对象起作用的范围。

局部变量：仅在函数内部起作用的变量。

全局变量：在整个文件中起作用的变量。

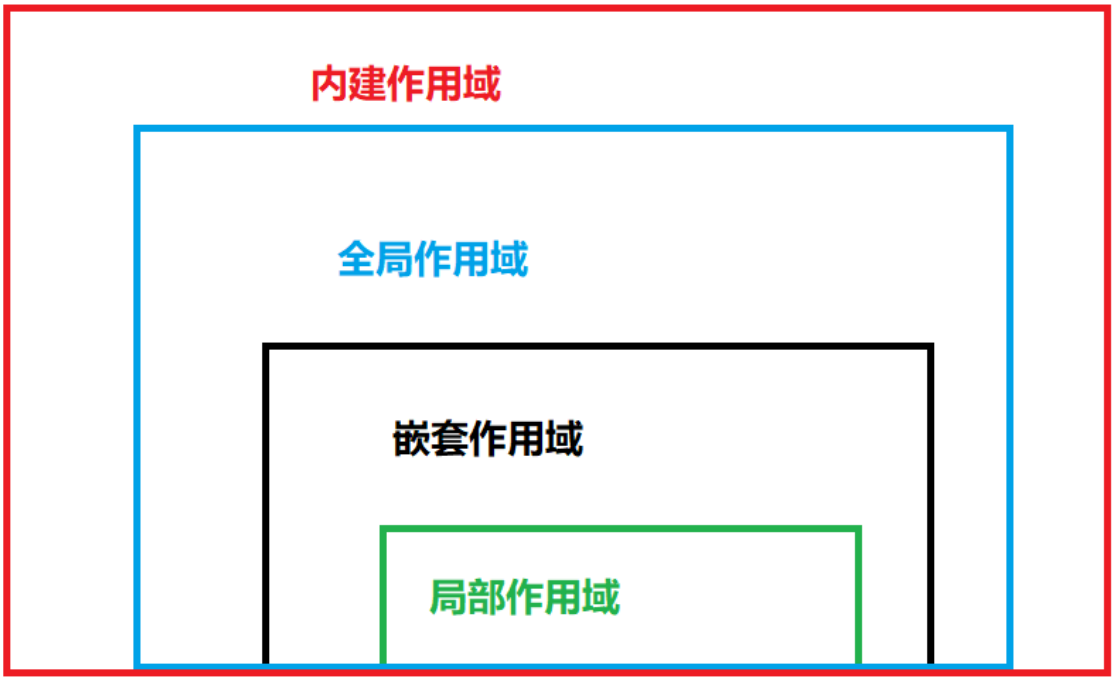
2 作用域分类（LEGB）

- Local(函数内部)局部作用域
- Enclosing（嵌套函数的外层函数内部）嵌套作用域（闭包）
- Global（模块全局）全局作用域
- Built-in（内建）内建作用域

```
a = 10
def func():
    b = 20
    def inner():
        c = 30
        print(c)
    inner()
    print(b)

print(a)
```

注意：

在Python中，模块（module），类（class）、函数（def、lambda）会产生新的作用域。



四 global和nonlocal

注意：仅仅是修改操作时候需要声明。

- 将局部变量声明为全局变量：`global 变量名`

```
# 情形1
n = 20
def func():
    n = 30
    print(n)  # 30

func()


# 情形2
n = 20
def func():
    # n = 30
    print(n)  # 20
```

```
func()

# 情形3
n = 20
def func():
    # n = 30
    print(n+1)  # 21

func()

# 情形4
# n = 20
# def func():
#     # n = 30
#     n = n+1  # n 一定是局部变量
#     print(n)  # 报错
#
# func()
# print(n)  #


# 情形5
n = 20
def func():
    global n  # 将声明为全局变量
    n = n+1
    print(n)  # 21

func()
print(n)  # 21
```

- 在函数嵌套中，将内层局部变量声明为外层局部变量

```
# 情形1
def func():
    age = 20
    def inner():
        age = 30
        print(age)  # 30
    inner()
    print(age)   # 20

func()


# 情形2
def func():
    age = 20
    def inner():
        # age = 30
        print(age)  # 20
    inner()
    print(age)   # 20

func()
```

```
# 情形3
def func():
    age = 20
    def inner():
        # global age  # 函数外无age变量，声明为全局变量会错误
        nonlocal age  # 将内层变量age声明为外层age
        age = age + 1
        print(age)  # 21
    inner()
    print(age)   # 21

func()
```

五 内置函数

1 abs():取绝对值

```
print(abs(-10))  # 10
print(abs(10))  # 10
```

2 max,min()取最大最小值

```
# 参数形式1：max(ars1,args2...)
#  全部数字
# print(max(1,23,5,34,8,987,9,3,3,32,))  # 987

# 字母
# print(max('v','w','wrq','hh','sw'))  # wrq

# 参数形式2：max(iterable)
# print(max([12,4,5,7,2,6,97]))  # 97
# print(max([12,4,5,7,2,6,97],23,5,6,12,999))  # 错误


### 指定规则查找最大值，key参数必须为函数
# 按照绝对值大小
# print(max([1,4,99,-100,43],key=abs))  # -100


#
lst = [
    {"name":'wer','price':100},
    {"name":'qq','price':70},
    {"name":'yy','price':99},
]

# 按照name字母顺序
def func(d):
    return d['name']

# 按照价钱排序
def func2(d):
    return d['price']

print(max(lst,key=func))
print(max(lst,key=func2))
```

3 map(func,iterable)

```
# 将lst转化为[1,4,9]
lst = [1,2,3]

# 方法1
# l2= []
# for i in lst:
#     l2.append(i*i)

# 法2：列表推导式
# lst2 = [i**2 for i in lst]

# 法3：map
# def f(x):
#     return x*x
#
# res = map(f,lst)
# res2 = map(f,lst)
# for i in res2:
#     print(i)  # 1  4  9
#
# lst2 = list(res)  # [1,4,9]
# print(lst2)
```

4 filter(func,iterable)

```
# 练习:过滤出列表中的所有奇数
l = list(range(1,11))
# 推导式
l2 = [i for i in l if i%2 == 1]


# filter
def f2(x):
    if x % 2 == 1:
        return True
    else:
        return False
```

5 zip(iter1,iter2,....)

```
# for i in zip([1,2,3,4],[4,5,6,],[7,8,9]):   # (1,4,7)
#     print(i)

# print(list(zip([1,2,3],[4,5,6,],[7,8,9])))


# 练习：现有两个元组('a','b'),('c','d')，请生成[{'a':'c'},{'b':'d'}]格式.

# 法1
# l3 = []
# for i in zip(('a','b'),('c','d')):
#     l3.append({i[0]:i[1]})
```

```
# print(l3)

# 法2
# l3 = [{i[0]:i[1]} for i in zip(('a','b'),('c','d'))]
# print(l3)
```

六 匿名函数

格式：
函数名 = lambda [参数1,参数2...]:表达式

lambda表达式总结：
优点：代码简洁，不增加额外变量
缺点:难于理解，降低了可读性
建议：不提倡使用lambda， 除非你知道自己在干什么。在团队开发中， 一个良好易读的代码是非常重要的，有助于提升团队协同开发效率，减少沟通和维护成本

```
"""
# 示例1
func = lambda :3<2
# print(type(func))
print(func())
# 练习2：传递多个参数
func2 = lambda a,b,c,d:a+b+c+d
print(func2(1, 3, 6, 8))

# 定义函数给定两个数返回较小值
func3 = lambda n1,n2:n1 if n1<n2 else n2
print(func3(10, 100))
```

七 嵌套作用域

理解课件中的习题

```
def f():
    x = 4
    action = lambda n,y=x:y**n
    # def action(n,y=x):
    #     return y**n
    return action

a = f()
# print(a)
b = a(3)  # action(n,y=x)
print(b)  # 4**3 = 64

d = a(3,3)
print(d)  # 3**3=27
```

```
# 练习2：请说出li[0](2)的值，并且说明为什么

def f():
    li = []
    for i in range(5):
        li.append(lambda x:i**x)
```

```
        # def f2(x):
        #     return i**x
        # li.append(f2)
    return li


li = f()


print(li[0](3))  # i**3  4**3 = 64
print(li[1](3))  # i**3  4**3 = 64
print(li[2](3))  # i**3  4**3 = 64
print(li[3](3))  # i**3  4**3 = 64
print(li[4](3))  # i**3  4**3 = 64

# 如何改进是输出结果为： 0 1 8 27 64

def f():
    li = []
    for i in range(5):
        # li.append(lambda x:i**x)
        def f2(x,y=i):
            return y**x
        li.append(f2)
    return li


li = f()
print(li[0](3))  # i**3
print(li[1](3))  # i**3
print(li[2](3))  # i**3
print(li[3](3))  # i**3
print(li[4](3))  # i**3
```