nsd1904_py02_day02

函数基础

- 定义函数的时候，参数它的值不确定，只是起个名称，形式上占个位置，所以叫形式参数，简称形参
- 调用函数的时候，将具体的数据传递进去，这个时候相当于是变量赋值，因为使用的数据值已经确定，是实际使用的参数，叫作实际参数，简称实参
- 多个函数创建的顺序不重要，重要的是调用顺序
- 函数的参数，如果是key=val的形式，称作关健字参数；如果是只有参数arg1,arg2...称作位置参数

```
>>> def func1(name, age):
...   print('%s is %s years old' % (name, age))
...
>>> func1()   # 参数不够
TypeError: func1() missing 2 required positional arguments: 'name' and 'age'
>>> func1('bob', 20, 30)   # 参数太多
TypeError: func1() takes 2 positional arguments but 3 were given
>>> func1('bob', 20)   # OK
>>> func1(20, 'bob')    # 语法正确，但是语义不对
>>> func1(age=20, name='bob')   # OK
>>> func1(age=20, 'bob')   # 语法错误，key=val的形式必须在后
SyntaxError: positional argument follows keyword argument
>>> func1(20, name='bob')   # name变量得到了多个值
TypeError: func1() got multiple values for argument 'name'
>>> func1('bob', age=20)   # OK
```

- 定义函数的时候，在参数前加一个*表示使用元组接收参数，加两个*表示用字典接收参数；调用函数的时候，在参数前加一个*表示把序列对象拆开，加两个*号表示把字典拆开

```
>>> def func1(*args):
...   print(args)
...
>>> func1()
()
>>> func1('hello')
('hello',)
>>> func1('hello', 123)
('hello', 123)

>>> def func2(**kw_args):
...   print(kw_args)
...
>>> func2()
{}
>>> func2(name='bob', age=20)
{'name': 'bob', 'age': 20}
```

```
>>> def func3(x, y):
...   return x + y
...
>>> nums = [10, 15]
>>> func3(nums[0], nums[1])
>>> func3(*nums)   # func3(10, 15)
25
>>> adict = {'x': 20, 'y': 100}
>>> func3(**adict)   # func3(x=20, y=100)
120
```

练习：简单的加减法数学游戏

```
10 + 5 = 15
Very Good!
Continue(y/n)? y
46 + 28 = 50
Wrong Answer
46 + 28 = 60
Wrong Answer
46 + 28 = 70
Wrong Answer
46 + 28 = 74
Continue(y/n)? n
Bye-bye
```

匿名函数

- 通过lambda关键字定义的、“没有名字”的函数
- 如果函数主体只有一行代码，可以使用匿名函数

```
>>> def add(x, y):
...   return x + y
...
>>> myadd = lambda x, y: x + y
>>> add(5, 2)
7
>>> myadd(5, 2)
7
```

filter函数

- 它接受两个参数，第一个参数是函数，该函数接收一个参数，返回值必须是True或False；第二个参数是序列对象。
- 工作时，序列对象它的每个数据都成为第一个函数的参数，如果结果为真则保留；如果结果为假则丢弃

map函数

- 它接受两个参数，第一个参数是函数，该函数接收一个参数，并对参数进行加工处理；第二个参数是序列对象。

- 工作时，序列对象它的每个数据都成为第一个函数的参数，加工后返回。

变量作用域

- 全局变量：在函数外面定义的变量。从定义开始到函数结束，一直是可见可用的。
- 局部变量：在函数内定义，只在函数内部使用，外界看不到这个函数
- 全局和局部有同名变量，函数内的变量将遮盖住全局变量
- 如果希望在局部改变全局变量，可以使用global关键字
- 函数在查找名称时，先在局部查找，再查全局，最后查内建

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

>>> def func2():
...   x = 'hello world'
...   print(x)
...
>>> func2()
hello world
>>> x
10

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

偏函数

- 通过functools中的partial进行函数改造
- 把函数的某些参数固定下来，生成新的函数

```
>>> def add(a, b, c, d):
...   return a + b + c + d
...
>>> add(10, 20, 30, 5)
65
>>> add(10, 20, 30, 6)
66
>>> add(10, 20, 30, 7)
```

```
67
>>> from functools import partial
>>> myadd = partial(add, 10, 20, 30)
>>> myadd(5)
65
>>> myadd(6)
66

>>> int('11111111', base=2)   # int的base可以指定进制
255
>>> int2 = partial(int, base=2)   # 改造int函数，生成新函数int2
>>> int2('11111111')
255
```

递归函数

如果一个函数又包含对自身的调用，就是递归函数。

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

生成器

- 生成器表达式
- 函数方式：生成器通过yield关键字生成很多中间结果

```
# 生成器表达式方式
>>> ('192.168.1.%s' % i for i in range(1, 11))
<generator object <genexpr> at 0x7fc943bf4b48>
>>> ips = ('192.168.1.%s' % i for i in range(1, 11))
>>> for ip in ips:
...   print(ip)

# 函数的方式
>>> def mygen():
...   yield 100
...   yield 'Hello World!'
...   n = 5 + 5
...   yield n
...
>>> mg = mygen()
>>> for i in mg:
...   print(i)
...
100
Hello World!
10
```

模块

- 导入模块时，python到sys.path定义的目录查找模块，查到则导入，否则报错

```
>>> import sys
>>> sys.path
['', '/usr/local/lib/python36.zip', '/usr/local/lib/python3.6',
'/usr/local/lib/python3.6/lib-dynload', '/root/nsd1904/lib/python3.6/site-packages']
```

- 如果希望自己写的模块文件像系统模块一样，在任何位置都可以导入，可以将文件拷贝到site-packages目录，或定义环境变量PYTHONPATH，指定自己的模块文件目录

hashlib模块

- 用于计算数据的哈希值

```
>>> import hashlib
>>> with open('/etc/passwd', 'rb') as fobj:
...   data = fobj.read()
...
>>> m = hashlib.md5(data)
>>> m.hexdigest()
'022583cb13c9d3f78c45c6f1795be1ff'
```

tarfile模块

- 实现压缩、解压缩
- 可以支持gzip、bzip2等

```
>>> import tarfile
# 压缩
>>> tar = tarfile.open('/tmp/nsd1904/mytest.tar.gz', 'w:gz')
>>> tar.add('/etc/hosts')
>>> tar.add('/etc/security')
>>> tar.close()

# 解压缩
>>> tar = tarfile.open('/tmp/nsd1904/mytest.tar.gz')
>>> tar.extractall(path='/tmp/nsd1904')
>>> tar.close()
```