nsd1905_py02_day02

关键字参数

函数的参数写为key=val这种形式，称作关键字参数，只有一个参数，如arg，称作位置参数。

```
>>> def func1(name, age):
...   print('%s is %s years old' % (name, age))
>>> func1('bob', 20)  # OK
bob is 20 years old
>>> func1(20, 'bob')  # 语法正确，语义不对
20 is bob years old
>>> func1(age=20, name='bob')  # OK
bob is 20 years old
>>> func1(age=20, 'bob')  # 语法错误，位置参数必须在关键字参数前
  File "<stdin>", line 1
SyntaxError: positional argument follows keyword argument
>>> func1(20, name='bob')  # 错误，name得到了多个值
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: func1() got multiple values for argument 'name'
>>> func1('bob', age=20)  # OK
bob is 20 years old
```

参数组

- 定义参数时，参数前加*表示使用元组接收参数
- 定义参数时，参数前加**表示使用字典接收参数

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

- 传参时，参数前加*表示把序列对象拆开
- 传参时，参数前加**表示把字典拆开

```
>>> def add(x, y):
...   return x + y
...
>>> nums = [1, 2]
>>> nums2 = (10, 20)
>>> add(*nums)
3
>>> add(*nums2)
30
>>> nums3 = {'y': 100, 'x': 5}
>>> add(**nums3)   # add(y=100, x=5)
105
>>> nums4 = {'a': 10, 'b': 3}
>>> add(**nums4)   # add(a=10, b=5)  # Error，没有参数a/b
```

匿名函数

- 一般函数使用def定义，def后面是函数名
- 匿名函数就是没有名字的函数
- 使用lambda关键字定义

```
>>> def add(x, y):
...   return x + y
...
>>> myadd = lambda x, y: x + y
>>> myadd(1, 2)
3
>>> add(1, 2)
3
```

filter(func, seq)函数

- 是一个高阶函数，它的第一个参数是函数，第二个参数是序列对象
- 传给filter函数的函数（第一个参数），它接受一个参数，执行的结果必须为True或False
- 序列对象中每一个元素分别作为函数的参数，计算结果为True则保留，为False舍弃

```
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))
    result2 = filter(lambda x: True if x % 2 == 1 else False, nums)
    print(list(result2))
```

map(func, seq)函数

- map是一个高阶函数，它的第一个参数是函数，第二个参数是序列对象
- 序列对象中的每个元素都将作为函数的参数进行处理，处理的结果全部保存下来

```
def func1(s):
    return s + '.com'

if __name__ == '__main__':
    alist = ['qq', 'sohu', '163']
    result = map(func1, alist)
    print(list(result))
    result2 = map(lambda s: s + '.com', alist)
    print(list(result2))
```

变量

全局变量：在函数外定义的变量，全局变量从它定义的位置到程序结束，一直可见可用

局部变量：在函数内定义的变量，只能在函数内使用

```
>>> x = 10
>>> def func1():
...   print(x)
...
>>> func1()
10
>>> def func2():
...   y = 100
...   print(y)
...
>>> func2()
100
>>> print(y)  # 报错，局部变量只能在函数内使用

# 如果局部和全局有同名变量，局部变量将会遮盖住全局变量的值
>>> def func3():
...   x = 'hello world'
...   print(x)
...
>>> func3()
hello world
>>> print(x)
10

# 如果需要通过局部改变全局变量，需要使用global关键字
>>> def func4():
...   global x
...   x = 100
...   print(x)
...
>>> func4()
100
>>> print(x)
```

```
100
```

偏函数

- 改造现有函数，将现有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, 5)
105
>>> add(10, 20, 30, 40, 2)
102
>>> add(10, 20, 30, 40, 8)
108

# 改造add函数，将前4个参数固定下来，生成新的函数
>>> from functools import partial
>>> myadd = partial(add, 10, 20, 30, 40)
>>> myadd(5)
105
>>> myadd(8)
108

>>> int('11', base=2)  # 将2进制数11转为10进制数
3
>>> int('11000011', base=2)
195
>>> int2 = partial(int, base=2)
>>> int2('11')
3
>>> int16 = partial(int, base=16)
>>> int16('11')
17
```

递归函数

- 函数调用自己，就是递归函数

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

生成器

生成器潜在可以生成很多数据，但是不会立即生成。运行时节约空间。

实现的方法有两种：生成器表达式和函数的形式

```
```

```
>>> nums = (randint(1, 100) for i in range(10))
>>> nums
<generator object <genexpr> at 0x7f0dd3a19308>
>>> list(nums)
[32, 53, 83, 72, 5, 71, 26, 36, 90, 44]
# 生成器对象只能用一次，再用就没有值可取了
>>> nums = (randint(1, 100) for i in range(10))
>>> for i in nums:
...   print(i)

# 生成器函数可以通过yield返回很多中间结果
>>> def scq():
...   yield 10
...   n = 10 + 20
...   yield n
...   yield 100
...
>>> a = scq()
>>> list(a)
[10, 30, 100]

>>> b = scq()
>>> for i in b:
...   print(i)
```

模块

导入模块时，python从sys.path定义的路径查找模块

```
>>> sys.path   # 空串表示当前目录
['', '/usr/local/lib/python36.zip', '/usr/local/lib/python3.6',
'/usr/local/lib/python3.6/lib-dynload', '/root/nsd1905/lib/python3.6/site-packages']
```

自定义模块，如果希望在任何位置都可以导入，那么，可以把模块拷贝到site-packages目录。也可以通过环境变量PYTHONPATH定义路径

```
(nsd1905) [root@room8pc16 day02]# export PYTHONPATH=/path/to/module
# 把自己写的模块放到/path/to/module，在任意位置就都可以导入自定义模块了
```

python把目录当成特殊的模块

```
(nsd1905) [root@room8pc16 py02]# pwd
/var/ftp/nsd2019/nsd1905/py02
(nsd1905) [root@room8pc16 py02]# ls day02/qsort.py
>>> import day02.qsort
>>> day02.qsort.qsort('hello')
['e', 'h', 'l', 'l', 'o']
```

hashlib模块

```
# 计算数据的md5值
>>> import hashlib
>>> m = hashlib.md5(b'123456')
>>> m.hexdigest()
'e10adc3949ba59abbe56e057f20f883e'

# 计算文件的md5值
>>> with open('/etc/hosts', 'rb') as fobj:
...   data = fobj.read()
...
>>> m1 = hashlib.md5(data)
>>> m1.hexdigest()
'1b8f16e6e10c1a822dfc36cc4256344d'

>>> m2 = hashlib.md5()
>>> m2.update(b'12')
>>> m2.update(b'34')
>>> m2.update(b'56')
>>> m2.hexdigest()
'e10adc3949ba59abbe56e057f20f883e'
```

tarfile模块

实现tar包操作

```
# 打包并使用gz压缩
>>> import tarfile
# 以写方式打开，并且启用gzip压缩
>>> tar = tarfile.open('/tmp/mytest.tar.gz', 'w:gz')
>>> tar.add('/etc/security')
>>> tar.add('/etc/hosts')
>>> tar.close()

# 解压到/var/tmp目录
>>> tar = tarfile.open('/tmp/mytest.tar.gz')
>>> tar.extractall(path='/var/tmp')  # 不写参数，解压到当前目录
>>> tar.close()
```