Python函数基础(下)

一、函数基本类型

1. 无参数，无返回值

   一般用于提示信息打印

2. 无参数，有返回值

   多在数据采集中，比如获取系统信息。

3. 有参数，无返回值

   多在设置某些不需要返回值的参数设置。

4. 有参数，有返回值

   一般就是计算型的，需要参数，最终也要返回结果。

```
# 无参数，无返回值
def myprint():
    print("-"*20)

# 无参数，有返回值
def mycpu():
    # 获取cpu信息
    return info

# 有参数，无返回值
def set(a):
    pass

# 有参数，有返回值
def cal(a,b):
    c = a+b
    return c
```

二、局部变量

> 局部变量，就是在函数内部定义的变量

不同的函数，可以定义相同名字的局部变量，但是每个函数内部不会产生影响;

局部变量的作用，是为了临时保存数据，需要在函数中定义变量来进行存储。

```
def func1():
    a = 10
    print(a)

def func2():
    a = 20
    print(a)

func1()   # 函数执行打印  10
func2()   # 函数执行打印  20
```

三、全局变量

1.定义

> 如果一个变量可以在函数内部调用，也可以在函数外部调用，那么这个变量就称为全局变量。

```
a = 100  # 定义一个全局变量
def function():
    # 函数内部可以访问a这个全局变量
    print('function ---->%s'%a)
print(a) # 函数外部打印a
function()
# 输出
#100
#function ---->100
```

> 全局变量与局部变量重复,局部变量不会影响全局变量，函数使用局部变量。

```
a = 100 # 定义一个全局变量
def function():
    a = 200
    print('function 局部变量---->%s'%a)
    a += 100
    print('function 局部变量 +100 ---->%s'%a)

function()
# 输出结果
# function 局部变量---->100
# function 局部变量 +100 ---->300

print(a) # 打印全局变量
# 100
```

> 全局变量能在函数内部访问，也可以在外部访问，但函数内部不能修改全局变量。

```
a = 100
a+=100
print('全局变量 a=%s'%a)
def function()
    print('访问全局变量%s'%a)
    a += 100  # 函数内部直接修改全局变量
function()
```

2.global 语句

> 如果一定要在函数内部修改全局变量需要用global声明

```
a = 100
a += 100
print('调用函数之前',a)
def function():
    global a
    a += 100
function()
print('调用函数之后',a)
```

> 可变类型的全局变量，可以不用global声明。

```
li = ['lilei','xiaoming']
def function():
    li.append('hanmeimei')
print(li)
function()
print(li)
```

执行结果

```
print(li)
['lilei', 'xiaoming']
function()
print(li)
['lilei', 'xiaoming','hanmeimei']
```

> 不可变类型的全局变量，每次修改都是创建一个新的对象，并不是修改变量的内容，如果不使用global声明不可变类型的全局变量引用指向就无法修改。可变类型的内容是可以被修改的，所以修改可变类型的全局变量，并没有改变其引用指向。

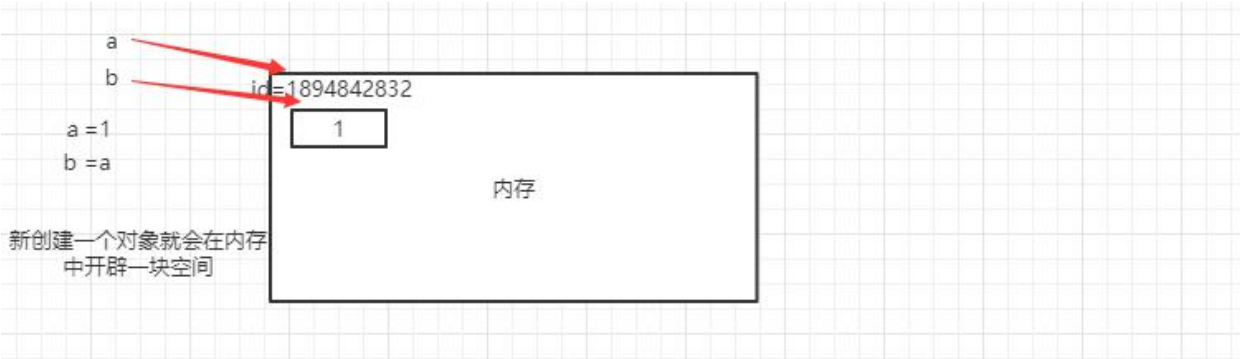
> - 全局变量是位于模块文件内部的顶层的变量名
> - 全局变量如果在函数内部被赋值的话，必须经过声明
> - 全局变量名在函数内部不经过声明也可以被引用。

四、引用

在python中，值是靠引用来传递来的，可以用id()查看一个对象的引用是否相同，id是值保存在内存中那块内存地址的标识。

```
>>> a = 1
>>> b=a
>>> id(a)
10919424
>>> id(b)
10919424
>>>
>>>
>>> a = [1,2,3]
>>> b=a
>>> b.append(4)
>>> a
[1, 2, 3, 4]
>>> b
[1, 2, 3, 4]
>>> id(a)
140033920404552
>>> id(b)
140033920404552
```

> 引用与内存地址的关系



> 可变类型与不可变类型

可变类型：

字典(dict)

列表(list)

```
>>> a = [1,2,3,4]
>>> id(a)
140033920379528
>>> a.append(5)
>>> id(a)
140033920379528
>>> b ={'name':'hanmeimei'}
>>> id(b)
140033913309960
>>> b['age']=18
>>> id(b)
140033913309960
```

不可变类型：

字符串（str）

数字（number）

元祖（tuple）

修改不可变类型的值实际上是新创建了一个对象。

```
>>> a = "a"
>>> id(a)
140033944362544
>>> a += 'b'
>>> id(a)
140033912903360
```

五、匿名函数

> python中使用lambda关键字创建匿名函数，所谓匿名即这个函数没有名字不用def关键字创建标准的函数。
> lambda创建匿名函数

1、python中使用lambda关键字创建匿名函数，所谓匿名即这个函数没有名字不用def关键字创建标准的函数。
lambda创建匿名函数

```
lambda 参数1,参数2,参数3: 执行代码语句
```

使用lambda表达式计算两个数和

```
sum = lambda x,y:x +y
sum(1,3)
sum(4,5)
```

执行结果4,9 换成普通函数其实就是这样的

```
def sum(x,y):
    return x + y
```

> lambda可以接收多个参数，但是只能返回一个值。要调用匿名函数得先赋值给一个变量，通过变量才能调用。

2、lambda表达式可以当一个参数传递。

```
def sum(a,b,lam):
    print(a)
    print(b)
    print(lam(a,b))

a = 1
b = 2
sum(a,b,lambda x,y:x+y)


# 输出结果
#1
#2
#3
```

3、lambda与三元运算

如下语句

```
if a:
    b
else:
    c
```

能够由以下等效的表达式来模拟：

```
b if a else c
```

这样的表达式（三元运算）能够放在lambda中，它们能够在lambda函数中来实现选择逻辑：

```
greater = (lambda x, y: x if x > y else y)
greater(3, 5)
greater(6, 2)
```

六、递归函数

1. 定义

> 递归函数 如果一个函数在内部不调用其它的函数，而是自己本身的话，这个函数就是递归函数。示

例：　求阶乘 5! = 5*4*3*2*1

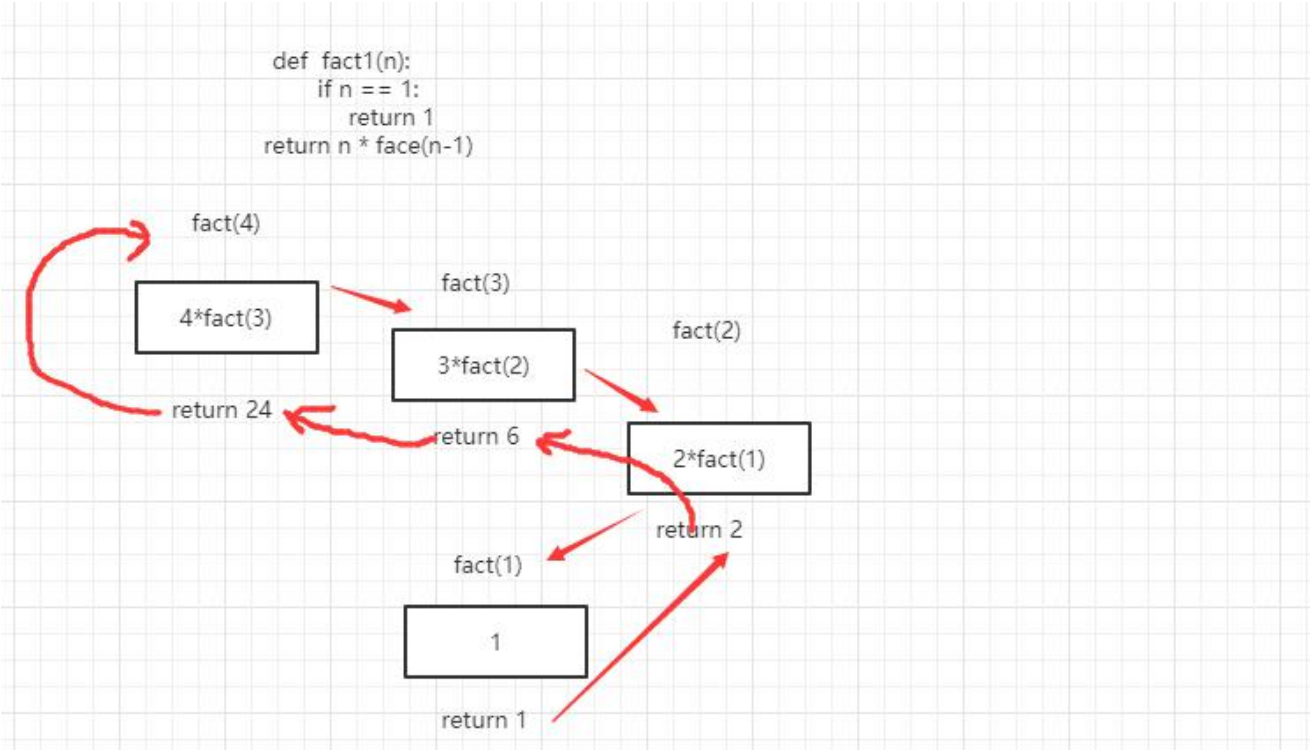
使用循环实现：

```
def fact(n):
    if n == 1:
        return 1
    else:
        num = 1
        while n > 1:
            num *= n
            n -= 1
        return num
```

递归实现：

```
def fact1(n):
    if n == 1:
        return 1
    return n * face(n-1)
```

递归调用过程



2. 递归函数必须有一个结束条件，否则递归无法结束会一直递归下去，只到到达最大递归深度报错。

3. 递归的优缺点：

优点：

1. 递归使代码看起来更加整洁、优雅

2. 可以用递归将复杂任务分解成更简单的子问题
3. 使用递归比使用一些嵌套迭代更容易

缺点：

1. 递归逻辑很难调试，递归条件处理不好容易造成程序无法结束，直到达到最大递归错误。
2. 递归占用大量内存，耗费计算机资源。