nsd1903_py02_day02

函数

- 函数声明的顺序不重要，重要的是什么时候调用
- 函数的参数，只写成单一的名称，叫作位置参数；如果写成key=val形式，称作关键字参数

```
# 定义函数时，默认值参数必须在非默认值参数后面，否则将出现语法错误
>>> def func1(name='bob', age):
...   pass
...
  File "<stdin>", line 1
SyntaxError: non-default argument follows default argument

>>> def func1(name, age=20):   # 正确
...   pass
```

- 传递参数也有相应的约定

```
>>> def func1(name, age):
...   print('%s is %s years old' % (name, age))
>>> func1('tom', 20)   # OK
tom is 20 years old
>>> func1(20, 'tom')   # 语法正确，但是语义不正确
20 is tom years old
>>> func1(age=20, name='tom')  # OK
tom is 20 years old
>>> func1(age=20, 'tom')   # 语法错误
  File "<stdin>", line 1
SyntaxError: positional argument follows keyword argument
>>> func1(20, name='tom')   # 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('tom', age=20)  # OK
tom is 20 years old
```

- 参数组，当函数的参数不确定时可以使有参数组接收参数
    - *参数，表示使用元组接收参数
    - **参数，表示使用字典接收参数

```
>>> def func1(*args):
...   print(args)
...
>>> func1()
()
>>> func1('hao')
```

```
('hao',)
>>> func1('hao', 123, 'tom')
('hao', 123, 'tom')

>>> def func2(**kwargs):
...   print(kwargs)
...
>>> func2()
{}
>>> func2(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: func2() takes 0 positional arguments but 1 was given
>>> func2(name='tom', age=20)
{'name': 'tom', 'age': 20}

>>> def func3(*args, **kwargs):
...   print(args)
...   print(kwargs)
...
>>> func3()
()
{}
>>> func3(10, 20, 30, name='tom', age=20)
(10, 20, 30)
{'name': 'tom', 'age': 20}
```

- 传参的时候也可以加*号，表示把序列或字典拆开

```
>>> def add(a, b):
...   return a + b
...
>>> nums = [10, 20]
>>> add(nums)   # nums传递给a，b没有得到值
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: add() missing 1 required positional argument: 'b'
>>> add(*nums)
30
>>> mydict = {'a': 100, 'b':200}
>>> add(**mydict)  # 相当于add(a=100, b=200)
300
```

练习：数学加减法小程序

```
5 + 5 = 10
Very Good!!!
Continue(y/n)? y
42 + 26 = 50
Wrong Answer!!!
42 + 26 = 55
Wrong Answer!!!
42 + 26 = 60
Wrong Answer!!!
42 + 26 = 68
Continue(y/n)? n
Bye-bye
```

匿名函数：没有名字的函数。

- 代码只有一行
- 能过lambda关键字定义
- 参数直接写到lambda后面
- 表达式的计算结果是返回值

```
>>> def func1(x):
...   return x + 10
...
>>> func1(2)
12
>>> add10 = lambda x: x + 10
>>> add10(2)
12
```

filter函数

- 用于过滤数据
- 它接受两个参数
    - 第一个参数是函数，返回值必须是True或False
    - 第二个参数是序列对象
- 序列对象中的每一个元素传递给函数，结果为True的保留

```
>>> from random import randint
>>> nums = [randint(1, 100) for i in range(10)]
>>> nums
[93, 2, 11, 70, 16, 23, 89, 17, 47, 91]
>>> def func1(x):
...   return True if x > 50 else False
>>> list(filter(func1, nums))
[93, 70, 89, 91]
>>> list(filter(lambda x: True if x > 50 else False, nums))
[93, 70, 89, 91]
```

map函数

- 用于加工数据
- 接受两个参数
  - 第一个参数是函数，用于加工数据
  - 第二个参数是序列，序列中的每个对象作为前面函数的参数
- 将序列中的每个数据作为函数的参数，加工后返回

```
>>> def func2(x):
...   return x * 2
...
>>> list(map(func2, nums))
[186, 4, 22, 140, 32, 46, 178, 34, 94, 182]
>>> list(map(lambda x: x * 2, nums))
[186, 4, 22, 140, 32, 46, 178, 34, 94, 182]
```

变量的作用域

1. 定义在函数外面的是全局变量，全局变量从定义开始到程序结束，一直可见可用
2. 函数内部定义的变量是局部变量，只在函数内部可见可用
3. 如果局部和全局有同名变量，优先使用局部，局部变量遮盖住全局变量
4. 如果需要在局部改变全局变量，使有global关键字

```
# 1.
>>> x = 10
>>> def func1():
...   print(x)
...
>>> func1()
10

# 2.
>>> def func2():
...   y = 10
...   print(y)
...
>>> func2()
10
>>> print(y)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'y' is not defined

# 3.
>>> def func3():
...   x = 100
...   print(x)
...
>>> func3()
100
>>> print(x)
10
```

```
# 4.
>>> def func4():
...     global x
...     x = 1000
...     print(x)
...
>>> func4()
1000
>>> print(x)
1000
```

偏函数

改造现有函数，将其一些参数固定下来，生成新的函数。

```
>>> 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, 22)
122
>>> from functools import partial
>>> myadd = partial(add, 10, 20, 30, 40)
>>> myadd(10)
110
>>> myadd(5)
105

# int接受base指定字符串的进制
>>> int('11111111', base=2)
255
>>> int2 = partial(int, base=2)  # 改造int函数，将2进制转10进制
>>> int2('11111111')
255
```

递归函数

- 函数自己又调用自己

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

```
>>> def func(x):
...   if x == 1:
...     return 1
...   return x * func(x - 1)
...
>>> func(5)
120
>>> func(6)
720
```

快速排序

- 假设第一个数是中间值。 middle = nums[0]
- 比middle大的都放到larger列表中
- 比middle小的都放到smaller列表中
- 将三者拼接：smaller + [middle] + larger
- 使用相同的方法继续对smaller和larger排序
- 当列表只有一项或是空的，就不用再排了

生成器

- 潜在可以提供很多数据
- 不会立即生成全部数据，所以节省内存空间
- 可以通过生成器表达式实现
- 也可以通过函数实现

```
# 生成器表达式与列表解析的语法一样，只是使用()
>>> ips = ('192.168.1.%s' % i for i in range(1, 255))
>>> for ip in ips:
...   print(ip)

# 使用函数，通过yield多次返回中间值
>>> def mygen():
...   yield 100
...   n = 10 + 20
...   yield n
...   yield 'hello world'
...
>>> mg = mygen()
>>> for i in mg:
...   print(i)
...
100
30
hello world
```

模块

- 模块就是把一个python文件名去掉.py后的部分
- 导入模块时，python在sys.path定义的路径中搜索模块

hashlib模块

用于计算哈希值。哈希值如md5 / sha等，常用于加密密码和文件完整性校验。

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

# 如果数据量太大，可以创建md5对象后，每次更新一部分
>>> m1 = hashlib.md5()
>>> m1.update(b'12')
>>> m1.update(b'34')
>>> m1.update(b'56')
>>> m1.hexdigest()
'e10adc3949ba59abbe56e057f20f883e'

>>> with open('/bin/ls', 'rb') as fobj:
...   data = fobj.read()
...
>>> m = hashlib.md5(data)
>>> m.hexdigest()
'038a0a2d35a503d299a46096f6ff9890'
```

tarfile模块

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

# 解压缩到/tmp/demo
>>> tar = tarfile.open('/tmp/mytar.tar.gz')
>>> tar.extractall(path='/tmp/demo/')
>>> tar.close()
```

os.walk()

- 它可以递归列出某个目录下的所有内容
- 返回值由多个元组构成
    - ('路径字符串', [路径下目录列表], [路径下文件列表])
- 将路径字符串与文件拼接就可以得到文件的路径

```
>>> 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', [],
[])]
>>> result = list(os.walk('/etc/security'))
>>> len(result)
5
>>> result[0]
('/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'])
>>> result[1]
('/etc/security/console.apps', [], ['config-util', 'xserver', 'liveinst', 'setup'])
>>> result[2]
('/etc/security/console.perms.d', [], [])

>>> for path, folers, files in os.walk('/etc/security'):
...   for file in files:
...     os.path.join(path, file)
```