nsd1909-py02-day02

函数

- 函数定义时，不执行其中的代码
- 所以函数定义的先后顺序无所谓

```
def func1():
    print('in func1')
    func2()

def func2():
    print('in func2')

func1()   # 不会报错，因为此时 func2已定义
```

- 函数的参数部分，直接给定一个名字，如arg，称作位置参数
- 函数的参数部分，给定的形式像key=val这样，称作关键字参数

```
>>> 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，语义不对
20 is tom years old
>>> func1(age=20, name='tom')   # OK
tom is 20 years old
>>> func1(age=20, 'tom')   # Error，关键字参数必须在位置参数后面
>>> func1(20, name='tom')  # Error，变量name得到了两个值
>>> func1('tom', age=20)   # OK
tom is 20 years old
>>> func1()                # Error，参数太少
>>> func1('tom', 20, 30)   # Error，参数太多
```

- 在定义函数时，参数名前加上*，表示使用元组接收参数

```
>>> def func1(*canshu):
...   print(canshu)
...
>>> func1()
()
>>> func1('tom')
('tom',)
>>> func1('tom', 20)
('tom', 20)
```

- 在定义函数时，参数名前加上**，表示使用字典接收参数

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

- 调用函数时，在参数名前加*表示将序列拆开

```
>>> def func3(x, y):
...   print(x + y)
...
>>> l = [10, 20]
>>> t = (100, 200)
>>> func3(*l)
30
>>> func3(*t)
300
```

- 调用函数时，在参数名前加**，表示把字典拆成key=val的形式

```
>>> def func4(name, age):
...   print('%s is %s years old' % (name, age))
...
>>> user = {'name': 'tom', 'age': 20}
>>> func4(**user)   # func4(name='tom', age=20)
tom is 20 years old
```

练习：简单的加减法数学游戏

运行方式：

```
1 + 1 = 2
Very Good!!!
Continue(y/n)? y
1 + 2 = 0
Wrong
1 + 2 = 1
Wrong
1 + 2 = 2
Wrong
1 + 2 = 3
Continue(y/n)? n
Bye-bye
```

匿名函数

- 当函数体只有一行代码时，可以通过关键字lambda创建匿名函数

```
>>> def add(x, y):
...   return x + y
...
# 以上函数可以改为以下形式：
>>> myadd = lambda x, y: x + y  # x,y是参数，x+y的结果自动成为返回值返回，不用 return
>>> myadd(10, 20)   # 调用函数
```

filter函数

- 它接受两个参数
- 第一个参数是函数，该函数接受一个参数，返回值必须是真或假
- 第二个参数是序列对象
- filter将序列中的每个值传给第一个函数，返回真保留，返回假过滤掉

```
from random import randint

def func1(x):
    '接受一个数字，奇数返回 True，偶数返回False'
    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)
    result2 = filter(lambda x: True if x % 2 == 1 else False, nums)
    print(list(result))
    print(list(result2))
```

map函数

- 它接受两个参数
- 第一个参数是函数，该函数接受一个参数，将该数据处理后返回
- 第二个参数是序列对象
- map将序列中的每个值传给第一个函数，处理结果保存

变量作用域

- 在函数外面定义的变量，是全局变量，全局变量在声明开始到程序结束，一直可见可用

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

- 函数内部的变量是局部变量，只能在函数内部使用

```
>>> def func2():
...   a = 10
...   print(a)
...
>>> func2()
10
>>> a  # 报错
```

- 函数内如果有和全局变量同名的局部变量，只是起到了一个遮盖作用，不能修改全局变量

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

- 如果需要在函数内改变全局变量，需要使用global关键字

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

偏函数

- 使用functools模块中的partial功能，改造现有函数，将其中的某些参数固定，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
>>> from functools import partial
>>> myadd = partial(add, 10, 20, 30, 40)
>>> myadd(5)
105
>>> myadd(2)
102
>>> myadd(8)
108

>>> int('11001100', base=2)  # 字符串是2进制数
204
>>> int('10001100', base=2)
140
>>> int2 = partial(int, base=2)  # 改造int函数，将base=2固定下来，生成新函数，名为 int2
>>> int2('11000000')
192
```

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

```
# 数字阶乘是非常直观的递归
5!=5x4x3x2x1
5!=5x4!
5!=5x4x3!
5!=5x4x3x2!
5!=5x4x3x2x1!
1!=1
```

生成器

- 使用生成器表达式，与列表解析有着一样的语法

```
>>> ['192.168.1.' + str(i) for i in range(1, 255)]
>>> ips = ('192.168.1.' + str(i) for i in range(1, 255))
>>> for ip in ips:
...     print(ip)
```

- 使用生成器函数。与普通函数不一样，生成器函数可以用yield关键字返回很多中间值

- 生成器函数的代码不是一次全部执行完，遇到yield将会产生中间值，并停在那里，下一次向生成器函数取值时，它将继续向下运行。

```
>>> def mygen():
...   yield 100
...   a = 10 + 5
...   yield a
...   b = 100 + 200
...   yield b
>>> mg = mygen()   # 创建生成器对象
>>> for i in mg:
...   print(i)
```

模块

- 模块导入时，将会到sys.path定义的路径下查找模块

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

- 我们自己编写的模块目录，可以使用PYTHONPATH环境变量定义

```
[root@localhost day02]# export PYTHONPATH=/var/ftp/nsd2019/nsd1909/py02/day01
>>> import sys
>>> sys.path
['', '/var/ftp/nsd2019/nsd1909/py02/day01', '/usr/local/lib/python36.zip',
'/usr/local/lib/python3.6', '/usr/local/lib/python3.6/lib-dynload',
'/usr/local/lib/python3.6/site-packages']
```

- 目录也可以当作一个特殊的模块，术语叫作包

```
[root@localhost day02]# mkdir aaa
[root@localhost day02]# echo 'hi = "hello world"' > aaa/hello.py
[root@localhost day02]# cat aaa/hello.py
hi = "hello world"
>>> import aaa.hello
>>> aaa.hello.hi
'hello world'
```