nsd_weekend_py0202

关键字参数

在定义函数时，参数的形式，如果直接写为一个参数，如arg，它被称作位置参数；如果写为key=val的形式，就被称作关键字参数。

```
>>> def get_info(name, age):
...   print('%s is %s years old.' % (name, age))
...
>>> get_info('bob', 22)  # OK
bob is 22 years old.
>>> get_info(22, 'bob')  # 语法正确的，语义不对
22 is bob years old.
>>> get_info(age=22, name='bob')  # OK
bob is 22 years old.
>>> get_info(age=22, 'bob')  # Error，关键字参数必须在位置参数之后
  File "<stdin>", line 1
SyntaxError: positional argument follows keyword argument
>>> get_info(22, name='bob')  # Error,name得到了多个值
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: get_info() got multiple values for argument 'name'
>>> get_info('bob', age=22)  # OK
bob is 22 years old.
>>> get_info()  # Error, 参数个数不够
>>> get_info('bob', 22, 100)  # Error, 参数太多
```

参数组

当参数个数不确定时，可以在参数前加一个*号，表示将参数放到元组中，还可以加**表示把参数放到字典中。

```
>>> def func1(*args):
...   print(args)
...
>>> func1()
()
>>> func1(10)
(10,)
>>> func1(10, 'bob', 20, 30, 40)
(10, 'bob', 20, 30, 40)

>>> def func2(**kwargs):
...   print(kwargs)
...
>>> func2(name='bob', age=22)
{'name': 'bob', 'age': 22}
```

```
>>> func2()
{}
```

调用函数时，也可以在参数前加*号，一个*表示将序列拆开，加**表示将字典拆开。

```
>>> def add(x, y):
...   print(x + y)
...
>>> nums = [10, 20]
>>> add(nums)  # Error, nums传给x，y没有得到值
>>> add(*nums)
30
>>> nums2 = {'x': 100, 'y': 200}
>>> add(**nums2)  # add(x=100, y=200)
300
```

匿名函数

```
>>> def add(x, y):
...   return x + y
...
>>> lambda x, y: x + y
<function <lambda> at 0x7f1b8b072b70>
>>> fn = lambda x, y: x + y
>>> fn(10, 20)
30
```

filter函数：它是一个高阶函数，filter的第一个参数是函数，第二个参数是序列。序列中的第一个值都会作为函数的参数传递，该函数必须返回True或False，返回True的值保留，否则丢弃。

```
>>> from random import randint
>>> nums = [randint(1, 100) for i in range(10)]
>>> nums
[55, 12, 75, 82, 6, 86, 56, 6, 82, 69]
>>> result = filter(lambda x: True if x % 2 == 0 else False, nums)
>>> list(result)
[12, 82, 6, 86, 56, 6, 82]
```

map函数：与fileter类似，但是它是加工序列中的数据，加工的结果全部保留。

```
>>> from random import randint
>>> nums = [randint(1, 100) for i in range(10)]
>>> nums
[55, 12, 75, 82, 6, 86, 56, 6, 82, 69]
>>> result2 = map(lambda x: x * 2, nums)
>>> list(result2)
[110, 24, 150, 164, 12, 172, 112, 12, 164, 138]
```

变量作用域

- 在函数外声明的变量是全局变量，它在定义开始到程序结束的任意位置一直可见可用
- 在函数内部定义的变量是局部变量，只能在函数内部使用
- 全局和局部有同名变量，局部变量将会遮盖住全局变量
- 在函数内改变全局变量的值，需要使用global关键字
- 函数的参数也是局部变量

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

>>> def func1():
...   a = 100
...   print(a)
...
>>> func1()
100
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined

>>> def func2():
...   x = 100
...   print(x)
...
>>> func2()
100
>>> x
10

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

偏函数

用于改造现有的函数，将其中的某些参数固定下来

```
>>> from functools import partial
>>> 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, 15)
115
>>> myadd = partial(add, 10, 20, 30, 40)  # 改造add函数，生成新函数myadd
>>> myadd(5)
105
>>> myadd(15)
115

>>> int('11')
11
>>> int('11', base=2)  # base=2，表示'11'是2进制
3
>>> int('11', base=8)
9
>>> int2 = partial(int, base=2)  # 改造int函数，将base固定为2
>>> int2('1100')
12
```

递归函数

一个函数的内部又对自身进行调用，就是递归函数。

生成器

生成器有两种形式，一种是生成器表达式，一种是函数。生成器的优势，是在数据量非常大的时候，节省内存空间。

生成器只能用一次，再用必须重新赋值。

生成器函数可以通过yield返回很多中间结果。创建生成器对象后，从生成器对象中取值，遇到yield就暂停执行，返回中间结果。

```
>>> nums = (randint(1, 100) for i in range(10))
>>> nums
<generator object <genexpr> at 0x7f1b87fa9728>
>>> for i in nums:
...     print(i)

>>> def mygen():
...     yield 100
...     n = 10 + 10
...     yield n
...     yield 200
...
>>> mg = mygen()
>>> for i in mg:
...     print(i)
...
100
```

```
20
200
```

模块

文件是物理上组织代码的形式，模块是逻辑上组织代码的形式。

导入模块

python导入模块时，会到sys.path定义的路径下搜索模块。还可以定义一个环境变量PYTHONPATH，PYTHON也会到这个环境变量定义的路径去搜索模块。

导入模块的方法

```
# 常用的方法
>>> import time
>>> from random import randint, choice

# 不常用的方法
>>> import os, time, random
>>> import pickle as p     # 导入模块时，为其起别名
```

可以把目录当成特殊的模块，叫作包。在python2版本中，一个目录如果想成为包，必须在目录中创建一个名为__init__.py的文件。

```
[root@room8pc16 py0202]# tree mypack/
mypack/
├── hello.py
[root@room8pc16 py0202]# cat mypack/hello.py
hi = "hello world"

>>> import mypack.hello
>>> mypack.hello.hi
'hello world'
```

hashlib模块

```
>>> import hashlib
>>> with open('/etc/passwd', 'rb') as fobj:
...   data = fobj.read()
...
>>> m = hashlib.md5(data)
>>> m.hexdigest()
'decb544ed171583bb1d7722500910d9e'

>>> m = hashlib.md5(b'123456')
>>> m.hexdigest()
'e10adc3949ba59abbe56e057f20f883e'
>>>
>>> m1 = hashlib.md5()
```

```
>>> m1.update(b'12')
>>> m1.update(b'34')
>>> m1.update(b'56')
>>> m1.hexdigest()
'e10adc3949ba59abbe56e057f20f883e'
```

tarfile模块

```
>>> import tarfile
>>> tar = tarfile.open('/tmp/myfiles.tar.gz', 'w:gz')
>>> tar.add('/etc/hosts')
>>> tar.add('/etc/security')
>>> tar.close()

[root@room8pc16 py0202]# mkdir /tmp/abcd
>>> tar = tarfile.open('/tmp/myfiles.tar.gz')
>>> tar.extractall(path='/tmp/abcd')
>>> tar.close()
```

os.walk

```
>>> list(os.walk('/etc/security'))
[('/etc/security', ['console.apps', 'console.perms.d', 'limits.d', 'namespace.d'],
['access.conf', 'chroot.conf', 'console.handlers', 'console.perms', 'group.conf',
'limits.conf', 'namespace.conf', 'namespace.init', 'opasswd', 'pam_env.conf',
'sepermit.conf', 'time.conf', 'pwquality.conf']), ('/etc/security/console.apps', [],
['config-util', 'xserver', 'liveinst', 'setup']), ('/etc/security/console.perms.d', [],
[]), ('/etc/security/limits.d', [], ['20-nproc.conf']), ('/etc/security/namespace.d', [],
[])]
# os.walk返回值由元组构成，元组有三项，第一项是路径，第二项是该路径下的目录列表，第三项是该路径下的文件列表
# 想得到文件的具体路径，可以把路径和文件名进行拼接
>>> for path, folders, files in os.walk('/etc/security'):
...   for file in files:
...     os.path.join(path, file)
```

OOP面向对象编程

OOP将现实世界中的事物进行抽象，找出他们的共同特点，编写成一个class类。这个class中有名词性质的属性，用变量表示；还有一些动作性质的属性，用函数表示。在class中的函数，叫作方法。

oop实现了数据和行为的统一、融合。

组合

当两个类有明显的不同，但是一个类是另一个类的组件，用组合。

继承

当两个类非常相似的时候用。子类可以有多个父类，称作多重继承，子类拥有所有父类的方法。子类的实例在执行方法时，查找的顺序是自下向上，自左向右，先找到的执行。

类的特殊方法

也被称作magic魔法方法，方法以双下划线开头、结尾，往往有特殊的作用。

```
```