14.函数式编程

一、函数概念和调用

1. 概念
   在程序设计中，函数是指用于进行某种计算的一系列语句的有名称的组合。定义一个函数时，需要指定函数的名称并写下一系列程序语句。之后，就可以用名称来“调用”这个函数。
2. 调用函数
   函数名加括号就可以调用函数，如果函数需要传递参数，那么就传入对应的参数即可。

二、定义函数（创建函数）

1. 定义函数
   - 使用def语句定义一个函数，def语句后面跟着函数的名称，然后是一对小括号，括号里面可以填写函数需要的参数，最后是冒号结束，然后下一行缩进开始写函数体的内容。

```
def foo():  # foo是函数的名称
    print("hello world")
```

   - 函数的命名规范跟变量的命名规范一样，开始只能是字母或下划线，后面可以是数字、字母和下划线。注意的是函数名称不要跟系统的关键字以及自带的内建函数重名。
2. 参数和返回值
   - 函数的参数就是一个变量名，这个变量名可以在函数的内部使用，它的值是从外面传入进来的。定义函数时的变量名一般称为形参，调用函数时传入的值叫实参。

```
def hello(name):  # 这里的name就是函数的参数，叫形参
    print("hello %s" % name)

hello("brooks")  # brooks这个字符串就是实参，实际传入的参数
```

   - 函数的参数可以有很多个，传入函数参数的数量一定要跟定义函数时一致，顺序也是要对应，如果顺序不对应，那么可能得到的结果就不是预期的那样了。

```
def add(start, end):  # 有参数没有返回值
    num_list = range(start, end)
    total = 0
    for i in num_list:
        total += i
    print("从%d加到%d=%d" % (start, end, total))
```

- 函数的参数可以给定默认值，给有默认值的参数必放在没有默认值参数列表的后面。如果有默认值，那么在函数调用的时候可以不填，不填的话就使用默认的值。

```
def add(start, end, step=1):  # 有参数没有返回值
    num_list = range(start, end, step)
    total = 0
    for i in num_list:
        total += i
    print("从%d加到%d=%d" % (start, end, total))
```

- 使用return语句返回内容到函数外,return语句可以返回所有的数据类型，可以返回一个或多个。如果没有写return语句，函数的默认返回值是None

```
def myname():  # 没有参数，有返回值
    return "brooks"

def add(a, b):  # 有参数有返回值
    # 返回a+b的值
    return a + b

def append(a, b):  # 有参数有返回值
    # 将b添加到a中，然后返回a
    a.append(b)
    return a

def sizeyunsuan(a, b):  # 有参数有返回值
    add = a + b
    sub = a - b
    mux = a * b
    div = a / b
    # 这里看起来时返回了多个数值，但其实是将这些返回值组成了一个元组进行返回的
    return add, sub, mux, div
```

- 关键字参数和非关键字参数
  在函数声明的时候，如果我们给定了参数一个默认值，那么这个参数也称为关键字参数，同时我们在调用函数的时候也可以通过变量名来给对应的参数赋值，不管这个参数是否有默认值。但是需要注意的是，没有指定变

量名的参数必需是在前面的。
在函数声明和调用时没有指定变量名的参数就是非关键字参数。

```
def add(start, end, step=1):  # 有参数没有返回值
    num_list = range(start, end, step)
    total = 0
    for i in num_list:
        total += i
    print("从%d加到%d=%d" % (start, end, total))

add(10, start=2)
```

- 接收可变长参数
  当我们不能确定用户到底输入多少个参数的时候，那么我们就不能在函数定义的时候确定参数的个数，这样的话就不能很好的处理用户的输入。python提供了一种很好的办法来解决这个问题。使用*arg来接收多个非关键字参数，用**kargs来接收多个关键字参数。

```
def add(*arg):
# 使用*arg来接收多个参数，arg是变量名，可以自己随便取，只要变量名前面有一个*号就可以了
    for i in arg:
        print i

def add(**kws):
    # 变量名前面用两个*号表示接收的是关键字参数
    print kws
```

3. 函数的四种类型
   - 没有参数，没有返回值
   - 没有参数，有返回值
   - 有参数，没有返回值
   - 有参数，有返回值

三、匿名函数lambda

当我们需要实现一些很小的功能的时候，如果使用def声明一个函数就会显得比较复杂了，这种情况下就可以使用lambda来为我们创建一些小的功能函数。

1. lambda的语法
   lambda [参数列表]: 表达式
   说明：参数列表是可选的，就是说可以不填写参数，表达式就是函数的返回值，一般如果有参数的话，那么表达式中也会用到参数。表达式的结果就是函数的返回值

```
s = lambda : True  # 返回True
s1 = lambda x, y: x + y  # 返回x+y的值

### 字典排序的案例
d1 = {'a': 10, 'b': 23, 'c':3, 'd': 9}
sr = sorted(d1.items(), key=lambda x: x[1])  # 根据字典的值进行排序
```

四、函数里面定义函数

python允许函数里面再定义函数，在函数里面定义的函数只能在函数体里面使用，而不能在外面使用，要想在外面使用里面定义的函数，那么必须把里面的函数返回出去才能使用。

```
def foo():
    def coo():
         print("hello")
    coo()
    print("world")

foo()

### 将里面的函数返回到外面
def func():
    def add(x, y):
        return x + y
    return add

add = func()
add(2, 4)
```

五、变量作用域

变量的作用域是定义为其声明在程序里的可用范围，就是我们所说的变量的可见性，也就是说这个变量在哪里可以用，哪里不能用。python变量的作用域主要有两种：全局作用域和局部作用域，拥有全局作用域的变量又叫全局变量，拥有局部作用域的变量叫局部变量。

python变量的作用域是对于单个python文件而言的，因为当涉及到多个文件或者模块的时候，需要引入另一个概念“命名空间”。（讲到python模块和包的时候再讲）

一般的，在单个python文件里，如果变量声明在函数体之外的，都是全局变量，全局变量的作用域在声明它的那一行开始，一直到文件的结束。如果是在函数体内声明的变量叫局部变量，局部变量只能在函数体内使用。如果函数体内的局部变量名

称与全局变量的名称相同，那么局部变量将会覆盖全局变量，也就是说会优先使用局部的，而不会使用全局的。

```
a = 10  # 全局变量
def foo():
    a = 11  # 局部变量
    b = 20  # 局部变量
    print(a + b)
print(a)
```

循环语句和条件判断语句中定义的变量也是全局变量，但是不建议这么做，在循环语句和条件判断语句外使用在其中定义的变量会报一个警告信息（变量可能为定义就被引用，主要原因在于条件语句和循环语句都是需要满足一定的前提下才会执行的，因此就会存在条件不满足的情况，所以才会报警告）。

```
a = 0
while a < 10:
    a += 1
    b = 20

for x in range(3):
    y = x + 1
if a > 0:
    num = 100  # 如果num在else语句中也定义了，那么就没事
else:
    # num = 10
    snum = 100

print(a)
print(b)
print(x)
print(y)
print(num)
print(snum)
```

要想在函数内明确使用的是全局变量而不是重新定义个局部变量，那么就需要使用global关键字声明函数内的这个变量是全局变量不是局部变量。
global语句语法： global 变量名
变量名可以是一个或多个，每个名字直接用逗号隔开。

```
a = 10
b = 20
def foo():
    global a
    a  = 100

def func():
    global a, b
    a += 1
    b += a
    return b
```

六、闭包

如果在一个内部函数里，对外在作用域（但不是在全局作用域）的变量进行引用，那么内部函数就被认为是闭包（closure）。定义再外部函数内的但由内部函数引用或者使用的变量被称为自由变量。

```
def foo():
    x = [0]
    def fun():
        x[0] += 1
        return x[0]
    return fun

add_one = foo()
print(add_one())
print(add_one())
print(add_one())
print(add_one())
print(add_one())
```

闭包在实际的运用比较少，这里大家只是作为了解就可以了。如果还不理解的也没有关系，先把这个知识点放下，等自己的经验积累到一定程度之后再回头来进行理解。

七、递归

递归就是函数直接或间接地调用自身以进行循环的函数。递归是颇为高级的话题，并且在python中相对少见，但是它是一项应该了解的有用的技术，而且在我们的爬虫开发中还是经常用到的。
递归案例：计算阶乘

一个数的阶乘等于它和之前所有数的乘积，比如3的阶乘等于3 2 1

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

factorial(5)
# 执行流程
# 5 * factorial(4)
# 4 * factorial(3)
# 3 * factorial(2)
# 2 * factorial(1)
# 1
# 开始返回
# 1 * 2
# 1 * 2 * 3
# 1 * 2 * 3 * 4
# 1 * 2 * 3 * 4 * 5
```

但是用递归来实现阶乘，其实并不是一个很好的方法，因为完全可以用循环来实现。用递归实现只是为了说明递归的原理。

```
# 循环实现阶乘
def factorial(n):
    result = 1
    while n > 1:
        result *= n
        n -= 1
    return result
```

还有一个比较明显的，但是又不是很好的使用阶乘来解决问题的案例，那就是计算斐波那契数列的第N项。斐波那契数列就是从第3项开始，每一项都等于前两项之和（1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233，377，610，987，1597，2584，4181），如果设F(n）为该数列的第n项（n∈N*），那么这句话可以写成如下形式:
F(n)=F(n-1)+F(n-2)

```
def fibonacc(n):
    if n <= 2:
        return 1
    return fibonacc(n-1) + fibonacc(n-2)
```

用递归的方式计算斐波那契数列是非常不明智的，而且也是非常耗时的。因为有很多的项都重复计算了（冗余计算），比如在计算fibonacc(10)时fibonacc(3)的值被计算了21次，在计算fibonacc(30)的时候，fibonacc(3)被计算了317811次，而这些计算的结果都是一样的。因此用递归来计算斐波那契数列项是一项非常不明智的行为，

而这些复杂的计算仍然可以通过循环来实现。

```
# 使用循环实现斐波那契数列
def fibonacc(n):
    result = prev_result = next_result = 1
    while n > 2:
        next_result = prev_result
        prev_result = result
        result = prev_result + next_result
        n -= 1
    return result
```

递归在爬虫中的应用，在爬虫中无法避免的就是进行网络请求，那么网络请求无法避免的就是可能会有各种请求失败问题，那么对于请求失败的话我们需要进行重新请求，如果使用while循环的话，代码看起来就没那么友好了。所以一般情况下都是使用递归的方式进行的。下面是一个简单的例子：

```
import urllib2

def get_html(url, retries=5):
    try:
        req = urllib2.urlopen(url, timeout=5)
    except urllib2.HTTPError:
        html = None
        if retries > 0:
            return get_html(url, retries - 1)
    else:
        html = req.read()
    return html
```

八、生成器

python生成器是一个带yield语句的函数，return语句只返回一次，但是一个生成器能暂停执行并返回一个中间结果--那就是yield语句的功能，返回一个值给调用者并暂停执行。当调用生成器的next()方法时，它会准确的从离开的地方继续。
当函数没有更多的返回值时就会抛出StopIteration异常。

```python
def foo():
    yield 0
    yield "hello"

# 模拟range函数
def myrange(n):
    x = 0
    while n > 0:
        n -= 1
        yield x
        x += 1
```

作业

1. 将前面的关键词去重和关键词分类封装成两个个函数
2. 将前面的第二个爬虫封装成函数
3. 熟悉递归函数的使用，并使用递归的方法实现一个计算N层汉诺塔移动一共需要多少步的函数。

   > 汉诺塔相关资料： **https://baike.baidu.com/item/%E6%B1%89%E8%AF%BA%E5%A1%94/3468295**
   > 汉诺塔小游戏： **http://www.4399.com/flash/109504_1.htm**

4. 预习python文件操作相关函数。