Python程序设计

五毫升

> Python 作为当下极具影响力的编程语言，其重要性体现在语法简洁易读，新手能快速入门，且拥有庞大的库生态，像 numpy、tensorflow 等库覆盖数据处理、机器学习等多领域，大幅提升开发效率；它具备跨平台兼容性，代码可在不同系统部署，动态类型特性也让编程更灵活。
>
> 从应用看，它深度渗透数据分析、人工智能、Web 开发、自动化运维、网络爬虫等众多行业，是数据科学家处理科研与企业数据的利器，也是 AI 领域主流框架的核心接口，还能通过 Django 等框架构建各类 Web 应用，更可用于编写脚本实现系统自动化任务，堪称连接技术与行业需求的 “万能工具”，在技术演进和产业实践中都占据着不可替代的关键地位。

一、Python基础及语法

1.为什么介绍Python?

- 掌握接受其他新语言的能力
- 熟悉其他类型语言
- 增强解决问题的能力
- **解释型语言（代码 → Python解释器 → 执行）**

2.什么是Python?

- 创始人 Guido van Rossum
- 大蟒蛇
- *Python 3.0 与之前的版本不兼容*

3.Python优点

为什么用Python?

- 简单易学
- 代码优美
- 轻量级开发工具
- 类库丰富

强类型：除了强制转换，数据类型不可改变
动态类型：运行期间数据类型检查
解释性：边翻译边执行，运行速度慢
脚本语言

框架与类使用：网页框架（Flask），数据计算（numpy），普通框架（Requests）
IDE:Pycharm、IDLE、Ulipad、Vim、Emacs

4.Python语法

> - 数据类型
> - 变量
> - 函数
> - 控制流语句
> - 类库、包

(1) 数据类型

①数值类型

- 整数
- 长整数（2.0）
- 浮点数
- 复数
- 布尔类型
  运算符同 C++
  eval用来计算表达式结果

②字符类型

- 字符串：三种引号赋值均可
  str() 与 repr()
  str() ——人可以读懂的字符串
  一般用于整数和浮点数
  repr() ——解释器识别的字符串
  一般用于对象

操作：

可以用加法和乘法
从0开始，开括号
-1代表最后一个字符
不可修改

```
s[0]
s[0:4]
s[5:]
s[6:-1]
```

函数:

- count （实例化）出现次数
- len 字符串长度
- rjust 右对齐填充
- ljust 左对齐填充
- upper/lower 大小写
- split 切分
- join 连接
- find rfind index rindex 查找（子串匹配算法）
- replace 替换
- isdigit(),islower(),isupper() 检测是否只有数字/小写/大写
- strip(),lstrip(),rstrip() 处理头尾、头、尾 去除空白字符
- title() 首字母大写
- encode,decode 转换为unicode

```
s.count('word') // 实例化（成员函数）
```

格式化方法:

- %s: string (uses function 'str')
- %r: string (uses function 'repr')
- %i: int
- %f, %e, %g: float

③列表类型:

List

```
shoplist = ['apple', 'mango', 'carrot', 'banana']
shoplist[1] -> 'mango'
shoplist*2 -> ['apple', 'mango', 'carrot', 'banana', 'apple', 'mango', 'carrot', 'banana']
shoplist+['papaya'] -> ['apple', 'mango', 'carrot', 'banana', 'papaya']

for item in shoplist:
    print (item)    //遍历
```

列表方法：

- append(value), extend
- insert(index, value)
- remove (value) 删除
- index(value) 搜索

- reverse() 倒转
- sort () 排序

```
r.append(['1', '2'])     [r, ['1', '2']]
```

对于列表、字典等类型（不包含数值类型和字符串类型）：
赋值代表引用，而不是拷贝!

```
>>> a = [1, 3, 2]
>>> b = a     //引用
>>> c = b[0:2]
>>> d = b[:]  //赋值
```

相等性比较

```
>>> a = [1, 2]
>>> b = [1, 2]
>>> a == b        # test whether values are equal
True
>>> a is b        # is用于判断两个变量引用对象是否为同一个，就是所引用的对象的内存地址是否一致
False
```

④元组类型:
tuple
和list 类似，但是不能修改
zoo = ('wolf', 'elephant', 'penguin')
可以嵌套定义：
new_zoo = ('monkey', 'dolphin', zoo)
元组没有方法
List转换为tuple：

```
l=[1,2,3]
t=tuple(l)  ->  (1,2,3)
l=list(t)   ->  [1,2,3]
```

⑤字典类型：
Dictionary
Key/value对

```
d = {  'Swaroop'  : 'swaroopch@byteofpython.info',
                'Larry' : 'larry@wall.org',
                'Matsumoto' :'matz@ruby-lang.org',
                'Spammer':'spammer@hotmail.com'}
```

查找key值 `d['Larry' ]=  'larry@wall.org'`
查找key `d.has_key('Larry') = True`
添加 `d['lucy']='lucy@gmail.com'`
删除 `del d['Larry' ]`

常用方法：

- len()
- Keys()
- Values()
- Items()
- clear()
- copy()
- deepcopy()

⑥集合类型：
Set
set类似是数学里的集合概念
可以具有任意数量的项目，并且可以具有不同的类型（整数，浮点数，元组，字符串等）
与list，tuple不同的地方是，set更加强调的是一种“*从属关系*”，
跟顺序无关，所以**有重复的元素会先排除**
set类型的创建：集合创建用花括号 { } 或者 set() 函数

方法：

- **set.add(元素)** 向set集合中添加元素
- **set1.update(set2)** 将集合set2更新到集合set1中
- **set.pop()** 随机选一个元素删除并将这个值返回。
  pop()不能有参数，否则报错。如果set是空的了,也报错
- **set.remove(obj)** 删除指定元素obj,该元素必须是set中的元素,否则就报错
- **set.discard(obj)** 删除指定元素obj，obj如果是set中的元素,就删除,如果不是,就什么也不做
- **set.clear()** 清空集合中的所有元素，得到空集合

⑦类型转换：

```
typename(x)
```

(2) 变量与语句

1. 变量：

标识符的第一个字符

- 必须是字母表中的字母（大写或小写）
- 或者一个下划线（‘ _ ’）

标识符名称的其他部分

- 可以由字母（大写或小写）
- 下划线（‘ _ ’）
- 数字（0-9）组成

标识符名称是对大小写敏感的

- 例如，myname和myName不是一个标识符。
- 有效标识符：I _my_name name_23 a1b2_c3
- 无效标识符：2things、this is spaced out my-name

2. 语句：

逻辑行与物理行

i = 5　　*逻辑行*
print (i)

i = 5； print (i) ; *物理行*

缩进

同一层次的语句必须相同的缩进
i=5
　j=6 *错误*
建议一致TAB或者SPACE

3. 控制流:

If 语句

```
if a==b:
elif a>=b:
else:
```

While语句

```
while i<2:
```

For语句

```
for i in [1,2,3,4,5]:
for c in ‘hello python':
```

Break语句

```
while true:
If s==2:
break
```

Continue语句

```
If s==2:
continue
```

断行

```
使用\
if a_complicated_expression and \
 another_complicated_expression:
    print ('this is valid syntax')
使用()
if (a_complicated_expression and
another_complicated_expression):
    print ('this is valid syntax')
```

(3) 函数

函数定义

```
def sum(a,b) :
      return a+b
```

函数调用

```
func = sum （函数也可以赋值）
r = func(5,6)
```

传参

参数个数固定：

```
def add(a,b):
   return a+b

r=add(1,5)
print (r) -> 6
```

默认参数：

```
def say(message, times = 1):
    print (message * times)

say('World', 5) -> 'WorldWorldWorldWorldWorld'
```

关键参数：

根据参数的名字进行参数传递

```
def func(a, b=5, c=10):
    print ('a is', a, 'and b is', b, 'and c is', c)

func(3, 7) -> a=3, b=7, c=10
func(25, c=24) -> a=25, b=5, c=24
func(c=50, a=100) ->  a=100, b=5, c=50
```

元组参数：

args可以代表多个参数

```
def noargs(a, *args):
        print ("a=%s, others=%s" % (a, args))

noargs("hello", 1, 2, 3, "python", "good")
输出:
            a=hello, others=(1, 2, 3, 'python', 'good')
```

字典参数：

```
def keyword_args(a, b='bla', **kwargs):
    return "a=%s, b=%s, kwargs=%s" % (a, b, str(kwargs))

keyword_args(c='call', d=12, a='gr')
输出：
                    a=gr, b=bla, kwargs={'c': 'call', 'd': 12}
```

规则：

- 默认参数必须在非默认参数之后
- 只能用一个元组参数和一个字典参数
- 元组参数必须在默认参数之后（*arg）
- 字典参数必须在最后（**arg）

Lambda函数

Lambda 定义单行最小函数
作用类似于宏定义

```
g = lambda x: x*2
g(3) = 6

(lambda x,y:x+y)(2,3) -> 5
```

局部变量
局部变量作用域 – 函数体内部

不可变对象：Number ,String , Tuple，bool
可变对象： List , Set , Dictionary是可以改变内部的元素

```
def test_local(a, r):
    print ('local original ', a, r)
    a = 12
    r[1] = 999
    print ('local changed  ', a, r)

a = -5
r = [0, 1, 2]
print ('global original', a, r)
test_local(a, r)
print ('global changed ', a, r)
```

输出结果:

```
global original -5 [0, 1, 2]
local original  -5 [0, 1, 2]
local changed   12 [0, 999, 2]
global changed  -5 [0, 999, 2]
```

全局变量

```
def func():
        global x
        print ('x is', x)
        x = 2
        print ('Changed local x to', x)
x = 50
func()
print( 'Value of x is', x )
```

Return语句

```
def maximum(x, y):
   if x > y:
      return x
   else:
      return y
print (maximum(2, 3) )
```

(4) 包

包机制

```
a.py
def sum(a,b):
    return a+b;


b.py
from a import sum
Print ("3+2=",sum(3,2))
```

DocStrings-文档字符串
使得程序更加易懂：函数的第一逻辑行字符串是该函数的文档字符串DocStrings
调用：help(printMax)，或print (printMax.**doc**)

创建自己的包

```
def sayhi():
        print ('Hi, this is mymodule speaking. ’)
version = '0.1'

from mymodule import sayhi, version
        sayhi()
        print ('Version', version )
```

二、面向对象及Python文件读写

（一）Python面向对象编程

1.Python对象（object）

- 一切皆对象
- 每个对象
  - 有一个类型 type
  - 有内部的数据表示
  - 有一系列的与该对象进行交互的函数
  - dir(x) 类的成员函数

```
>>> dir()       # 获得当前模块的属性列表
>>> dir([ ])    # 查看列表的方法
```

- 对象是类型的实例（instance）

> **面向对象编程（OOP）**

> *对象* —— 带有一些属性、功能的实体
>
> - 抽象性
>   - 提取一类对象的共同属性和行为
>   - 数据抽象与功能抽象
> - 封装性
>   - 封装：将数据和行为集成到同一个实体中
>   - 解耦（Decouple）类的使用者与设计者
>   - 类的使用者经常不需要了解实现细节
>   - 类的设计者可以修改类中方法的实现方式，而不影响使用者
> - 继承性
>   - 父类（parent class, superclass）
>   - 子类（child class，subclass）
>   - 继承父类的所有属性和方法
>   - 添加新方法和新属性
>   - 重写（override）父类的已有属性
> - 多态性
>   - 对于同样的方法
>   - 类的继承体系中不同的对象调用时，功能呈现不同的形态

2.Python类（class）

类的定义与调用

成员变量
self: 类似this 指针 ，对当前实例本身的引用
成员函数
__init__：构造函数
__del__：析构函数

对象的定义：调用类的构造函数
通过“.”访问对象的属性和方法

```
#类的定义
class Circle (object):
    ‘“A 2D circle.”’
    def __init__(self, x, y, radius=1):
        self.x = x
        self.y = y
        self.radius = radius
#对象的定义
#缺省参数 radius = 1
i1 = Circle(0, 2)
#不使用缺省参数
i2 = Circle(0, 2, 3)
#通过“.”访问属性和方法
print(i1.area())
print(i2.radius)
```

类的属性与方法

- 私有属性 __attr 加双下划线
  - 不能删除、修改
- 普通类方法只有对象（类实例）可见
  - 通过self参数隐式的传递当前类对象的实例
- 静态方法 staticmethod对类和对象可见
  - @staticmethod
  - def sm
  - 不接受一个隐式的第一个参数
- 类方法 classmethod 对类和对象可见
  - @classmethod
  - def cm(cls)
  - 接受一个隐式的第一个参数cls
  - 通过cls参数传递当前类对象

```
class p:
    z=0  #类变量
         #此处的z为类p的变量，所有由p定义对象之间，共享z变量
         #可以用类变量记录类对象的格数
    def __init__(self,x,y,z=50):
        self.x=x
        self.y=y
        self.__z=5  #私有属性
        p.z=z
    def pout(self):
        print ('self.x=', self.x)
        print ('self.y=', self.y)
        print ('self.__z=', self.__z)
        print ( 'p.z=', p.z)
        print ("\n”)
```

类的继承

- super() 函数是用于调用父类(超类)的一个方法

  ```
  super(type[, object-or-type])
  ```

  type -- 类

  object-or-type -- 类，一般是 self
- super() 是用来解决多重继承问题的
  直接用类名调用父类方法在使用单继承的时候没问题
- 但是如果使用多继承
  会涉及到查找顺序（MRO）、重复调用（钻石继承）等种种问题

```
class A:
     def add(self, x):
         y = x+1
         print(y)
class B(A):
    def add(self, x):
        super().add(x)  #不带self
b = B()
b.add(2)  # 3
```

类的多态

- 子类可以重载（override）父类的方法，从而实现多态

> 多态：对于同样的方法，类的继承体系中不同的对象调用时，功能呈现不同的形态

- 如果某方法未在子类中重载，则沿继承体系向上寻找对应的接口，调用第一个找到的方法，如flugelhorn的adjust

（二）文件读写

1.文件

- 打开文件
  - f = open(filename, 'w')
    - w 写 r 读 a 追加
- 关闭文件
  - f.close ()
- 当前位置
  - f.tell ()
- 寻址
  - f.seek (offset, position)
    - position =0 文件开头为原点
    - position =1 当前位置为原点
    - position =2 文件结尾为原点
- 读取文件
  - 读取一行 f.readline()
  - 读取多行 f.readlines() 返回一个list
  - 读取指定长度 f.read(100)
  - 读取全部 f.read()
- 写文件
  - 写入一个值 f.write (value)
  - 写一行 f.writeline(signle-line)
  - 写多行 f.writelines(multi-lines)

```
spath=“test.txt”
f=open(spath,"w")
f.write(“line 1.\n")
f.writelines(“line 2.\n \line 3 \n")
f.close()
f=open(spath,"r")
for line in f:
    print line
f.close()
```

2.异常处理

- try except

    - try:
        - ……..
    - except:
        - ……..
- try finally
    - Finally 模块里语句必须执行
    - 用于释放资源

3.其他模块

help模块

- 系统帮助
    - help(str)
    - help(list)

system模块

- 参数sys.argv
- 平台 sys.platform
- 版本 sys.version
- 输入输出 sys.stdin sys.stdout

os模块

- os.name 平台
- os.getcwd() 工作目录
- os.getenv(‘path’)、os.putenv() 环境变量
- os.listdir(‘path’) 显示目录
- os.mkdir (‘path’) 创建目录
- os.rmdir (‘path’) 删除目录
- os.curdir:返回当前目录（'.‘)
- os.chdir(dirname):改变工作目录到dirname
- os.remove(‘path’) 删除
- os.linesep 平台的行终止符

time模块

```
import time
time.time()   1281282117.892
time.localtime()
time.struct_time(tm_year=2010, tm_mon=8, tm_mday=8, tm_hour=23, tm_min=42, tm_sec=31, tm_wday=6
time.asctime()
'Mon Sep 05 18:44:29 2011'
time.strftime('%Y-%m-%d %H:%M:%S', time.gmtime())
```

math模块

```
acos(x) 求x的反余弦（结果是弧度） acos(2.0)等于0.0
asin(x) 求x的反正弦（结果是弧度） asin(0.0)等于0.0
atan(x) 求x的反正切（结果是弧度） atan(0.0)等于0.0
ceil(x) 为x取整，结果是不小于x的最小整数  ceil(9.2)等于10.0
cos(x)  求x的余弦（x是弧度）      cos(0.0)等于1.0
exp(x)  求幂函数e        exp(1.0)等于2.71828
fabs(x) 求x的绝对值      fabs(-5.1)等于5.1
floor(x)        为x取整，结果是不大于x的最大整数  floor(-9.8)等于-10.0
fmod(x,y)       求x/y的余数，结果是浮点数         fmod(9.8,4.0)等于1.8
hypot(x,y)      求直角三角的斜边长度，直边长度为x和y：Sqrt(x2-y2) hypot(3.0,4.0)等于5.0
log10(x)        求x的对数（以10为底）     log10(10.0)等于1.0
pow(x,y)        求x的y次方（xy） pow(2.7,7.0)等于128.0
sin(x)  求x的正弦（x是弧度）      sin(0.0)等于0.0
sqrt(x) 求x的平方根      sqrt(900.0)等于30.0
tan(x)  求x的正切（x是弧度）      tan(0.0)等于0.0
```

Range函数

- range([start,=0] stop[, step=1])
  - Start 起始值
  - Stop 结束值
  - Step步长
- range(0,10,2)
  - [0, 2, 4, 6, 8]
- range(5,10)
  - [5, 6, 7, 8, 9]
- range(10)
  - [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Map函数

map函数作用于给定序列的每个元素，并用一个列表来提供返回值。

```
>>> from math import *
>>> r = [0, 1, 2, 3, 4, 5, 6]
>>> map(cos, r)
[1.0, 0.54030230586813977, -0.41614683654714241, -0.98999249660044542,
 -0.65364362086361194, 0.28366218546322625, 0.96017028665036597]
```

Reduce函数

reduce函数为二元函数，作用于序列的元素
每次携带一对（先前的结果以及下一个序列的元素）
连续的将现有的结果和下一个值作用在获得的随后的结果上
最后减少我们的序列为一个单一的返回值。

```
>>> r = [0, 1, 2, 3, 4, 5, 6]
>>> def sum(x, y): return x+y
>>> reduce(sum, r)
# (((((1+2)+3)+4)+5)+6)=21
print (reduce(lambda x,y:x*y, range(1, 1001)))
# 1000的阶乘
```

Filter函数

filter函数的功能相当于过滤器

```
>>> r = [0, 1, 2, 3, 4, 5, 6]
>>> def large(x): return x>3
>>> filter(large, r)
[4, 5, 6]
```

网页处理 urllib2模块

- 下载网页

```
import urllib2
response = urllib2.urlopen('http://python.org/')
html = response.read()
```

- 基于请求应答的方法

```
    import urllib2
    req = urllib2.Request('http://www.tinoweb.cn')
        response = urllib2.urlopen(req)
    the_page = response.read()
```

通过Request打开的好处是:
我们可以很方便的为Request 添加HTTP请求的头部信息

- 发送数据

```
import urllib
import urllib2
url = 'http://dict.youdao.com/search'
user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
headers = { 'User-Agent' : user_agent }
values = {'q' : 'python'}
data = urllib.urlencode(values)
req = urllib2.Request(url, data, headers)
response = urllib2.urlopen(req)
the_page = response.read()
print the_page
```

- 异常处理

```
from urllib2 import Request, urlopen, URLError, HTTPError
req = Request(someurl)
try:
        response = urlopen(req)
except HTTPError, e:
        print  ('The server couldn\'t fulfill the request.' )
        print ('Error code: ', e.code )
except URLError, e:
        print ('We failed to reach a server.' )
        print ('Reason: ', e.reason )
else:
        print ("everything is fine ")
```

Urllib库是Python中的一个功能强大、用于操作URL，并在做爬虫的时候经常要用到的库。在Python2.x中，分为Urllib库和Urllib2库，Python3.x之后都合并到Urllib库中，使用方法稍有不同。

```
# get code of given URL as html text string
# Python3 uses urllib.request.urlopen()
# instead of Python2's urllib.urlopen() or urllib2.urlopen()
import urllib.request
fp = urllib.request.urlopen("http://www.baidu.com")
mybytes = fp.read()
# note that Python3 does not read the html code as string
# but as html code bytearray, convert to string with
mystr = mybytes.decode("utf8")
fp.close()
print(mystr)
```

HTML解析

- HTMLParser
- 自己用正则表达式（后面详细介绍）

处理中文

- python3中str默认为Unicode的编码格式，Unicode是32位编码格式，不适合用来传输和存储，需转换成utf-8，gbk等等
- 在Python3中将str类型转换成bytes类型的，使用encode的方式可以进行字符的编码

```
>>>a = "中国"
>>> a.encode("utf-8")
b'\xe4\xb8\xad\xe5\x9b\xbd'
>>>a.encode("utf-8").decode("utf-8")
'中国'
```

三、正则表达式

（一）通用正则表达式

什么是正则表达式?

在计算机科学中，指一个*用来描述或者匹配一系列符合某个句法规则的字符串*的**单个字符串**

```
010-\d{8}
010-627[789]\d{4}
```

用途：

- 模式匹配：是否满足某一特征的字符串

    - 匹配电话号码
    - 匹配URL
- 查找字符串
    - 字符串替换

语法：

表达式	作用
\d	匹配任意一个数字，0~9 中的任意一个
\w	匹配任意一个字母或数字或下划线，也就是 A~Z、a~z、0~9、_ 中任意一个
\s	匹配包括空格、制表符、换页符等空白字符的其中任意一个
.	匹配小数点以外的任意一个字符（除了换行符 \n）
\D	匹配非 \d 的字符
\W	匹配非 \w 的字符
\S	匹配非 \s 的字符

```
# 北京电话号码
010-\d\d\d\d\d\d\d\d
# 用户名长度为的6的Gmail地址
\w\w\w\w\w\w@gmail.com
```

表达式	作用
{n}	表达式重复n次，比如："\w{2}" 相当于 "\w\w"；"a{5}" 相当于 "aaaaa"
{m,n}	表达式至少重复m次，最多重复n次，比如："ba{1,3}"可以匹配 "ba"或"baa"或"baaa"
{m,}	表达式至少重复m次，比如："\w\d{2,}"可以匹配 "a12","_456","M12344"...
?	匹配表达式0次或者1次，相当于 {0,1}，比如："a[cd]?"可以匹配 "a","ac","ad"
+	表达式至少出现1次，相当于 {1,}，比如："a+b"可以匹配 "ab","aab","aaab"...
*	表达式不出现或出现任意次，相当于 {0,}，比如：“a*b"可以匹配 "b",“aaaab"...

. #匹配单个字符
r.t
匹配 rat rut
不匹配root
* #匹配0或多个字符
r.*t
匹配 rt rot root
+ #匹配1或多个字符
9+ 匹配9999
？ #匹配0个或1个字符
9？ 匹配9

表达式	作用
[ab5@]	匹配 "a" 或 "b" 或 "5" 或 "@"
[^abc]	匹配 "a","b","c" 之外的任意一个字符
[f-k]	匹配 "f"~"k" 之间的任意一个字母
[^A-F0-3]	匹配 "A"~"F","0"~"3" 之外的任意一个字符

[ ] 匹配括号中的任何一个字符
r[aou]t
匹配 rat、rot、rut
不匹配 root
[0-9] 匹配任何数字
[a-z] 匹配a-z之间的任何字符
[^a-z] 匹配除了a-z之间的字符
{i}匹配指定数目的字符
A[0-9]{3} 匹配 A123、A348

一个复杂的例子（年月日）

1.^[1-9]表示年是以数字1-9开头的,\d{0,3}表示年的位数,
^[1-9]\d{0,3}就表示1-9999年之间

^ 与字符串开始的地方匹配，不匹配任何字符

2.(1[0-2]|0?[1-9])中|前面的1[0-2]表示从10到12,
后面的0?[1-9]表示01-09或者1-9,(1[0-2]|0?[1-9])表示月,01-12或者1-12

3.(3[01]|[12]\d|0?[1-9])$其中3[01]表示30或31,
[12]\d表示从10-29,最后的0?[1-9]表示从01-09或者是从1-9.
整体就表示从01-31或者1-31

转义字符

\\ 表示\
* 表示*
\$ 表示 $
\. 表示.

贪婪与懒惰

懒惰匹配
*? 重复任意次，但尽可能少重复
+? 重复1次或更多次，但尽可能少重复
?? 重复0次或1次，但尽可能少重复
{n,m}? 重复n到m次，但尽可能少重复
{n,}? 重复n次以上，但尽可能少重复

字符边界

表达式	作用
^	与字符串开始的地方匹配，不匹配任何字符
$	与字符串结束的地方匹配，不匹配任何字符
\b	匹配一个单词边界，也就是单词和空格之间的位置，不匹配任何字符

$
are you$
匹配 how are you
不匹配how old are you?
^
^how are
匹配 how are you
不匹配how old are you
\b
\bho\b
匹配 ho are you
不匹配how old are you

逻辑与分组

- \B 和\b相反
  - \Bthe\B
    - 不匹配in the world
    - 匹配otherwise
- | 或运算
  - 0\d{2}-\d{8}|0\d{3}-\d{7}
- ( )
  - 整体被修饰，以后使用分组
    - (\d{1,3}.){3}\d{1,3} \ ip地址
    - ((2[0-4]\d|25[0-5]|[01]?\d\d?).){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)

表达式 (((ab*)c)d)e
字符串 abcdef

分组结果
group(0) = abcde 整个匹配结果
group(1) = abcd 第一组结果
group(2) = abc
group(3) = ab

THSS
如何匹配出 http://thss.tsinghua.edu.cn
<a\s*href\s*=\s*"([^"]*)"\s*title="([^"]*)"[^>]*>
group(1)
http://thss.tsinghua.edu.cn
group(2)
thss

注释

(?#comment)来包含注释
2[0-4]\d(?#200-249)|[01]?\d\d?(?#0-199)

（二）Python正则表达式的使用

> - 包re支持正则表示式
> - 字符串匹配
> - 字符串分解
> - 字符串替换

查找开头

- re.match:字符串匹配
- re.match的函数原型为：re.match(pattern, string, flags)
  - 第一个参数是正则表达式
  - 第二个参数表示要匹配的字符串；
  - 第三个参数是标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。
    - re.I 表示大小写忽略
    - re.M 多行模式 使 ^ $ 匹配除了string开始结束外，还匹配一行的开始和结束
    - re.S “.“ 匹配包括‘\n’在内的任意字符，否则 . 不包括‘\n’
    - re.X 为了写正则表达式，会忽略一些空格和#后面的注释
    - re.L 本地化 \w也匹配本地化语言 例如汉字
- re.match只匹配字符串的开始，如果字符串开始不符合正则表达式，则匹配失败，函数返回None；如果匹配成功，则返回一个Match

```
import re
text = "JGood is a handsome boy, he is cool, clever, and so on..."
m = re.match("(\w+)\s", text)
if m:
    print (m.group(), '\n')
else:
    print ('not match')
>>> JGood
```

查找一个结果

- re.search和re.match参数相同
  - re.search函数会在字符串内查找模式匹配,直到找到第一个匹配然后返回，如果字符串没有匹配，则返回None。

- re.match与re.search的区别：re.match只匹配字符串的开始，如果字符串开始不符合正则表达式，则匹配失败，函数返回None；
- 而re.search匹配整个字符串，直到找到一个匹配。

```
rst = re.match()
rst = re.search()

找到的起始位置 rst.start()
找到的结束位置rst.end()
找到的区间 rst.span ()
找到的组 rst.group ()
```

查找全部

- re.findall
  - re.findall可以获取字符串中所有匹配的字符串，返回一个列表
  - 如：re.findall('\w*oo*\w', text)；获取字符串中，包含'oo'的所有单词。
- re.finditer
  - re.finditer可以获取字符串中所有匹配的字符串，返回一个迭代器
  - 如：re.finditer('\w*oo*\w', text)；返回一个迭代器。

编译成对象

- re.compile
  - 把正则表达式编译成一个正则表达式对象。
  - compile(pattern, [flags])
    - 根据正则表达式字符串 pattern 和可选的flags 生成正则表达式 对象，其中flags有下面的定义：
      - I 表示大小写忽略
      - M 多行模式 使 ^ $ 匹配除了string开始结束外，还匹配一行的开始和结束
      - S 单行模式 “.“ 匹配包括'\n'在内的任意字符，否则 . 不包括'\n'

```
p = re.compile( '(blue|white|red)')
p.search(“I like blue and red”)
```

替换

- re.sub
  - re.sub用于替换字符串中的匹配项。
  - re.sub的函数原型为：

    ```
    re.sub(replacement, string, [count = 0])
    ```
  - 第1个参数是替换后的字符串；
  - 第2个参数是替换的字符串；

- 第3个参数是替换个数。默认为0，表示每个匹配项都替换。

切分

- re.split
  - 用来分割字符串
  - split(string , [maxsplit = 0])
  - p = re.compile('\s')
  - print (p.split('This is a test, short and sweet, of split().'))
    - ['This', 'is', 'a', 'test,', 'short', 'and', 'sweet,', 'of', 'split().']
  - print (p.split('This is a test, short and sweet, of split().', 3))
    - ['This', 'is', 'a', 'test, short and sweet, of split().']