nsd_1908_py02_day02

函数

参数

- 只写参数名，如args，称作位置参数
- 写成key=val形式的参数，称作关键字参数

```
>>> def get_age(name, age):
...     print('%s is %s years old' % (name, age))
...
>>> get_age('tom', 20)    # OK
tom is 20 years old
>>> get_age(20, 'tom')  # OK，但是语义不对
20 is tom years old
>>> get_age(age=20, name='tom')  # OK
tom is 20 years old
>>> get_age('tom', 20, 30)  # Error，参数太多
>>> get_age('tom')  # Error，参数不足
>>> get_age(age=20, 'tom')  # Error，位置参数需要在关键字参数前面
>>> get_age(20, name='tom')  # Error，name得到了多个值
>>> get_age('tom', age=20)  # OK
```

参数组

- 创建函数时，在参数名前加*，表示参数是元组
- 创建函数时，在参数名前加**，表示参数是字典

```
>>> def func1(*args):
...     print(args)
...
>>> func1()
()
>>> func1('hao')
('hao',)
>>> func1('hao', 123)
('hao', 123)
>>> def func2(**kwargs):
...     print(kwargs)
...
>>> func2()
{}
>>> func2(name='tom', age=20)
{'name': 'tom', 'age': 20}
```

- 调用函数时，在序列对象前加*表示将序列对象拆分开
- 调用函数时，在字典对象前加**表示将字典对象拆分成key=val的形式

```
>>> def get_age(name, age):
...   print('%s is %s years old' % (name, age))
...
>>> user = ['bob', 25]
>>> get_age(user[0], user[1])
bob is 25 years old
>>> get_age(*user)
bob is 25 years old
>>> user_dict = {'name': 'tom', 'age': 20}
>>> get_age(**user_dict)
tom is 20 years old
>>> get_age(name='tom', age=20)
tom is 20 years old
```

匿名函数

```
>>> def add(x, y):
...   return x + y
...
>>> add(10, 3)
13
# 可以改写成以下的匿名函数形式
>>> myadd = lambda x, y: x + y
>>> myadd(10, 20)
30
```

filter函数

- filter函数接受两个参数，第一个参数是函数，第二个参数是序列对象
  - (filter的参数)函数接受一个参数，返回值必须是True或False
  - 将序列对象中的每一个数据交给函数处理，返回真的留下，假的丢弃

```
from random import randint

def func1(x):
    return True if x % 2 == 1 else False

if __name__ == '__main__':
    nums = [randint(1, 100) for i in range(10)]
    print(nums)
    result = filter(func1, nums)
    print(list(result))
    result1 = filter(lambda x: True if x % 2 == 1 else False, nums)
    print(list(result1))
```

map函数

- map函数接受两个参数，第一个参数是函数，第二个参数是序列对象
  - 函数接受一个参数，对参数进行加工处理，返回处理结果
  - map调用时，将序列对象逐一交给函数处理

```
from random import randint

def func2(x):
    return x * 2

if __name__ == '__main__':
    nums = [randint(1, 100) for i in range(10)]
    print(nums)
    result = map(func2, nums)
    print(list(result))
    result1 = map(lambda x: x * 2, nums)
    print(list(result1))
```

变量

- 在函数外面的变量是全局变量。全局变量从定义开始，到程序结束，一直可见可用

```
>>> x = 10
>>> def func1():
...   print(x)
...
>>> func1()
10
```

- 在函数内部的变量是局部变量，只能在函数内部使用

```
>>> def func2():
...   y = 100
...   print(y)
...
>>> func2()
100
>>> y  # NameError
```

- 全局和局部拥有同名变量，函数调用时，局部变量遮盖住了全局变量

```
>>> x = 10
>>> def func3():
...   x = 200
...   print(x)
...
>>> func3()
200
>>> x
10
```

- 如果需要在函数内部修改全局变量，需要在函数内部使用global语句声明

```
>>> x = 10
>>> def func4():
...   global x
...   x = 100
...   print(x)
...
>>> func4()
100
>>> x
100
```

偏函数

- 改造现有函数，将函数的一些参数固定下来，生成新函数

```
>>> def add(a, b, c, d, e):
...   return a + b + c + d + e
...
>>> add(10, 20, 30, 40, 3)
103
>>> add(10, 20, 30, 40, 5)
105
>>> add(10, 20, 30, 40, 9)
109
>>> from functools import partial
>>> myadd = partial(add, 10, 20, 30, 40)
>>> myadd(3)
103
>>> myadd(5)
105


>>> int('10101100', base=2)  # 说明数字字符串是2进制数
172
>>> int('11000000', base=2)
192
>>> int2 = partial(int, base=2)  # 改造int函数，指定base=2
>>> int2('11000011')
195
```

递归函数

- 一个函数内部又包含了对自身的调用，就是递归函数

```
5!=5x4x3x2x1
5!=5x4!
5!=5x4x3!
5!=5x4x3x2!
5!=5x4x3x2x1!
1!=1
```

生成器

- 本质上是一个函数
- 可以通过yield语句返回很多中间结果

```
>>> def mygen():
...     yield 100
...     a = 10 + 20
...     yield a
...     yield 1000
>>> mg = mygen()
>>> mg
<generator object mygen at 0x7fdcc9ec4a98>
>>> list(mg)
[100, 30, 1000]
# 生成器只能使用一次，产生一次值，再次使用需要重新赋值
>>> for i in mg:
...     print(i)    # 输出为空
...
>>> mg = mygen()
>>> for i in mg:
...     print(i)
...
100
30
1000
```

模块

```
# 导入模块时，python从sys.path定义的路径下搜索模块
>>> import sys
>>> sys.path
['', '/usr/local/lib/python36.zip', '/usr/local/lib/python3.6',
'/usr/local/lib/python3.6/lib-dynload', '/root/nsd1908/lib/python3.6/site-packages']

# 如果希望自己编写的模块能在任意思位置导入，可以定义PYTHONPATH环境变量
(nsd1908) [root@room8pc16 day02]# mkdir /tmp/mylibs
(nsd1908) [root@room8pc16 day02]# cp qsort.py /tmp/mylibs
(nsd1908) [root@room8pc16 day02]# export PYTHONPATH=/tmp/mylibs
```

- python将目录当成特殊的模块，称作包（在python2中，目录下必须有一个名为__init__.py的文件才能成为包）

```
(nsd1908) [root@room8pc16 day02]# mkdir mydir
(nsd1908) [root@room8pc16 day02]# vim mydir/star.py
hi = 'Hello World!'

def pstar(n=30):
    print('*' * n)

if __name__ == '__main__':
    pstar()
(nsd1908) [root@room8pc16 day02]# python
>>> import mydir.star
>>> mydir.star.hi
'Hello World!'
>>> mydir.star.pstar()
******************************
```