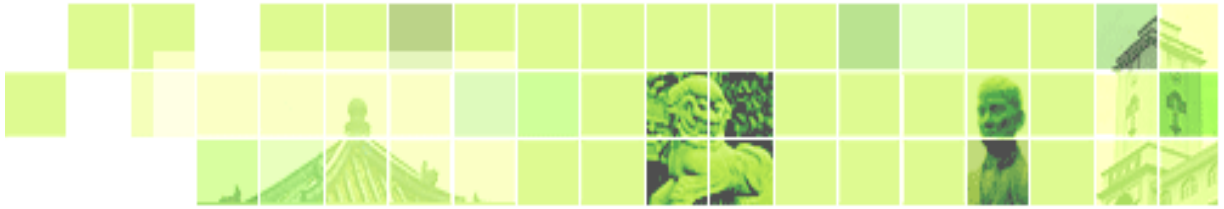
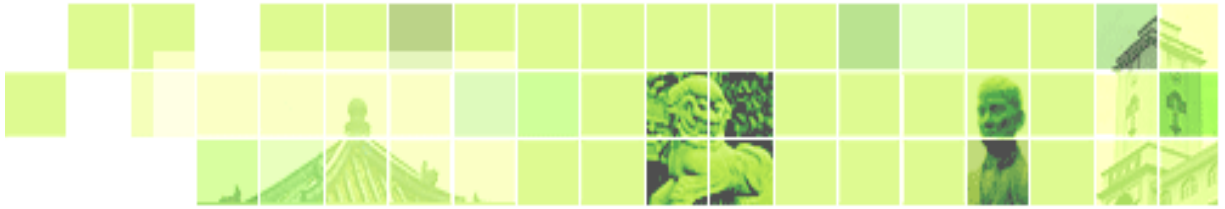
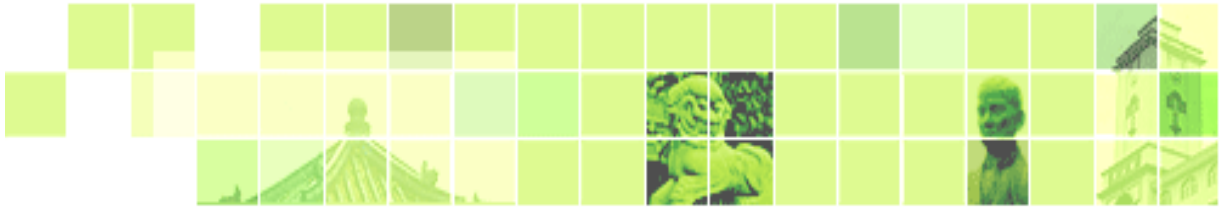


第2节练习：Python正则表达式



正则表达式

- 又称正规表达式，Regular Expression
- 在代码中常简写为regex、regexp或RE
- 使用当个字符串来描述、匹配一系列某个句法规则的字符串
- 常被用来检索、替换那些匹配某个模式的文本



正则表达式 vs 字符串处理

假设爬取某网页的HTML源码，截取其中一段如下：

```
<html><body><h1>hello world<h1></body></html>
```

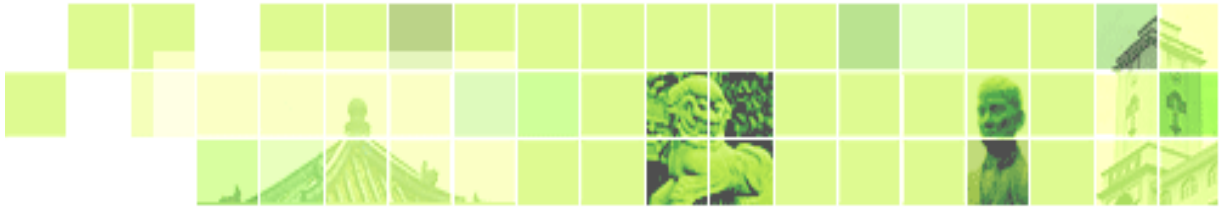
若想提取“hello world”：

- Python字符串处理

```
s = <html><body><h1>hello world<h1></body></html>
start_index = s.find('<h1>')
```

然后从此位置向下查找下一个“<h1>”...

需要考虑多个标签，若想做到准确判断需循环判断，效率较低



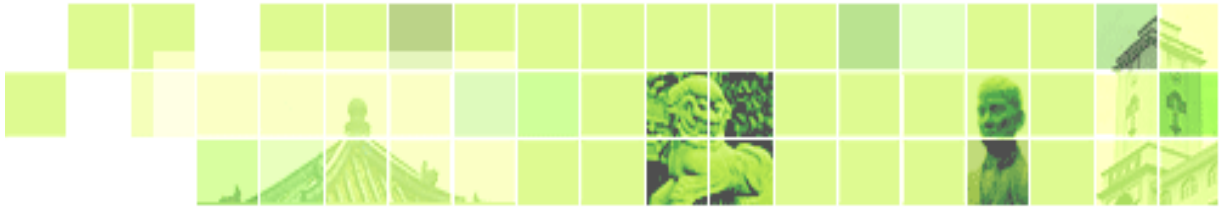
正则表达式 vs 字符串处理

- 正则表达式

```
import re

key = r"<html><body><h1>hello world<h1></body></html>" #要匹配的文本
p1 = r"(?<=<h1>).+?(?=<h1>)" #正则表达式规则
pattern1 = re.compile(p1) #编译正则表达式
matcher1 = re.search(pattern1,key) #在源文本中搜索符合正则表达式的部分
print matcher1.group(0)#打印
```

（python 2.7）

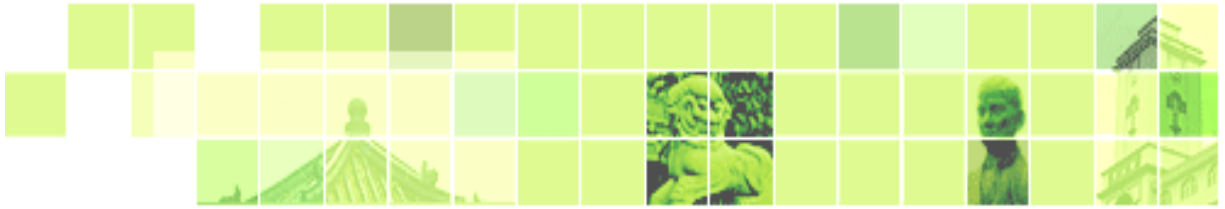
- 优点：代码少，效率高，便捷简单



Python 正则表达式

匹配字符串“javapythonhtmlvhdl”中所有的“python”

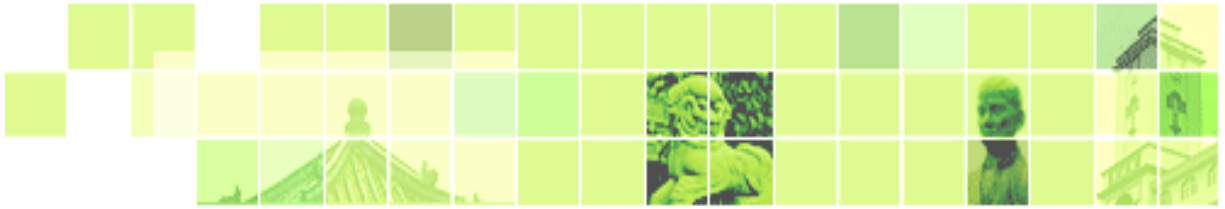
```
import re

key = r"javapythonhtmlvhdl"
p1 = r"python"
pattern1 = re.compile(p1)
matcher1 = re.search(pattern1,key)
print matcher1.group(0)
```



Python 正则表达式

```
import re

key = r"<h1>hello world<h1>" #源文本
p1 = r"<h1>.+<h1>" #正则表达式规则
pattern1 = re.compile(p1)
print pattern1.findall(key)
```

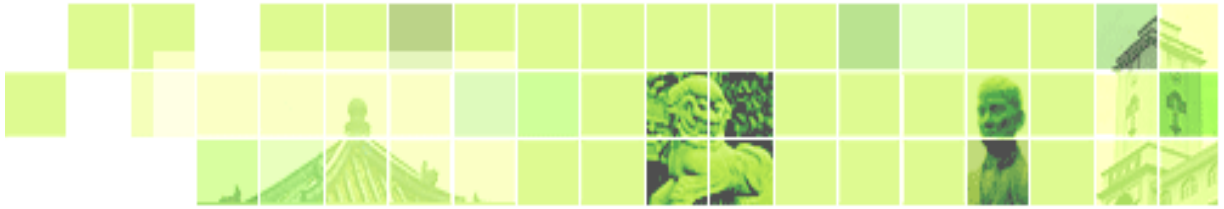
- “.”字符在正则表达式中代表任何一个字符（包括它本身）
- “findall”返回所有符合要求的元素列表（包括仅有一个元素时）



Python 正则表达式

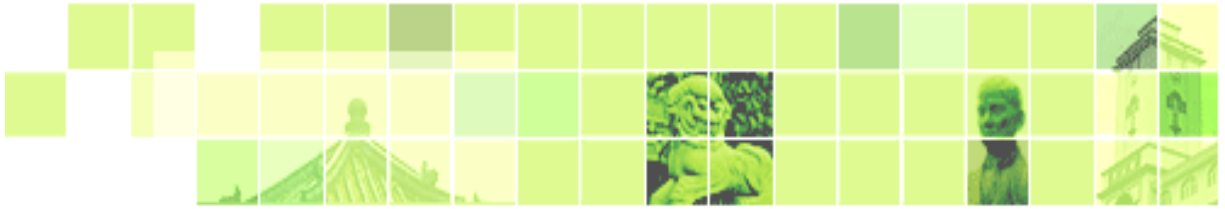
```
import re

key = r"afiouwehrfuichuxiuhong@hit.edu.cnaskdjhfiosueh"
p1 = r"chuxiuhong@hit\.edu\.cn"
pattern1 = re.compile(p1)
print pattern1.findall(key)
```

- 当匹配“.”本身时，需要在“.”前加转义符“\”
- “+”：将前面的一个字符或一个子表达式重复一次或多次
- “*”：将前面的字符或表达式重复0次或多次



练习

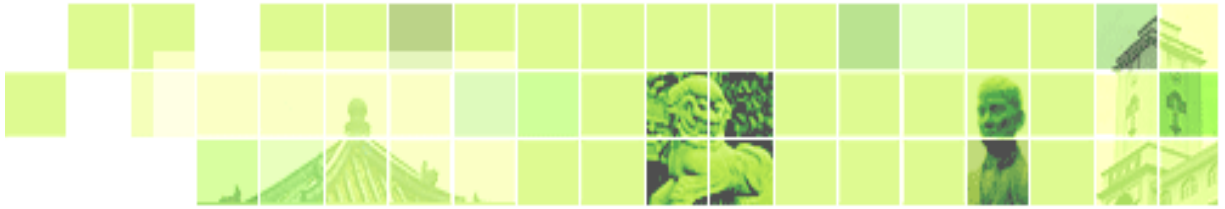
请在网页链接“http://www.poshoaiu.com and https://iusdhbfw.com”中，匹配出“http://”和“https://”



解答

请在网页链接“http://www.poshoaiu.com and https://iusdhbfw.com”中，匹配出“http://”和“https://”

```
import re

key = r"http://www.poshoaiu.com and https://iusdhbfw.com"
p1 = r"https*://" #注意“*”
pattern1 = re.compile(p1)
print pattern1.findall(key)
```



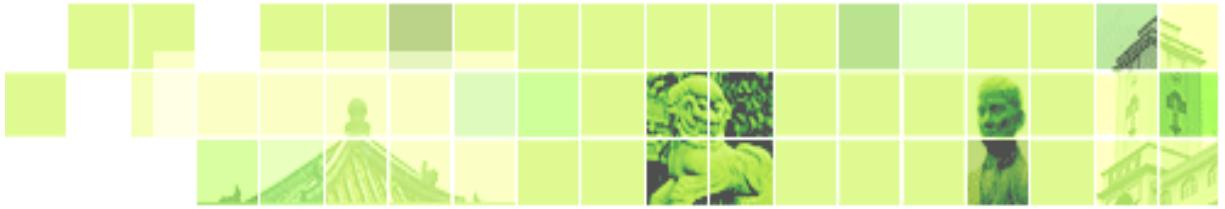
正则表达式：多字符匹配

```
import re

key = r"12345<hTml>hello</Html>67890"
p1 = r"<[Hh][Tt][Mm][Ll]>.+?</[Hh][Tt][Mm][Ll]>"
pattern1 = re.compile(p1)
print pattern1.findall(key)
```

输出

```
['<hTml>hello</Html>']
```

- “[]”：匹配括号中字符的任意一个



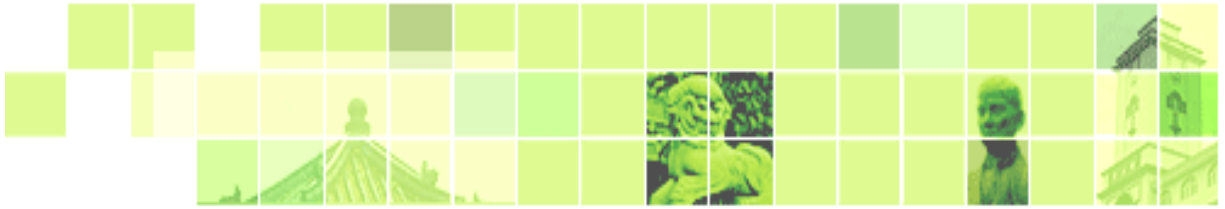
正则表达式：多字符匹配

```
import re

key = r"12345<hTml>hello</Html>67890"
p1 = r"<[Hh][Tt][Mm][Ll]>.+?</[Hh][Tt][Mm][Ll]>"
pattern1 = re.compile(p1)
print pattern1.findall(key)
```

输出

```
['<hTml>hello</Html>']
```

- “[]”：匹配括号中字符的任意一个



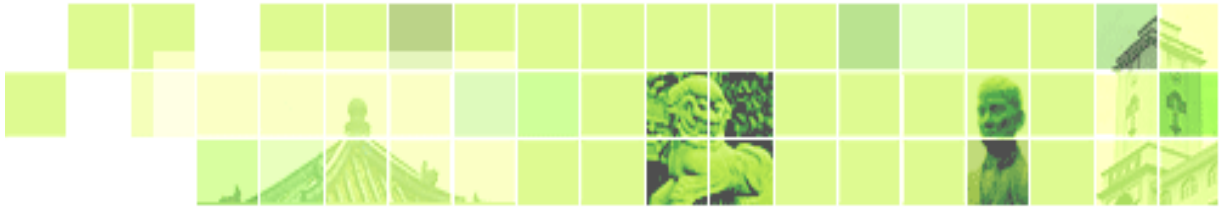
正则表达式：多字符匹配

```
import re

key = r"mat cat pat"
p1 = r"[^p]at" #除p以外都匹配
pattern1 = re.compile(p1)
print pattern1.findall(key)
```

输出

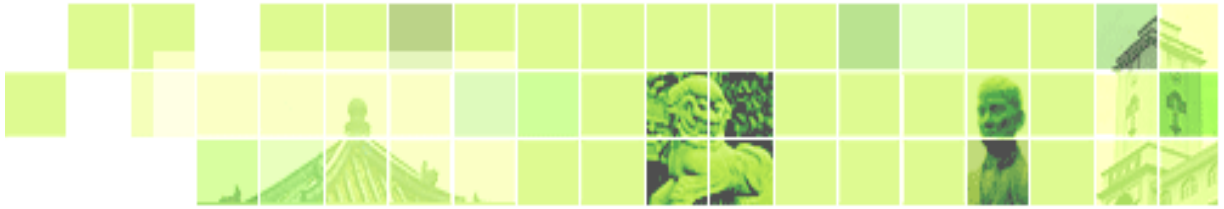
```
['mat cat ']
```

- “[^]” ：除内部包含字符外都能匹配



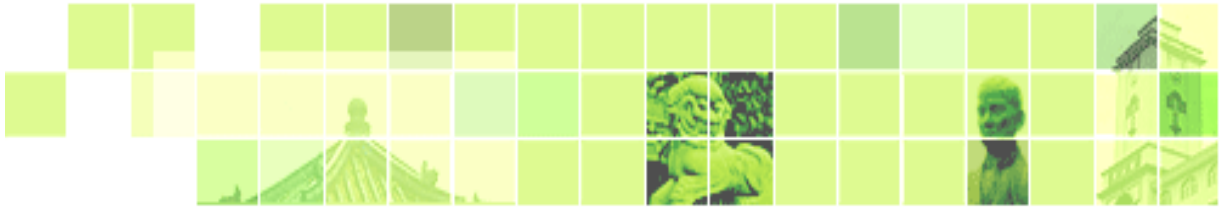
简洁正则表达式

正则表达式	匹配字符
[0-9]	0123456789任意之一
[a-z]	任意小写字母
[A-Z]	任意大写字母
\d	等同于[0-9]
\D	等同于[^0-9]，匹配非数字
\w	等同于[a-z0-9A-Z_]，匹配大小写字母、数字和下划线
\W	等同于[^a-z0-9A-Z_]



练习

请在字符串“sysu@hotmail.edu.cn”中，匹配出“@hotmail.”



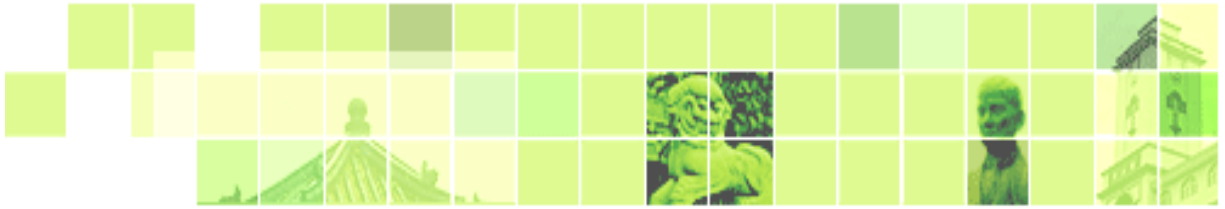
错误解答

请在字符串“sysu@hotmail.edu.cn”中，匹配出“@hotmail.”

```
import re

key = r"sysu@hotmail.edu.cn"
p1 = r"@.+\."
pattern1 = re.compile(p1)
print pattern1.findall(key)
```

输出

```
['@hotmail.edu.']
```



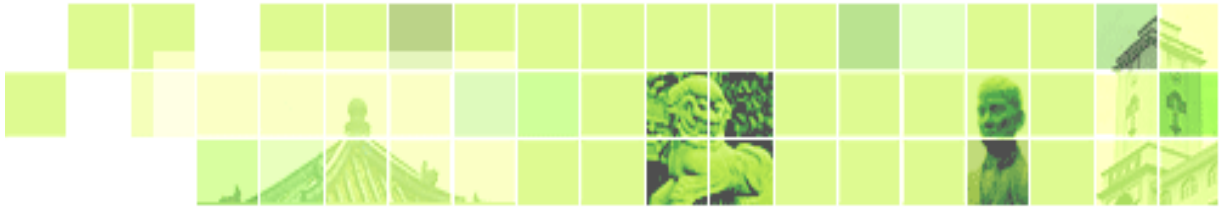
正解

```
import re

key = r"sysu@hotmail.edu.cn"
p1 = r"@.+?\."
pattern1 = re.compile(p1)
print pattern1.findall(key)
```

输出

```
['@hotmail.']
```

- 正则表达式默认是贪婪的。
- “+”代表字符重复一次或多次，尽可能贪婪地多匹配字符，在本例中错解即匹配到最后一个“.”
- 在“+”后加“？”，即将“贪婪”转换为“懒惰”



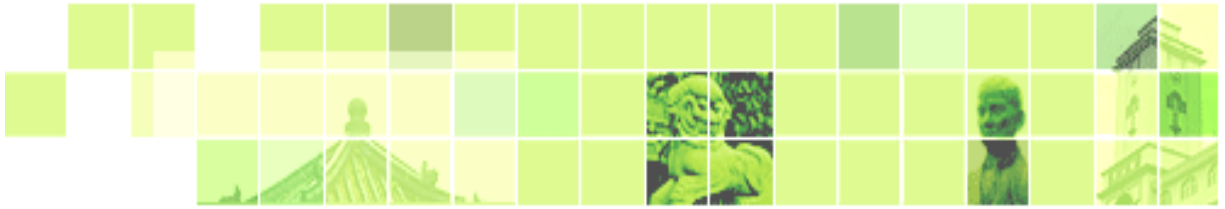
正则表达式：匹配次数控制

匹配字符串“saas and sas and saaas”中的“saas”和“sas”

```
import re

key = r"saas and sas and saaas"
p1 = r"sa{1,2}s"
pattern1 = re.compile(p1)
print pattern1.findall(key)
```

输出

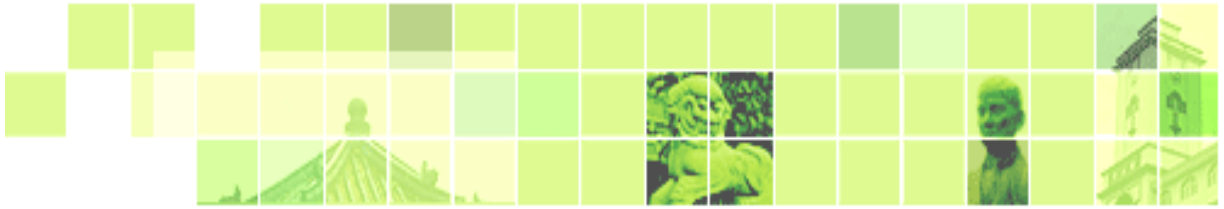
```
['saas', 'sas']
```



正则表达式：匹配次数控制

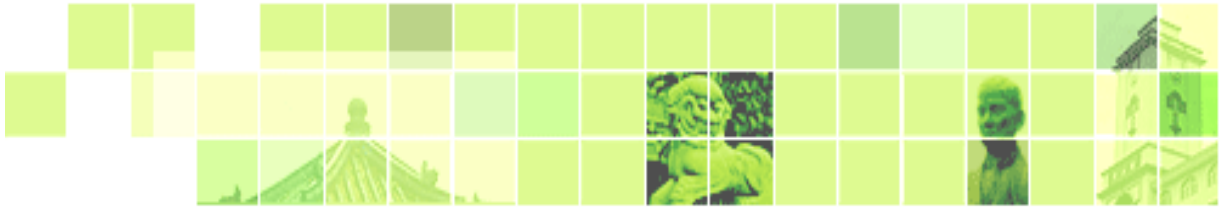
```
import re

key = r"saas and sas and saaas"
p1 = r"sa{1,2}s"
pattern1 = re.compile(p1)
print pattern1.findall(key)
```

- {a,b}：a<=匹配次数<=b
- 能够准确控制重复次数
- 当a=1且b省略时，代表至少匹配一次，等价于“？”



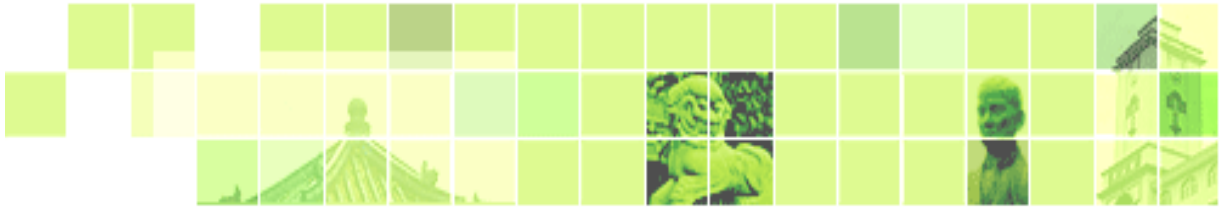
扩展练习

请用非懒惰匹配的方法，在字符串“sysu@hotmail.edu.cn”中，匹配出“@hotmail.”



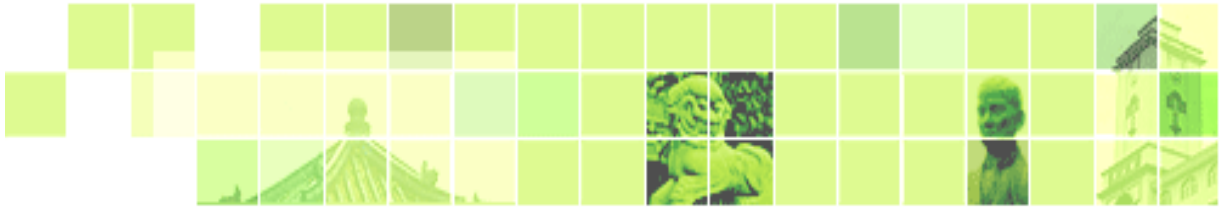
元字符及其作用

正则表达式	匹配字符
.	任意字符
|	逻辑“或”操作符
[]	匹配内部的任一字符或子表达式
[^]	对字符集取非
-	定义一个区间
\	对下一字符取非（通常是普通变特殊，特殊变普通）
*	匹配前面的字符或者子表达式0次或多次



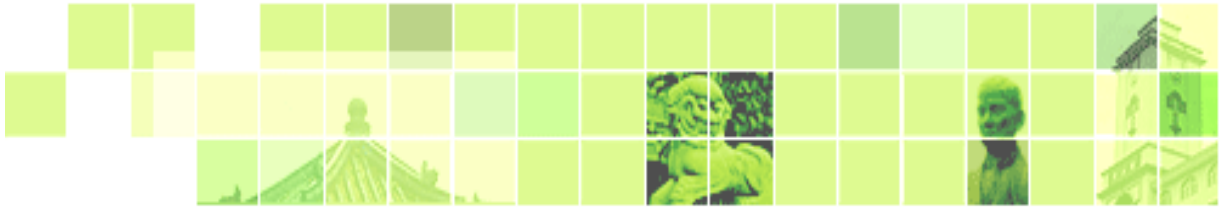
元字符及其作用

正则表达式	匹配字符
* ?	惰性匹配上一个字符0次或多次
+	匹配前一个字符或表达式一次或多次
+ ?	惰性匹配上一个字符一次或多次
?	匹配前一个字符或子表达式0次或一次
{n}	匹配前一个字符或子表达式
{m,n}	匹配前一个字符或子表达式至少m次，至多n次
{n,}	匹配前一个字符或子表达式至少n次



元字符及其作用

正则表达式	匹配字符
{n,}?	{n,}的惰性匹配
^	匹配字符串的开头
\A	匹配字符串的开头
$	匹配字符串的结束
[\b]	退格字符
\c	匹配一个控制字符
\t	匹配制表符



Thank you !