第5章 基础(标准)库

李宇

东北大学-计算机学院-智慧系统实验室

liyu@cse.neu.edu.cn

标准库(Standard Library)

- 标准库：标准安装Python时包含的模块
- 模块就是程序

```
#在hello.py内，一个简单的模块
print("Hello, world!")
```

假设这个文件存储在'C:/python'

- 告诉解释器去哪里查找这个模块

```
>>> import sys
>>> sys.path.append('C:/python')
```

sys.path包含一个目录，解释器将在目录中查找模块

- 然后就可以导入模块了

```
>>> import hello
Hello, world!
```

导入这个模块时，执行了其中的代码。但如果再次导入它，什么事情都不会发生

模块中定义函数

- 只包含一个函数的模块

```
# hello2.py
def hello():
        print("Hello, world!")
```

- 导入

```
>>> import hello2
```

- 访问模块的方法

```
>>> hello2.hello()
Hello, world!
```

- 模块是为了重用代码，如果要让代码是可重用的，务必将其模块化

包(package)

- 可以将模块编组为包
  - 包是一个目录，模块是一个.py文件，包里有模块
  - 目录里必须包含文件__init__.py（**变文件夹为python包**）
  - 例，把多个模块封装成有层级的包：

包名 graphics/

```
        __init__.py
        primitive/
                __init__.py
                line.py
                fill.py
                text.py
```

包名 formats/

```
                __init__.py
                png.py
                jpg.py
```

```
#有了封装好的包之后，可以使用import语句了
import graphics.primitive.line
from graphics.primitive import line
import graphics.formats.jpg as jpg
```

偷懒导入法

- from xx import *
  - 有时候我们在做导入时会偷懒，将包中的所有内容导入
- 这也是需要实现的，__all__变量就是做这个工作的
- __all__ 关联了一个模块列表，当执行 from xx import * 时，就会导入列表中的模块
- 将primitive路径下的__init__.py文件修改为：

```
__all__ = [‘text’, ‘line‘]
>>> from graphics.primitive import *
>>> dir()
#结果包含了line和text模块，并没有fill模块，因为fill不在__all__变量里
```

模块里包含什么

- 用内置函数dir()查看
  - 列出对象的所有属性
  - 对于模块，列出所有的函数、类、变量等等

```
import copy
dir(copy)
```

```
Out[6]: ['Error',
         '__all__',
         '__builtins__',
         '__cached__',
         '__doc__',
         '__file__',
         '__loader__',
         '__name__',
         '__package__',
         '__spec__',
         '_copy_dispatch',
         '_copy_immutable',
         '_deepcopy_atomic',
         '_deepcopy_dict',
         '_deepcopy_dispatch',
         '_deepcopy_list',
         '_deepcopy_method',
         '_deepcopy_tuple',
         '_keep_alive',
         '_reconstruct',
         'copy',
         'deepcopy',
         'dispatch_table',
         'error']
```

以单下划线打头的名称不供外部使用（保护变量）

__all__变量注意事项

- __all__变量包含一个列表
  - 在使用from xx模块 import * 时，只能导入__all__变量中包含的函数

```
>>> copy.__all__
#['Error', 'copy', 'deepcopy']
```

  - 此时dipatch_table模块并没有被导入
  - 如果要导入dispach_table，必须显式地导入copy并使用

```
copy.dispatch_table
```

或

```
from copy import dispatch_table
```

程序入口

- 对于C，C++，以及完全面向对象的编程语言 Java，C# 程序都必须要有一个入口

- Python 不同，它属于脚本语言，不像编译型语言那样先将程序编译成二进制再运行，而是动态的逐行解释运行。也就是从脚本第一行开始运行，没有统一的入口。

- Python 源码文件可被直接运行，还可以作为模块被导入。不管是导入还是直接运行，最顶层的代码都会被运行（Python 用缩进来区分代码层次）。而实际上在导入的时候，有一部分代码我们是不希望被运行的。

如果 Python 没有程序入口

- 假设我们有一个 const.py 文件

```
#const.py
PI = 3.14
def main():
        print ("PI:", PI)
main()
#输出：PI: 3.14
```

定义了一些常量，然后又写了一个 main 函数来输出定义的常量

- 编写area.py 文件，用于计算圆的面积

```
from const import PI
def calc_round_area(radius):
        return PI * (radius ** 2)
def main():
        print "round area: ",
calc_round_area(2)
main()
#PI: 3.14
#round area: 12.56
```

用到 const.py 文件中的 PI 变量，那么我们从 const.py 中把 PI 变量导入到 area.py 中

const 中的 main 函数也被运行，但我们只想引入const内容，不希望它内部的main被运行

if __name__ == '__main__':

- 把 const.py 改一下:

```
#const.py
PI = 3.14
def main():
        print ("PI:", PI)

if __name__ == "__main__":
        main()
```

然后再运行 area.py，输出:
round area: 12.56
(PI: 3.14 消失了)

此代码块此时不会被运行

- if __name__ == '__main__' 就相当于是 Python **模拟的程序入口**
- **只有运行const.py时，入口下的代码块才会执行**
- 由于模块之间相互引用，不同模块可能都有这样的定义，而入口程序只能有一个。到底哪个入口程序被选中，这取决于 __name__ 的值

__name__变量

- __name__是内置变量，用于表示当前**模块的名字**。反映了模块在包中的**层次**。其实，所谓模块名就是 import 时需要用到的名字

```
import tornado
import tornado.web
```

tornado 和 tornado.web 就被称为这两个模块的模块名

- if __name__ == '__main__' 的含义：
  - 如果一个模块被直接运行（点击run按钮），则其没有包结构，其 __name__ 值为 __main__
  - 如果模块是被直接运行的，则该代码块被运行，如果模块是被导入的，其__name__为模块名，不为__main__，该代码块不被运行。

模块的其他特殊变量

- __file__：放置模块的路径

```
>>> print(copy.__file__)
C:\Python35\lib\copy.py
```

- __doc__：文档字符串

帮助信息是从函数copy的文档字符串中提取的

```
>>> help(copy.copy)
Help on function copy in module copy:
copy(x)
Shallow copy operation on arbitrary Python objects.
See the module's __doc__ string for more info.
```

sys模块

- 可以使用sys访问与Python**解释器相关**的变量和函数

表10-2　模块sys中一些重要的函数和变量

函数/变量	描　　述
argv	命令行参数，包括脚本名
exit([arg])	退出当前程序，可通过可选参数指定返回值或错误消息
modules	一个字典，将模块名映射到加载的模块
path	一个列表，包含要在其中查找模块的目录的名称
platform	一个平台标识符，如sunos5或win32
stdin	标准输入流——一个类似于文件的对象
stdout	标准输出流——一个类似于文件的对象
stderr	标准错误流——一个类似于文件的对象

fileinput模块

- 可以对一个或多个文件中的内容进行迭代、遍历等操作。

表10-4　模块fileinput中一些重要的函数

函　　数	描　　述
input([files[, inplace[, backup]]])	帮助迭代多个输入流中的行
filename()	返回当前文件的名称
lineno()	返回（累计的）当前行号
filelineno()	返回在当前文件中的行号
isfirstline()	检查当前行是否是文件中的第一行
isstdin()	检查最后一行是否来自sys.stdin
nextfile()	关闭当前文件并移到下一个文件
close()	关闭序列

例子，给脚本添加行号

```
# numberlines.py

import fileinput

for line in fileinput.input(inplace=True):
    line = line.rstrip()
    num = fileinput.lineno()
    print('{:<50} # {:2d}'.format(line, num))
```

input可返回可迭代对象，inplace=True
表示将标准输出(print)的结果写回文件

rstrip方法去掉字符串末尾的空白

<(左对齐)，50(一行50个字符)
2d(长度为2的十进制数)

$ python numberlines.py numberlines.py

```
# numberlines.py                                  # 1
                                                  # 2
import fileinput                                  # 3
                                                  # 4
for line in fileinput.input(inplace=True):        # 5
    line = line.rstrip()                          # 6
    num = fileinput.lineno()                      # 7
    print('{:<50} # {:2d}'.format(line, num))     # 8
```

读取一个文件的所有行

- 可以传入多个文件进行批量迭代，需把文件名放在列表中

```
import fileinput

for line in fileinput.input(”numberlines.py"):
    print(line)
```

输出：

```
import fileinput


for line in fileinput.input(inplace=True):
        line = line.rstrip()
        num = fileinput.lineno()
        print('{:<50} # {:2d}'.format(line,num))
```

random模块

- 包含生成伪随机数的方法
  - 用于编写例子程序或者模拟程序的输入
  - random生成的随机数可预测!
  - os模块中的urandom方法更接近真正随机数(也是伪随机)

表10-8　模块random中一些重要的函数

函　　数	描　　述
random()	返回一个0~1（含）的随机实数
getrandbits(n)	以长整数方式返回n个随机的二进制位
uniform(a, b)	返回一个a~b（含）的随机实数
randrange([start], stop, [step])	从range(start, stop, step)中随机地选择一个数
choice(seq)	从序列seq中随机地选择一个元素
shuffle(seq[, random])	就地打乱序列seq
sample(seq, n)	从序列seq中随机地选择n个值不同的元素

random模块的例子

```
import random
print( random.randint(1,10) )
# 产生 1 到 10 的一个整数型随机数 [1,10]
print( random.random() )
# 产生 0 到 1 之间的随机浮点数[0, 1)
print( random.uniform(1.1,5.4) )
# 产生 1.1 到 5.4 之间的随机浮点数，区间[]可以不是整数
print( random.choice('tomorrow') )
# 从序列中随机选取一个元素（此例中为字符）
print( random.randrange(1,100,2) )
# 生成一个从1到100的间隔为2的随机整数

a=[1,3,5,6,7] # 将序列a中的元素顺序打乱
random.shuffle(a)
print(a)
```

数据持久化

- 持久化（Persistence）即把数据（如内存中的对象）以各种方式(数据库、文件)保存到持久化的设备(比如硬盘)。 *---Data continues to exist even after the application has ended*

- 持久化的内置标准库
  - build-in File API(write函数)
  - Shelve模块，Pickle模块，cPickle模块，JSON模块, XML…
- 第三方库
  - UltraJSON模块

shelve模块

- shelve模块
  - 将数据存储到**文件**中（默认为二进制）
  - 存储字典类型的数据到本地磁盘
  - shelve.open()方法
    - 传入一个文件名，返回一个Shelve对象，用来存储数据
    - 可以**把它当作字典**来操作（键必须是字符串）
    - 操作完毕时，调用close方法保存修改

```
>>> import shelve
>>> s = shelve.open('test1')
>>> s['x'] = ['a', 'b', 'c']
>>> s['x'].append('d')
>>> s['x']
['a', 'b', 'c']
```

'd' 消失了?

赋值时才会保存

获取存储的表示，会创建一个新列表，再将'd'附加到这个新列表末尾，但这个修改后的版本未被存储！因为没有使用=赋值

shelve模块如何保存？

```
>>> import shelve
>>> s = shelve.open('test1')
>>> s['x'] = ['a', 'b', 'c']
>>> s['x'].append('d')
>>> s['x']
['a', 'b', 'c']
```

在 Windows 上运行此代码，在当前工作目录产生有3个新文件：test1.bak 、 test1.dat 和 test1.dir 。在MacOS 上，只会创建一个 test.db 文件

- 要正确地修改使用模块shelve存储的对象，必须将获取的副本赋给一个临时变量，并在修改这个副本后再次存储

```
>>> temp = s['x']
>>> temp.append('d')
>>> s['x'] = temp
>>> s['x']
['a', 'b', 'c', 'd']
```

修改s['x']并保存

一个简单的数据库

- 使用shelve模块创建一个简单的**数据库**应用程序：
  - 创建关于Person的数据库，包含ID，name，age，phone
  - 用shelve.open打开database文件，返回一个shelve对象
  - 让用户输入数据，**保存**到shelve对象中
  - 提供**查询**数据库的功能
  - 提供**help**功能，提高用户使用体验
  - 提供**退出**功能

数据库例子

re模块(Regular Expression)

- re模块：正则表达式模块
  - Regular Expression：缩写为regex、regexp或RE
  - 在Python中使用正则时，必须“**import re**”
  - 正则是可匹配文本片段的模式(模式 ->字符串的匹配)
  - 最简单的正则表达式为普通的字符串，与自己匹配
    - 正则表达式'python'与字符串'python'匹配
    - 正则的功能：
      - 文本中查找模式(pattern)
      - 将特定的模式替换为计算得到的值
      - 将文本分割成片段

正则表达式使用场景

- 主要用于文本处理时：
  - 验证用户输入的邮箱、手机号、身份证号的格式，避免恶意胡乱输入
    - xxx@cse.neu.edu.cn，xxxx@qq.com
    - 1319425xxxx(11位)
  - 过滤文本中的特定内容（英文、数字或符号、网址、日期）
    - str.find、str.index难以处理此类工作
  - 批量替换文本内容
    - 找到文本中$xxx格式的内容、替换成￥xxx

在字符串
“我叫王宇，今年28岁，身高182cm，体重90kg，以前的手机号是13912345678，现在的手机号是：18812345678，家里的电话是66658933，身份证号是 340403198501011234”
中找出所有的数字

正则表达式使用场景

- 若用字符串的方法，逐个字符分析，代码如下：

```
def findnumbers(s):
    p = 0
    res = []
    for i in range(0, len(s)):
        if (p == 0 or not s[i-1].isdigit()) and s[i].isdigit():
            p = i
        elif s[i].isdigit() and (i + 1 == len(s) or not s[i+1]\
        .isdigit()):
            res.append(s[p:i+1])
    return res

content = "我叫王宇，今年28岁......"
findnumbers(content)

#['28', '182', '90', '13912345678', '18812345678', '66658933',
'340403198501011234']
```

正则表达式使用场景

- 若用正则的方法，代码如下：

```
import re
content='我叫张三今年28岁，身高182cm...'
#编译pattern，此pattern为一个或多个0-9的数字
regex=re.compile('[0-9]+')
#此时regex为pattern类型的数据结构
#在content字符串中寻找此pattern
res = regex.findall(content)
print('all numbers:', res)

#all numbers: ['28', '182', '90', '13912345678', '18812345678',
'66658933', '340403198501011234']
```

re模块(Regular Expression)

- 在提供了正则表达式的编程语言里，正则的语法几乎是一样的。
- 早期版本、旧时代中采用compile来编译正则表达式：

```
import re
# 用re.compile将正则表达式编译成Pattern对象
pattern = re.compile(r'hello')

# 使用Pattern匹配文本，获得匹配结果，无法匹配时将返回None
match = pattern.match('hello world!')

if match:
    # 使用Match获得分组信息
    print(match.group())

### 输出 ###
# hello
```

re模块(Regular Expression)

```
#旧时代（Python仍然支持这么写）：
import re
pattern = re.compile(r'hello')
match = pattern.match('hello world!')

if match:
    print(match.group())

# hello
```

```
#新时代写法：不需要re.compile了
re.match、re.search等方法会把第一个参数编译成正则，这些方法都会自动调用re.compile
import re
match = re.match(r'hello','hello world!')
if match:
    print(match.group())

# hello
```

re模块(Regular Expression)

- 无需多虑执行次数带来的开销

```
#旧时代:
texts = [包含一百万个字符串的列表]
#此时只compile一次，生成了pattern
pattern = re.compile('正则表达式')
for text in texts:
    pattern.search(text)
```

```
#新时代写法：不需要re.compile了

texts = [包含100万个字符串的列表]
for text in texts:
    re.search('正则表达式', text)
#相当于执行了100万次re.compile?
#多虑了，compile内部实现自带缓存，不多次执行
```

正则中的特殊字符

- 通配符 (‘.’)
  - 特殊字符
  - 句点(‘.’)与**除换行符(\n)**外的任何字符都匹配，因此被称为通配符(wildcard)

正则表达式:

```
'.ython'
```

匹配下列哪些字符串?

```
'python'
'jython'
'qython'
'+ython'
' ython'
'cpython'   X
'ython'     X
```

对特殊字符进行转义

- 普通字符只与自己匹配，特殊字符则完全不同，例如通配符（‘.’）
  - 假设要匹配’python.org’这个字符串，可以使用’python.org’这个模式吗?

    ‘python.org’　‘pythonxorg’　‘python1org’　‘pythonzorg’

    这些字符串都匹配
  - 对特殊字符进行转义：让特殊字符的行为与普通字符一样
  - 转义方法：前面加上两个反斜杠(\\)
    - 匹配字符串‘python.org’时，使用模式：**'python\\.org’**
      - 使用双反斜杠后，’.’已经不再是通配符
      - 此模式将只跟字符串’python.org’进行匹配
      - 也可使用一个反斜杠和**原始字符串**如 **r'python\.org'**

字符集

- 为了更精细的控制，使用通配符是不够的
- 使用方括号[ ]，创建一个字符集，与其包含的字符都匹配
  - 现有模式‘[pj]ython‘，与之匹配的字符串有：
    - 'python'
    - 'jython’
  - 范围字符集
    - 模式‘[a-z]‘ 与a~z的任何字母都匹配
  - 组合范围字符集，依次列出它们
    - '[a-zA-Z0-9]'与大写字母、小写字母和数字都匹配
      - 中间没有空格
    - [一-十]匹配中文数字、[^0]非数字0的字符

预定义字符集

\d： 匹配数字，等价于[0-9]

\D： 匹配任意**非**数字字符, 相当于[^0-9]

\s： 匹配空字符，如\t（tab）、\r\n（回车）、‘ ‘（空格）

\S： 匹配任意**非**空字符，相当于[^\t\n\r\r\v]

\w： 匹配任意英文字符或数字，等价于[0-9a-zA-Z]

\W：同理，和上面相反，相当于[^a-zA-Z0-9]

二选一和子模式

- 如果只想匹配字符串‘python’和‘perl‘，怎么办？使用字符集或者通配符来生成的模式可以做到吗？

- 管道字符(|): 表示二选一
  - 上述所需的模式为'python|perl’
- 子模式：使用圆括号（）
  - 不想将二选一运算符用于整个模式，只想用于模式的一部分
  - 于前面的示例，可重写为'p(ython|erl)’
  - 单个字符也可做为子模式，不用()括起来。

可选模式

- 可选模式：问号（?）
  - 在子模式后面加问号，将其指定为可选的，即**可包含可不包含。**
  - 例如有模式：r'(http://)?(www\.)?python\.org’
    - 对通配符进行了转义，防止它充当通配符
    - 为减少所需的反斜杠数量，使用了原始字符串。
    - 每个可选的子模式都放在圆括号内。
    - 每个可选的子模式都可以出现，也可以不出现。

上述模式与下列哪些字符串匹配?

```
'http://www.python.org'
'http://python.org'
'www.python.org'
'python.org'
```

都匹配

重复模式

- (?) 表示可选子模式可以出现一次，也可以不出现
- *号，+号，花括号{ } 用于表示子模式可重复多次
  - (pattern)*：pattern可重复0、1或多次。
  - (pattern)+：pattern可重复1或多次。
  - (pattern){m,n}：模式可重复m~n次, 也可以把n去掉。

模式 **r'w*\.python\.org'**，
与以下哪些字符串匹配?

```
'www.python.org'
'.python.org'
'ww.python.org'
'wwwwwww.python.org'
```

模式 **r'w+\.python\.org'**，
与以下哪些字符串匹配?

```
'www.python.org'
'.python.org'
'ww.python.org'
'wwwwwww.python.org'
```

模式 **r'w{3,4}\.python\.org'**
与以下哪些字符串匹配?

```
'www.python.org'
'.python.org'
'ww.python.org'
'wwwwwww.python.org'
```

字符串的开头和末尾

- 脱字符 (‘^’): 匹配字符串的开头
- 美元符号(‘$’): 匹配字符串的末尾

模式 **'^ht+p'**,
与以下哪些字符串匹配?

```
'http://python.org'
'htttttp://python.org'
'www.http.org'
```

模式 **‘world$’**,
与以下哪些字符串匹配?

```
'hello world'
'hello world Mr.Li'
'www.http.org.wworld'
```

练习：常用正则式

- 匹配合法的身份证号：
  - 18位身份证号，例如：410001199101011 23X
  - 草率写法：
    - "\d{18}|\d{17}[0-9Xx]“
  - 严肃写法：

第1位：1-9中的一个， 4 [1-9]
第2-6位： 5位数字， 10001（前6位表示省市县地区） \d{5}
第7-8位： 2位数字， 19，表示世纪（现可能取值范围18xx-20xx年）(18|19|20)
第9-10位： 2位数字， 91（年份）\d{2}
第11-12位：01-12， 01（月份）((0[1-9])|(10|11|12))
第13-14位：01-31， 01（日期）(([0-2][1-9])|10|20|30|31)
第15-17位：3位数字， 123（第17位奇数代表男，偶数代表女） \d{3}
第18位： 0123456789Xx其中的一个， X（第18位为校验值） [0-9Xx]

p = "[1-9]\d{5}(18|19|20)\d{2}((0[1-9])|(10|11|12))(([0-2][1-9])|10|20|30|31)\d{3}[0-9Xx]"

练习：常用正则式

- 过滤URL
  - 可以使用第三方库
  - 使用正则可以实现
  - 题目：
    - 匹配http://，“https://”开头的url，
    - 后面可以包含 “.”、数字、字符、”-,?,:”
  - p1 = "http[s]?://(?:[-\w\.])+“
  - （写法不唯一，此写法并不完备）

re模块的内容

东北大学 Northeastern University

表10-9　模块re中一些重要的函数

函　　数	描　　述
compile(pattern[, flags])	根据包含正则表达式的字符串创建模式对象
search(pattern, string[, flags])	在字符串中查找模式
match(pattern, string[, flags])	在字符串开头匹配模式
split(pattern, string[, maxsplit=0])	根据模式来分割字符串
findall(pattern, string)	返回一个列表，其中包含字符串中所有与模式匹配的子串
sub(pat, repl, string[, count=0])	将字符串中与模式pat匹配的子串都替换为repl
escape(string)	对字符串中所有的正则表达式特殊字符都进行转义

re.search()

- re.search：在给定字符串中查找**第一个**与指定正则表达式匹配的子串，找到将返回MatchObject(结果为真)，否则返回None(假)
  - 例：在文件中查找包含“From:”

使用str.find方法

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.strip()
    if line.find('From:') >= 0:
        print(line)
```

使用re.search

```
import re

hand = open('mbox-short.txt')
for line in hand:
    line = line.strip()
    if re.search('From:', line) :
        print(line)
```

要找的模式　从哪找

re.search() （二）

- 使用re.search() 来替代startswith()
  - 例：查找以“From:”开头的字符串

使用字符串的startswith方法

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.startswith('From:') :
        print(line)
```

使用re.search

```
import re

hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line) :
        print(line)
```

要找的模式　从哪找

re.match()和re.split()

- re.match方法: 在给定字符串开头查找与正则表达式匹配的子串，返回真（MatchObject）或假（None）
  - 用法：match(pattern, string, flags=0)
    - re.match('p','python’)　# 返回一个MatchObject
    - re.match(‘p’, ‘www.python.org‘)　# None

- re.split方法：根据与模式匹配的子串来分割字符串
  - 类似于字符串方法split
  - 多数情况都使用逗号或者空格分割字符串

```
>>> some_text = 'alpha, beta,,,,gamma delta'
>>> re.split('[, ]+', some_text)
#['alpha', 'beta', 'gamma', 'delta']
```

`[, ]+`表示一个或者多个逗号和空格

re.findall()

- re.findall返回一个列表，其中包含所有与给定模式匹配的子串
  - 用法：findall(pattern, string, flags=0)
  - 例：找出字符串中所有单词

```
>>> pat ='[a-zA-Z]+' ?
>>> text = '"Hm... Err -- are you sure?" he said, sounding insecure.'
>>> re.findall(pat, text)
['Hm', 'Err', 'are', 'you', 'sure', 'he', 'said', 'sounding', 'insecure']
```

  - 例： 找出所有的标点符号

```
>>> pat = r'[.?\-",]+'   ?
>>> re.findall(pat, text)
['"', '...', '--', '?"', ',', '.']
```

这里对连字符（-）进行了转义，因此Python不会认为它是用来指定字符范围的（如a-z）。

其他re的方法

- re.sub()：左往右将与模式匹配的子串替换为指定内容

```
>>> pat = '{name}'
>>> text = 'Dear {name}...'
>>> re.sub(pat, 'Mr. Gumby', text)
输出: 'Dear Mr. Gumby...'
```

替换成的内容

- re.escape：帮你转义，不用大量的输入 \\ 了。
  - 对字符串中所有可能被视为正则表达式运算符的字符进行转义

```
>>> re.escape('www.python.org')
输出: 'www\\.python\\.org'
>>> re.escape('But where is the ambiguity?')
输出: 'But\\ where\\ is\\ the\\ ambiguity\\?'
```

编组

- 编组(group)：就是放在圆括号内的子模式，编组0指的是整个模式
  - 有如下模式：

```
'There (was a (wee) (cooper)) who (lived in Fyfe)'
```

  - 该模式包含如下**编组**(group):

0 There was a wee cooper who lived in Fyfe
1 was a wee cooper
2 wee
3 cooper
4 lived in Fyfe

匹配对象

- 想知道与给定编组匹配的内容?
  - match()函数不是返回True/False，而是返回一个MatchObject
  - 通过分组可以查看MatchObject中的内容

```
>>> m = re.match(r'www\.(.*)\..{3}', 'www.python.org')
>>> m.group(1)
???
>>> m.start(1)
???
>>> m.end(1)
???
>>> m.span(1)
???
```

返回与模式中编组1匹配的子串

返回与模式中编组1匹配的子串 的起始索引

返回与模式中编组1匹配的子串 的终止索引

返回一个元组，包含与模式中编组1匹配的子串 的起始和终止索引

re.sub进阶

- 例：将纯文本文档的一部分转换成html代码
  - 例：使用re.sub和编组匹配将文本中

    '*something*'（两端为*，中间为除了*以外的任何字符）替换为

    'something'　　# 是emphasis的缩写

  - 先来创建一个正则表达式

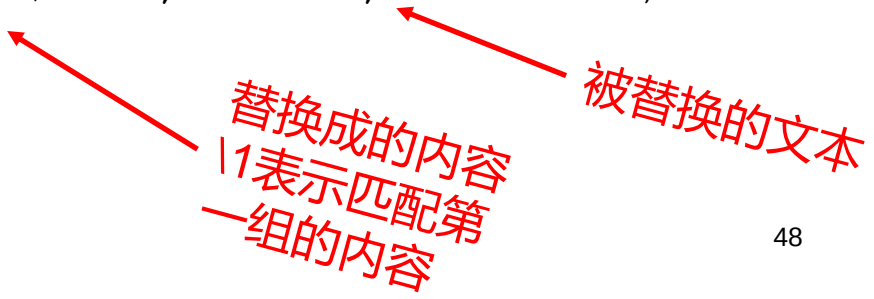
```
>>> emphasis_pattern = r'\*([^\*]+)\*'
```

1．如何解读此正则式？　2.分组情况?

  - 使用re.sub() 来替换

```
>>> re.sub(emphasis_pattern, r'<em>\1</em>', 'Hello, *world*!')
```

输出：'Hello, world!'



例：找到发件人姓名

- 假设有一封邮件，保存在message.eml文件中
- 目标：在邮件中找出发件人姓名，不包含邮件地址

Subject: Re: Spam
From: Foo Fie <foo@bar.baz>
To: Magnus Lie Hetland <magnus@bozz.floop>
CC: <Mr.Gumby@bar.baz>
Message-ID: <B8467D62.84F%foo@baz.com>
In-Reply-To: <20041219013308.A2655@bozz.floop> Mime- version: 1.0
Content-type: text/plain; charset="US-ASCII" Content-transfer-encoding: 7bit
Status: RO
Content-Length: 55
Lines: 6
So long, and thanks for all the spam!
Yours,
Foo Fie

解决方案

- **思路**：以‘From:’开头，<>尖括号结尾，中间的部分就是姓名

```
#在find_sender.py脚本内，有如下代码

import fileinput, re

pat = re.compile('From: (.*) <.*?>$')
for line in fileinput.input():
        m = pat.match(line)
        if m: print(m.group(1))
```

用()编组，此处为姓名部分

$结尾符，表示要匹配整行

使用?，非贪婪模式，只匹配最后一对尖括号

（假设电子邮件保存在文本文件message.eml中），可以运行程序：

```
$ python find_sender.py message.eml
Foo Fie
```

正则中其他特殊元素

\w	匹配数字字母下划线
\W	匹配非数字字母下划线
\s	匹配任意空白字符，等价于 [\t\n\r\f]。
\S	匹配任意非空字符
\d	匹配任意数字，等价于 [0-9]。
\D	匹配任意非数字
\A	匹配字符串开始
\Z	匹配字符串结束，如果是存在换行，只匹配到换行前的结束字符串。
\z	匹配字符串结束
\G	匹配最后匹配完成的位置。
\b	匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。
\B	匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'。
\n, \t, 等。	匹配一个换行符。匹配一个制表符, 等
\1...\9	匹配第n个分组的内容。
\10	匹配第n个分组的内容，如果它经匹配。否则指的是八进制字符码的表达式。

总结

- 介绍了模块
  - 创建模块、包、以及模块中的各种内置变量
    - __all__
    - __file__
    - __doc__
  - 介绍了程序入口： __name__ == '__main__’
  - 介绍了fileinput、random、shelve、re等标准库
    - 其中重点介绍了re模块的使用

正则练习

- 判断字符串是否全部是小写字母，应如何设置**模式**?
  - `s1 = 'adkkdk'`
  - `s2 = 'Hello 1234567 is a number. Regex String'`
- 取出字符串中的数字，如有下列字符串
  - content = 'Hello 1234567 is a number. Regex String'
- 匹配 如2016-12-24的日期格式.
  - s1= '他的生日是2016-12-12 14:34,是个可爱的小宝贝.二宝的生日是2016-12-21 11:34,好可爱的.'

正则练习一

- 判断字符串是否全部是小写字母，应如何设置**模式**?
  - `s1 = 'adkkdk'`
  - `s2 = 'Hello 1234567 is a number. Regex String'`

```
an =  re.match('[a-z]+$', s1)
if an:
    print  ('s1:', an.group(), '全为小写')
```

正则练习二

- 取出字符串中的数字，如有下列字符串
  - content = 'Hello 1234567 is a number. Regex String'

初级版：

```
result = re.match('^Hello (\d+).*String$', content)
if result: print(result.group(1))
```

改进版：

```
result = re.match('.*(\d+).*', content)
```

正则练习三

- 匹配 如2016-12-24的日期格式.
  - s1= '他的生日是2016-12-12 14:34,是个可爱的小宝贝.二宝的生日是2016-12-21 11:34,好可爱的.'

用search只能搜到第一个日期：

```
result = re.search(r"(\d{4}-\d{1,2}-\d{1,2})",s1)
print(result.group(0))
```

用findall更好

```
result = re.findall(r"(\d{4}-\d{1,2}-\d{1,2})",s1)
print(result)
```

- 感谢