Lesson 10

Python Senior, Lesson 10, v1.0.0, 2016.11 by David.Yi

复习

- 图片处模块 PIL
- python 中的函数式编程之一
  - 将函数作为值返回
  - 偏函数

本次内容要点

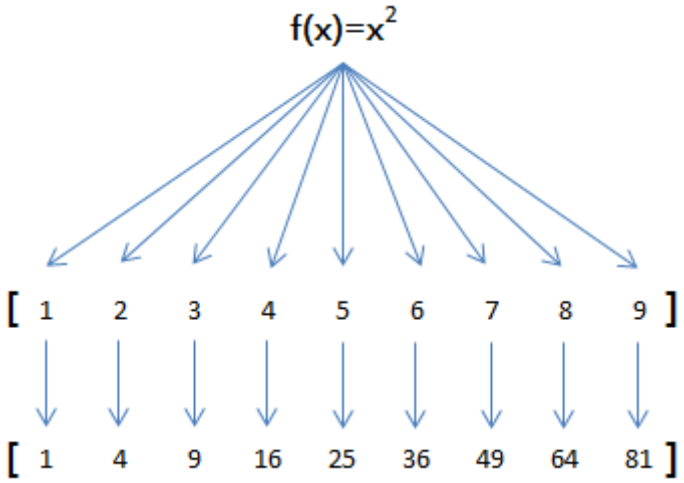
- python 中的函数式编程之二
  - 高阶函数 map/reduce filter sorted
  - 匿名函数
- 面向对象的编程之一
  - 类与实例
  - 类的属性
  - 类的方法
- 思考一下

***map()** 函数*

Python内建了 map() 和 reduce() 这两个功能强大的函数。

我们先看 map。map() 函数接收两个参数，一个是函数，一个是可迭代的序列，比如 list，map 将传入的函数依次作用到序列的每个元素，并把结果作为新的 Iterator 可迭代值返回。

举例说明，比如我们有一个函数 f(x) = x2，要把这个函数作用在一个list [1, 2, 3, 4, 5, 6, 7, 8, 9]上，就可以用map()实现如下：



> 如果你读过Google的那篇大名鼎鼎的论文“MapReduce: Simplified Data Processing on Large Clusters”，你就能大概明白map/reduce的概念。在 hadoop 时代，map/reduce 的理念是高效并行运算的核心。

In [3]:

```
# map 函数举例

def f(x):
    return x * x

r = map(f, [1, 2, 3, 4, 5, 6, 7, 8, 9])

for i in r:
    print(i)
```

```
1
4
9
16
25
36
49
64
81
```

In [2]:

```
# 这个 f(x) 函数可以更加复杂

def f(x):
    y = x * x + 3
    return y

r = map(f, [1, 2, 3, 4, 5, 6, 7, 8, 9])

for i in r:
    print(i)
```

```
4
7
12
19
28
39
52
67
84
```

In [6]:

```
# 进行 map 处理的数据也可以很复杂

def f(x):
    y = x * x + 3
    return y

l = [x for x in range(1,100,7) if x % 2 ==0]

print(l)

# 主要的程序还是很简洁
r = map(f, l)

for i in r:
    print(i)
```

```
[8, 22, 36, 50, 64, 78, 92]
67
487
1299
2503
4099
6087
8467
```

***reduce()** 函数*

再看 reduce 的用法。reduce 把一个函数作用在一个序列[x1, x2, x3, ...]上，这个函数必须接收两个参数，reduce 把结果继续和序列的下一个元素做累积计算，其效果就是：

reduce(f, [x1, x2, x3, x4]) = f(f(f(x1, x2), x3), x4)

In [8]:

```
# reduce 举例，一个加法函数
from functools import reduce

def add(x, y):
    return x + y

print(reduce(add, [1, 3, 5, 7, 9]))
```

```
25
```

In [22]:

```
# reduce，模拟一个字符串转换为整数的函数

from functools import reduce

def f(x, y):
    return x * 10 + y

# 将字符转换为数字
def char2int(s):
    return {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8':
 8, '9': 9}[s]

print(reduce(f, map(char2int, '13579')))
```

```
13579
```

In [18]:

```
# 先 map

l = map(char2num, '13579')
for i in l:
    print(i,type(i))
```

```
1 <class 'int'>
3 <class 'int'>
5 <class 'int'>
7 <class 'int'>
9 <class 'int'>
```

In [20]:

```
# 再 reduce

l = map(char2num, '13579')
print(reduce(fn,l))
```

Out[20]:

```
13579
```

In [1]:

```
# 将字符串转换成整数的函数再包装一下

from functools import reduce

def f(x, y):
    return x * 10 + y

def char2int(s):
    return {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5,
            '6': 6, '7': 7, '8': 8, '9': 9}[s]

def str2int(s):
    return reduce(f, map(char2int, s))

print(str2int('12345'))
```

```
12345
```

***filter()** 函数*

Python内建的filter()函数用于过滤序列。

和map()类似，filter()也接收一个函数和一个序列。和map()不同的是，filter()把传入的函数依次作用于每个元素，然后根据返回值是True还是False决定保留还是丢弃该元素。

In [6]:

```
# filter 举例，在一个list中，删掉偶数，只保留奇数

# 判断是否是奇数
def is_odd(n):
    return n % 2 == 1

print(list(filter(is_odd, [1, 2, 4, 5, 6])))
```

```
[1, 5]
```

In [19]:

```
# 筛选一个 list 中为空的元素

def not_empty(s):
    # strip() 用于移除字符串头尾指定的字符（默认为空格）
    if len(s.strip()) ==0:
        return False
    else:
        return True

print(list(filter(is_empty, ['A', '', 'B','C', '  '])))
```

```
['A', 'B', 'C']
```

In [21]:

```
# 返回一定范围内既不能被2整除也不能被3整数的数字

def f(x):
    return x % 2 != 0 and x % 3 != 0

print(list(filter(f, range(2, 30))))
```

```
[5, 7, 11, 13, 17, 19, 23, 25, 29]
```

高阶函数小结

使用高阶函数可以让代码简洁优雅，好的函数，可以使程序修改和调试都变得容易。

函数式编程，相对传统方式比较难理解，但是这样的确比较 pythonic，符合 python 的发展。

对于一些独立的功能、常用的功能，并且有很明显的输入和输出，放到独立函数中，是比较好的做法。

一般来说，一个函数的代码不要超过20行。

复杂的项目，一般要采用面向对象的开发方式，才能有利于维护和更新。

---

python 函数式编程的基本功能点：

函数作为返回值 偏函数 高阶函数 map/reduce, filter, sorted 匿名函数 lambda

面向对象编程

类与实例

类，是一个定义；实例是真正的对象的实现，创建一个实例的过程称作实例化。

所有的类都继承自一个叫做 object 的类。继承的定义之后再说。

类的一些操作方式和函数很像，在没有面向对象编程方式的时候，就是面向过程（函数）的开发方式。

类可以很复杂，也可以很简单，取决于应用的需要。面向对象的编程方式，是现代流行的开发方式，博大精深，需要慢慢理解体会。一开始有些不太清楚，也没有关系。

类的属性

类可以理解为一种称之为命名空间（ namespace），在这个类之下，数据都是属于这个类的实例的，我们称为属性，用实例名字+点+属性名字。这样的类比较简单，在 c 语言中称为结构体，在 pascal 中为记录类型，python 的数据结构比较简单。

In [3]:

```
# 申明一个 class MyData
class MyData(object):
    pass

# 实例化  MyData, 实例的名字叫做 obj_math
obj_math = MyData()
obj_math.x = 4
print(obj_math.x)
```

```
4
5
```

类的方法

光是把类作为命名空间太简单了，可以给类添加功能，称为方法。

方法定义在类中，使用在实例中。可以理解为实例是真正做事情的代码，所以在实例中调用方法完成某个功能是合理的。

In [6]:

```
class MyData(object):

    # 定义一个 SayHello 的方法，self 可以理解为必须传递的参数
    def SayHello(self):
        print('Hello!')

# 实例化
obj_math = MyData()

# 调用方法
obj_math.SayHello()
```

```
Hello!
```

In [9]:

```
# 类直接调用没有意义，报错
MyData.SayHello()
```

```
---------------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-9-c6a1e4d0136a> in <module>()
----> 1 MyData.SayHello(self)

NameError: name 'self' is not defined
```

In [5]:

```
# 在上面基础上，复杂一点的例子

class MyData(object):

    # 初始化方法，双下划线前后
    # 实例化的时候，需要传递 self 之后的参数
    def __init__(self, x, y):
        self.x = x
        self.y = y

    # 定义一个 SayHello 的方法，self 可以理解为必须传递的参数
    def SayHello(self):
        print('Hello!')

    def Add(self):
        print(self.x + self.y)

# 实例化
obj_math = MyData(3,4)

# 调用方法
obj_math.SayHello()

obj_math.x = 5

obj_math.Add()

o1 = MyData(1,3)
o1.Add()
```

```
Hello!
9
4
```

In [15]:

```
# 再复杂一点，创建多个实例

class MyData(object):

    # 初始化方法，双下划线前后
    # 实例化的时候，需要传递 self 之后的参数
    def __init__(self, x, y):
        self.x = x
        self.y = y

    # 定义一个 SayHello 的方法，self 可以理解为必须传递的参数
    def SayHello(self):
        print('Hello!')

    def Add(self):
        print(self.x + self.y)

# 实例化
obj_math = MyData(3,4)

# 调用方法
obj_math.SayHello()
obj_math.Add()

# 再创建一个实例
obj_math2 = MyData(5,6)
obj_math2.Add()
```

```
Hello!
7
11
```

思考一下

启动一个最简单的服务器 python3 -m http.server

然后在浏览器中 127.0.0.1:8000 来进行访问这个服务器了

In [17]:

```
# 输入 n， 如果 n 等于 1 或者 等于 2 或者 等于 3， 显示 small，其他显示 big

n = int(input('Please input:'))

if n in [1,2,3]:
    print('small')
else:
    print('big')
```

```
Please input:3
small
```

In [18]:

```
def multiply(x):
    return (x*x)

def add(x):
    return (x+x)

funcs = [multiply, add]

for i in range(5):
    value = list(map(lambda x: x(i), funcs))
    print(value)
```

```
[0, 0]
[1, 2]
[4, 4]
[9, 6]
[16, 8]
```

In [ ]:

```

```