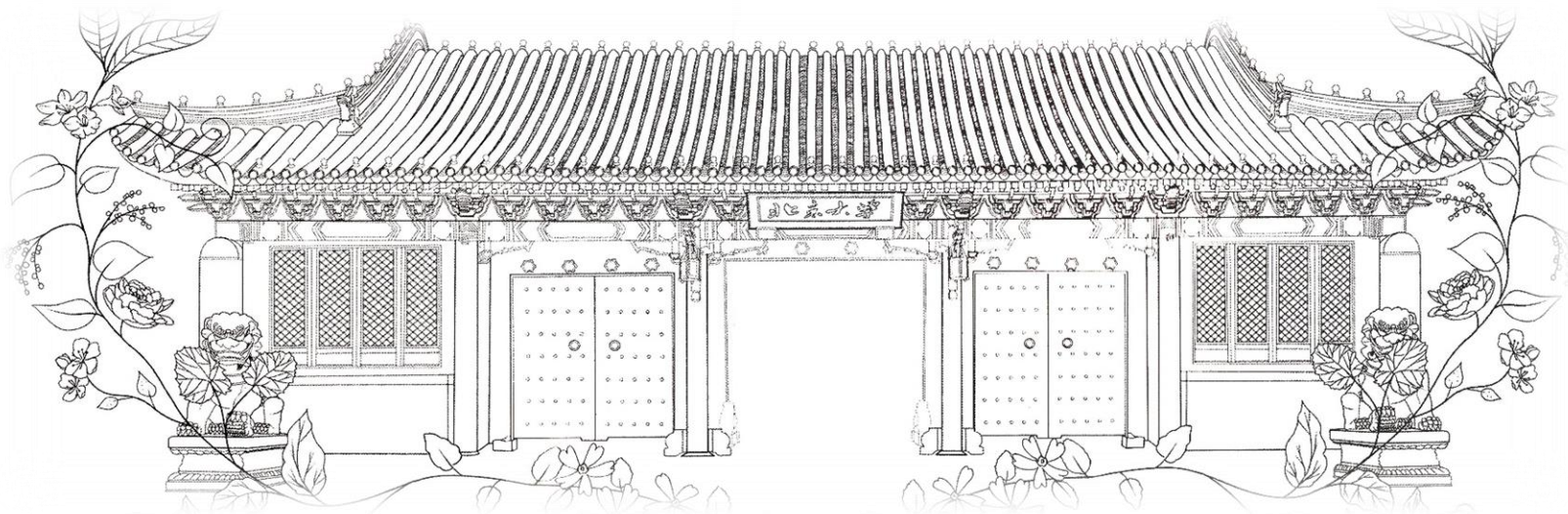


# 《Python数据分析》

## 应用篇：文本数据分析基础





# 目录

---

- Python 中文本/字符串的处理
- 文本数据分析及其任务
- 文本可视化分析



# Python 中 文 本 / 字 符 串 的 处 理





# 字符串

---

- 在Python中，字符串是除数字外最重要的数据类型。字符串无处不在：将字符串输出到屏幕上；从用户的键盘输入读取字符串；文件通常被视为大型字符串；网页大部分是由文本组成的。
- 字符串是一种聚合数据结构，可充分利用索引和切片，用于从字符串中提取子串。
  - Python正则表达式库，是一种用来处理字符串的微型语言，但功能强大。



# Python 中文本/字符串的处理

---

- 通用序列操作
- 字符串格式化与转义
- 字符串函数（方法）
- 正则表达式



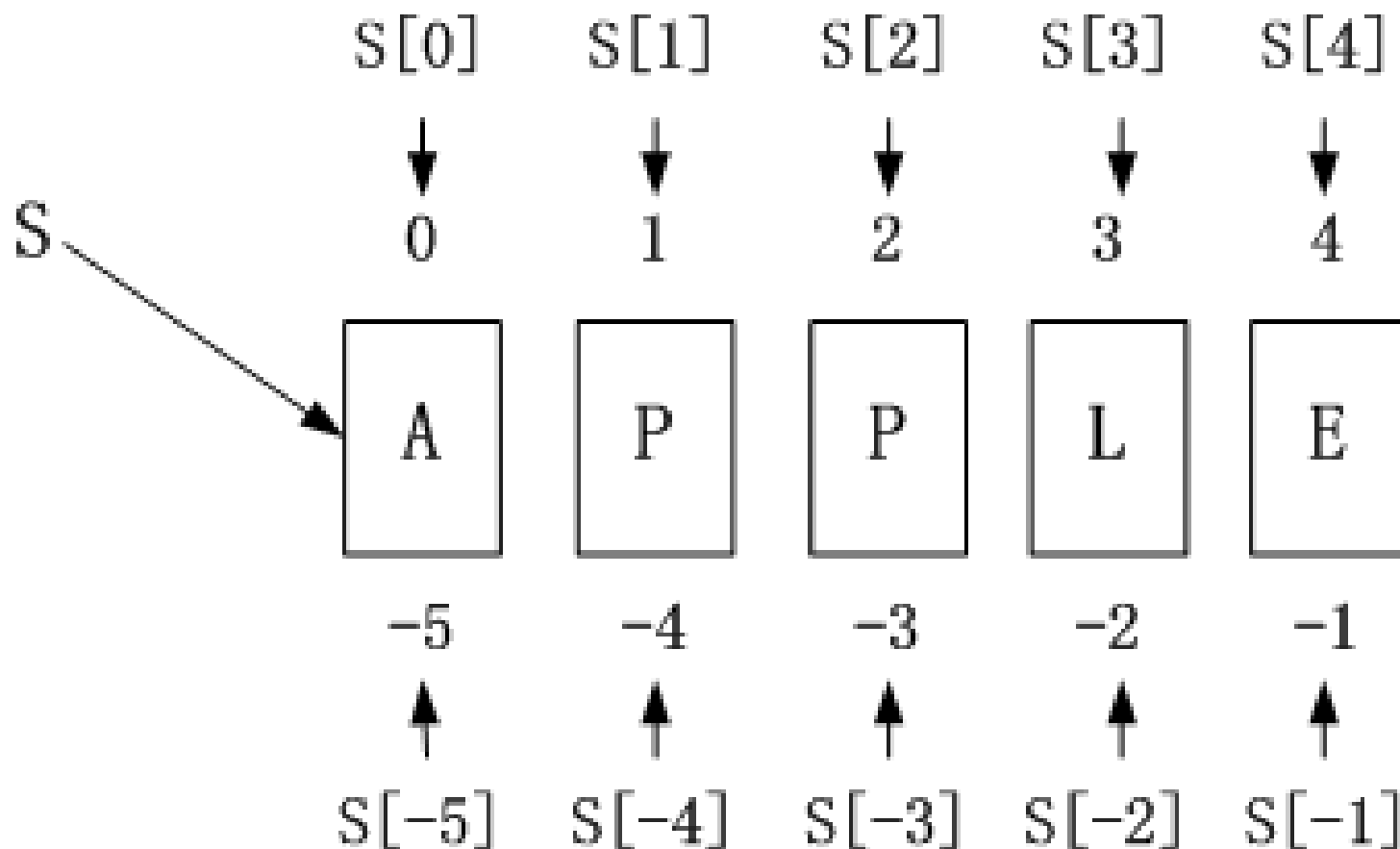
# 通用序列操作

---

- Python中，字符串、列表和元组都属于序列。
- 序列有一些通用的操作。
  - 索引 (indexing)
  - 分片 (slicing)
  - 加 (adding)
  - 乘 (multiplying)
  - 检查某个元素是否属于序列的成员 (成员资格)
  - 计算序列长度
  - 找出最大元素和最小元素
  - .....



# 通用序列操作：索引





# 通用序列操作：索引

---

```
s='apple'
for i in range(len(s)):
    print(s[i],end=" ")
print()
for i in range(len(s)):
    print(s[-(i+1)],end=" ")
```

执行结果：

```
apple
elppa
```





# 通用序列操作：索引

- 计算给定字符串的编码总和

```
def codesum(s):  
    """  
    Return the sums of the  
    character codes of s.  
    """  
    total=0  
    for c in s:  
        total=total+ord(c) #ord() 函数获取字符的Unicode编码,  
                           #该编码的前256个字符为ASCII码  
    return total  
  
print(codesum('Hi there!'))
```

执行结果：

778





# 通用序列操作：分片

- 返回文件名中的扩展名

执行结果：

```
def get_ext(fname):  
    """  
    Return the extension of file fname  
    """  
    dot=fname.rfind('.') #确定最右边的'.'的索引，  
                        #因此使用rfind从右往左查找  
    if dot==-1: #fname中没有  
        return ''  
    else:  
        return fname[dot+1:]  
  
print(get_ext('hello.text'))  
print(get_ext('pizza.py'))  
print(get_ext('pizza.old.py'))  
print(get_ext('pizza'))
```

text

py

py





# 通用序列操作：加

```
>>> "Hello, " + "world!"  
'Hello, world!'
```

## 实例

```
#!/usr/bin/python3  
  
s1 = "-"  
s2 = ""  
seq = ("r", "u", "n", "o", "o", "b") # 字符串序列  
print (s1.join( seq ))  
print (s2.join( seq ))
```

以上实例输出结果如下：

```
r-u-n-o-o-b
```

```
runoob
```



# 通用序列操作：乘；成员资格

---

```
>>> 'Python' * 5  
'PythonPythonPythonPythonPython'
```

```
>>> permissions='rw'  
>>> 'w' in permissions  
True  
>>> 'x' in permissions  
False
```



# 通用序列操作：长度、最值

---

- len、min和max都是内置函数。
- len—返回序列中包含的元素个数。
- min—返回序列中的最小值。
- max—返回序列中的最大值。

```
>>> len('Hello world!')
```

```
12
```

```
>>> max('bcd', 'fig', 'abcd', 'xyz', 'abab')
```

```
'xyz'
```

```
>>> min('bcd', 'fig', 'abcd', 'xyz', 'abab')
```

```
'abab'
```



# 字符串格式化

```
str1="version"  
num=1.0  
format="%s" % str1  
print(format)  
format="%s %d" % (str1, num)  
print(format)
```

执行结果:

```
version  
version 1
```

```
print("浮点型数字: %f" % 1.25) #以浮点格式输出  
print("浮点型数字: %.1f" % 1.25) #精确到小数点后1位  
print("浮点型数字: %.2f" % 1.254) #精确到小数点后2位  
print("浮点型数字: %.2f" % 1.256) #精确到小数点后2位
```

执行结果:

```
浮点型数字: 1.250000  
浮点型数字: 1.2  
浮点型数字: 1.25  
浮点型数字: 1.26
```





# 字符串的转义字符

符号	描述	符号	描述
\\	反斜线	\v	纵向制表符
\'	单引号	\r	回车符
\"	双引号	\f	换页符
\a	发出系统响铃声	\o	八进制数代表的字符
\b	退格符	\x	十六进制数代表的字符
\n	换行符	\000	终止符，其后的字符串全部忽略
\t	横向制表符		

## 执行结果:

```
' and " are quotes
\ must be written \\
one
two
three
hello    world

hello\tworld\n
1
5
```

```
print('\\' and \" are quotes')
print('\\ must be written \\\') #"\\"", 第1、3反斜线表示转义字符,
                                #第2、4反斜线表示要输出的反斜线

print('one\ntwo\nthree')
print("hello\tworld\n")
print(r"hello\tworld\n") #忽略转义字符的作用, 直接输出字符串原始内容
print(len('\\'))         #计算字符长度时, 不包括转义字符"\\"
print(len('a\nb\nc'))
```





# 字符串函数（方法）

- 字符串测试函数

```
>>> s='Hello world!'
>>> s.startswith('h')
False
>>> s.endswith('!')
True
>>> s.islower()
False
>>> s.isupper()
False
>>> s.isprintable()
True
```

函数名	何时返回True	函数名	何时返回True
s.endswith(t)	s以字符串t结尾	s.islower()	s只包含小写字母
s.startswith(t)	s以字符串t打头	s.isnumeric()	s只包含数字
s.isalnum()	s只包含字母和数字	s.isprintable()	s只包含可打印字符
s.isalpha()	s只包含字母	s.isspace()	s只包含空白字符
s.isdecimal()	s只包含表示十进制数字的字符	s.istitle()	s是个大小写符合标题要求的字符串
s.isdigit()	s只包含数字字符	s.isupper()	s只包含大写字母
s.isidentifier()	s是合法的标识符	t in s	s包含字符串t







# 字符串函数（方法）

函数index和find之间的差别在于没有找到指定子串的情形。函数index引发异常ValueError，而函数find将返回-1。

- 字符串查找函数

函数名	返回值
<b>s.find(t)</b>	若未找到字符串t，则返回-1；否则返回t在s中的起始位置
<b>s.rfind(t)</b>	与find相同，但从右往左查找
<b>s.index(t)</b>	与find相同，但如果在s中找不到t，则引发ValueError异常
<b>s.rindex(t)</b>	与index相同，但从右往左查找



# 字符串函数（方法）

- 字符串替换函数

函数名	返回的字符串
<code>s.replace(old,new)</code>	将s中的每个old替换为new
<code>s.expandtabs(n)</code>	将s中的每个制表符扩展为空格，空格宽度为n

```
s='up, up and away'
s1='up, \tup\tand\taway'
print(s.replace('up','down'))
print(s.replace('up',''))
print(s1)
print(s1.expandtabs(8))
print(s1.expandtabs(10))
```

执行结果:

```
down, down and away
, and away
up,      up      and      away
up,      up      and      away
up,      up              and      away
```



# 字符串函数（方法）

- 字符串拆分函数

函数	返回的字符串
<code>s.partition(t)</code>	将s拆分为三个字符串（ <b>head</b> 、 <b>t</b> 和 <b>tail</b> ），其中 <b>head</b> 为t前面的子串， <b>tail</b> 为t后面的子串。返回值为元组
<code>s.rpartition(t)</code>	与 <b>partition</b> 相同，但从s的右端开始搜索t。返回值为元组
<code>s.split(t)</code>	以t为分隔符，将s划分成一系列子串，并返回一个由这些子串组成的列表
<code>s.rsplit(t)</code>	与 <b>split</b> 相同，但从s的右端开始搜索t
<code>s.splitlines()</code>	返回一个由s中的各行组成的列表



# 字符串函数（方法）

- 字符串拆分函数

```
url='www.google.com'
print(url.partition('.'))
print(url.rpartition('.'))
print(url.split('.'))
story='A long time ago, a princess ate an apple.'
print(story.split())
```

执行结果:

```
('www', '.', 'google.com')
('www.google', '.', 'com')
['www', 'google', 'com']
['A', 'long', 'time', 'ago,', 'a', 'princess', 'ate', 'an', 'apple.']
```





# 字符串函数（方法）

- 字符串拆分函数

```
sentence="Bob said: 1, 2, 3, 4"
```

```
#根据空格获取子串，原串中含5个空格，返回6个子串
```

```
print("使用空格取子串",sentence.split())
```

```
#根据逗号取子串，原串中含3个逗号，返回4个子串
```

```
print("使用逗号取子串",sentence.split(","))
```

```
#根据逗号个数分割字符串，将原串分割为3个子串
```

```
print("使用两个逗号取子串",sentence.split(",",2))
```

## 执行结果：

```
使用空格取子串 ['Bob', 'said:', '1,', '2,', '3,', '4']
```

```
使用逗号取子串 ['Bob said: 1', ' 2', ' 3', ' 4']
```

```
使用两个逗号取子串 ['Bob said: 1', ' 2', ' 3, 4']
```





# 字符串函数（方法）

---

- 字符串与日期时间的转换
  - 已经在“时间序列分析基础”部分讲过！



# 字符串函数（方法）

在这些函数中，Python都创建并返回一个新字符串，Python不会真正修改原字符串。

- 字符串大小写转换函数

函数名	返回的字符串
<code>s.capitalize()</code>	将s[0]改为大写，其余小写
<code>s.lower()</code>	让s的所有字母都小写
<code>s.upper()</code>	让s的所有字母都大写
<code>s.swapcase()</code>	将小写字母改为大写，并将大写字母改为小写
<code>s.title()</code>	让s的大小写符合标题的要求



# 字符串函数（方法）

- 字符串剥除函数

- 剥除函数用于删除字符串开头或末尾多余的字符
- 默认情况下，剥除空白字符；如果指定了字符串参数，则剥除该字符串中的字符

函数名	返回的字符串
<code>s.strip(ch)</code>	从s开头和末尾删除所有包含在字符串ch中的字符
<code>s.lstrip(ch)</code>	从s开头（左端）删除所有包含在字符串ch中的字符
<code>s.rstrip(ch)</code>	从s末尾（右端）删除所有包含在字符串ch中的字符





# 字符串函数（方法）

- 字符串剥除函数

```
name=' Gill Bates '  
print(name.lstrip())  
print(name.rstrip())  
print(name.strip())  
title='__-Happy Days!!__-'  
print(title.strip())  
print(title.strip('__-'))
```

执行结果:

```
Gill Bates  
Gill Bates  
Gill Bates  
__-Happy Days!!__-  
Happy Days!!
```



# 字符串函数（方法）

- 去除转义字符

```
word="\thello world\n"
print("直接输出：",word)
print("strip()后输出：",word.strip()) #去除转义字符
print("lstrip()后输出：",word.lstrip()) #去除字符串前面的"\t", 末尾的"\n"依然保留
print("rstrip()后输出：",word.rstrip()) #去除字符串末尾的"\n", 前面的"\t"依然保留
```

## 执行结果：

```
直接输出：          hello world

strip()后输出：  hello world
lstrip()后输出：  hello world

rstrip()后输出：   hello world
```





# 字符串函数（方法）

- 字符串比较函数

```
str1=1
str2="1"
if str1==str2:
    print("相同")
else:
    print("不相同")
if str(str1)==str2:
    print("相同")
else:
    print("不相同")
```

执行结果:

不相同  
相同



# 字符串函数（方法）

- 字符串比较函数

```
word="hello world"
print("hello"==word[0:5]) #先取子串再与"hello"比较
print(word.startswith("hello")) #比较字符串的开头部分"hello"
print(word.endswith("ld",6)) #从字符串的结尾处到word[6]之间搜索子串"ld"
#从"切片"word[6:10]中搜索子串"ld", 注意搜索的字符不包括位置10所在的字符
print(word.endswith("ld",6,10))
print(word.endswith("ld",6,len(word))) #从切片word[6:len(word)]中搜索子串
```

执行结果:

```
True
True
True
False
True
```





# 正则表达式

---

- 正则表达式（Regular Expression，代码中常简写为regex或RE）。
  - 正则表达式使用单个字符串来描述、匹配一系列匹配某个句法规则的字符串。



*Stephen C. Kleene*

# 何时需要正则表达式？

- 在开发过程中，经常会对用户输入信息（如手机号、身份号码、邮箱、密码、域名、IP地址、URL等）做校验。
  - 正则表达式能很好地解决这类字符串校验问题。它是一串由特定意义的字符组成的字符串，表示某种匹配的规则。
  - 正则表达式由美国数学家斯蒂芬·科尔·克莱尼（Stephen Cole Kleene）于1956年提出，主要用于描述正则集代数。
  - 正则表达式能够应用在多种操作系统中，几乎所有程序设计语言都支持它。
- 正则表达式最基本的3种功能是查找、分组及替换。





# 应用场景

## 欢迎注册QQ

每一天，乐在沟通。

免费靓号

昵称

! 昵称不可以为空

密码

- ✓ 不能包括空格
- ! 长度为8-16个字符
- ! 必须包含字母、数字、符号中至少2种

+86 ▼

139123456X

! 手机号码格式不正确

短信验证码

发送短信验证码

立即注册

## 规则验证





# 应用场景

## 成份股列表

上证50指数依据样本稳定性和动态跟踪相结合的原则，每半年调整一次成份股  
每次调整的比例一般情况不超过10%。样本调整设置缓冲区，排名在40名之前的：

- 2018-10-11发布

[浦发银行 \(600000\)](#) [民生银行 \(600016\)](#) [宝钢股份 \(600019\)](#)  
[中国石化 \(600028\)](#) [南方航空 \(600029\)](#) [中信证券 \(600030\)](#)  
[招商银行 \(600036\)](#) [保利地产 \(600048\)](#) [中国联通 \(600050\)](#)  
[上汽集团 \(600104\)](#) [北方稀土 \(600111\)](#) [恒瑞医药 \(600276\)](#)  
[万华化学 \(600309\)](#) [华夏幸福 \(600340\)](#) [贵州茅台 \(600519\)](#)  
[山东黄金 \(600547\)](#) [海螺水泥 \(600585\)](#) [绿地控股 \(600606\)](#)  
[青岛海尔 \(600690\)](#) [三安光电 \(600703\)](#) [伊利股份 \(600887\)](#)  
[东方证券 \(600958\)](#) [招商证券 \(600999\)](#) [大秦铁路 \(601006\)](#)  
[中国神华 \(601088\)](#) [兴业银行 \(601166\)](#) [北京银行 \(601169\)](#)  
[中国铁建 \(601186\)](#) [国泰君安 \(601211\)](#) [上海银行 \(601229\)](#)  
[农业银行 \(601288\)](#) [中国平安 \(601318\)](#) [交通银行 \(601328\)](#)  
[新华保险 \(601336\)](#) [三六零 \(601360\)](#) [中国中铁 \(601390\)](#)  
[工商银行 \(601398\)](#) [中国太保 \(601601\)](#) [中国人寿 \(601628\)](#)  
[中国建筑 \(601668\)](#) [华泰证券 \(601688\)](#) [中国中车 \(601766\)](#)  
[中国交建 \(601800\)](#) [光大银行 \(601818\)](#) [中国石油 \(601857\)](#)  
[浙商证券 \(601878\)](#) [中国银河 \(601881\)](#) [中国银行 \(601988\)](#)  
[中国重工 \(601989\)](#) [洛阳钼业 \(603993\)](#)

获取网页中所有的股票代码  
特征：6位数字

$\backslash d\{6\}$







# 应用场景

- 3 · 基础语法
- 4 · 流程控制
- 5 · 字符串

替换前

Find: `^(\d)`

Replace: 第\$1章

Options: ☒ Regular Expression ☐ Ignore Whitespace  
☐ Ignore Case ☐ Wrap Around

In: Document matching \*

▼ demo.txt 替换后

1:	第3章	基础语法
2:	第4章	流程控制
3:	第5章	字符串



# 常用正则函数

---

查找

所有匹配

`re.findall(pattern, line)`

查找

第1个匹配

`re.search(pattern, line)`

替换

`re.sub(pattern, new, line)`

切分

`re.split(pattern, line)`

迭代查找

`re.finditer(pattern, line)`

查找行首

`re.match(pattern, line)`



# 正则示例：查找 findall

---

- `import re`
- 
- `line = "JAVA PHP Python C++ Perl SQL";`
- `pattern = r'\w+'`
- `r = re.findall(pattern, line)`
- `print(r)`
- `# ['JAVA', 'PHP', 'Python', 'C', 'Perl', 'SQL']`



# 正则示例：search 第一个匹配

```
import re

line = "Java PHP Python C++ Perl SQL"
pattern = r'[P|p]ython'
if re.search(pattern, line):
    print("exist")
else:
    print("don't exist")
```

**找出第1个匹配的字符串，  
并且提供了匹配串的位置**



# 正则示例：替换 sub

---

```
import re
line = "JAVA PHP Python C++ Perl SQL"
pattern = r'\s+'
new = ','
r = re.sub(pattern, new ,line)
print(r)
# JAVA,PHP,Python,C++,Perl,SQL
```

**把字符串中的空格替换为逗号**



# 正则示例：切分 split

---

```
import re

line = "Java PHP Python C++ Perl    SQL"
pattern = r'\s+'
r = re.split(pattern, line)
print(r)

# ['Java', 'PHP', 'Python', 'C++', 'Perl', 'SQL']
```

使用正则表达式切分字符串，  
这里使用空白来切分



# 正则示例：修饰符

---

```
import re

line = "Java Python PHP Python SQL python Python
PYTHON"
pattern = r'python'
r = re.findall(pattern, line, re.I)
print(r)

# ['Python', 'Python', 'python', 'Python', 'PYTHON']
```

修饰符 **re.I**, 使匹配对大小写不敏感





# 正则示例：research的返回值

```
import re

line = 'order date: 31-08-2019 delivery date: 15-09-2019'
pattern = r'\d{1,2}-\d{1,2}-\d{4}'
r = re.search(pattern, line)
if (r):
    print(r.group())           # 31-08-2019
    print(r.span())           # (12, 22)
    print(r.start(), r.end()) # 12 22
```

**返回第1个成功的匹配**





# 正则示例：获取所有的匹配串

## finditer

```
import re

line = 'order date: 31-08-2019 delivery date: 15-09-2019'
pattern = r'\d{1,2}-\d{1,2}-\d{4}'
rs = re.finditer(pattern, line)
for r in rs:
    print(r.group())
    print(r.span())
    print(r.start(), r.end())
```

找出所有的匹配串及其位置信息

```
31-08-2019
(12, 22)
12 22
15-09-2019
(39, 49)
39 49
```

找到两个匹配





# 正则示例：分组-重新排列日期顺序

```
import re

line = '15/09/2018 18:17'

pattern = r'(\d{2})/(\d{2})/(\d{4}).*(\d{2}):(\d{2})'

new = r'\3-\2-\1 \4:\5'

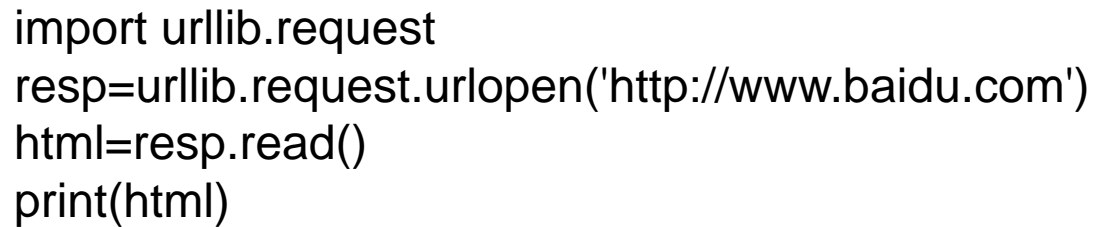
line = re.sub(pattern, new, line)

print(line)
# 2018-09-15 18:17
```

**分组：使用小括号**

**捕获分组**





```
b'<!DOCTYPE html><!--STATUS OK-->\n\n\n    <html><head><meta http-equiv="Content-Type" content="text/  
html; charset=utf-8"><meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"><meta content="always"  
name="referrer"><meta name="theme-color" content="#2932e1"><meta name="description"  
content="\xe5\x85\xa8\xe7\x90\x83\xe6\x9c\x80\xe5\xa4\xa7\xe7\x9a\x84\xe4\xb8\xad\xe6\x96\x87\xe6\x90\x9c  
\xe7\xb4\xa2\xe5\xbc\x95\xe6\x93\x8e\xe3\x80\x81\xe8\x87\xb4\xe5\x8a\x9b\xe4\xba\x8e\xe8\xae\xa9\xe7\xbd  
\x91\xe6\xb0\x91\xe6\x9b\xb4\xe4\xbe\xbf\xe6\x8d\xb7\xe5\x9c\xb0\xe8\x8e\xb7\xe5\x8f\x96\xe4\xbf  
\xa1\xe6\x81\xaf\xef\xbc\x8c\xe6\x89\xbe\xe5\x88\xb0\xe6\x89\x80\xe6\xb1\x82\xe3\x80\x82\xe7\x99\xbe\xe5\xba  
\xa6\xe8\xb6\x85\xe8\xbf\x87\xe5\x8d\x83\xe4\xba\xbf\xe7\x9a\x84\xe4\xb8\xad\xe6\x96\x87\xe7\xbd  
\x91\xe9\xa1\xb5\xe6\x95\xb0\xe6\x8d\xae\xe5\xba\x93\xef\xbc\x8c\xe5\x8f\xaf\xe4\xbb\xa5\xe7\x9e\xac  
\xe9\x97\xb4\xe6\x89\xbe\xe5\x88\xb0\xe7\x9b\xb8\xe5\x85\xb3\xe7\x9a\x84\xe6\x90\x9c\xe7\xb4\xa2\xe7\xbb  
\x93\xe6\x9e\x9c\xe3\x80\x82"><link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" /><link
```



# 编译模式 re.compile

```
import re

line = "Java Python PHP Python SQL python Python PYTHON"
pattern = r'python'
r = re.findall(pattern, line, re.I)
print(r)
```

```
import re

line = "Java Python PHP Python SQL python Python PYTHON"
pattern = re.compile(r'python', re.I)
r = pattern.findall(line)
print(r)
```

- Python使用正则表达式时，re模块内部会完成两件事情：编译正则表达式，如果正则表达式的字符串本身不合法，则会报错；用编译后的正则表达式去匹配字符串。
- 如果一个正则表达式要重复使用很多次，出于效率的考虑，建议编译该正则表达式，接下来重复使用时就不需要再次编译，直接匹配。

两段代码  
功能相同

适用于重复使用多次





# 函数re.compile参数flag的可选值

参 数	英文全称	说 明
re.I	IGNORECASE	忽略大小写
re.M	MULTILINE	多行模式，改变 “^” 和 “\$” 的行为
re.S	DOTALL	点任意匹配模式，改变 “.” 的行为
re.L	LOCALE	使预定字符类取决于当前区域设置
re.U	UNICODE	使预定字符类取决于Unicode定义的字符属性
re.X	VERBOSE	详细模式。正则表达式可以是多行，忽略空白字符，并可加入注释



# 元字符

## 数量词 (用在字符或(...)之后)

*	匹配前一个字符0或无限次。	abc*	ab abccc
+	匹配前一个字符1次或无限次。	abc+	abc abccc
?	匹配前一个字符0次或1次。	abc?	ab abc
{m}	匹配前一个字符m次。	ab{2}c	abbc
{m,n}	匹配前一个字符m至n次。 m和n可以省略：若省略m，则匹配0至n次；若省略n，则匹配m至无限次。	ab{1,2}c	abc abbc
*? +? ?? {m,n}?	使 * + ? {m,n} 变成非贪婪模式。	示例将在下文中介绍。	



# 元字符

## 边界匹配 ( 不消耗待匹配字符串中的字符 )

<code>^</code>	匹配字符串开头。 在多行模式中匹配每一行的开头。	<code>^abc</code>	<code>abc</code>
<code>\$</code>	匹配字符串末尾。 在多行模式中匹配每一行的末尾。	<code>abc\$</code>	<code>abc</code>
<code>\A</code>	仅匹配字符串开头。	<code>\Aabc</code>	<code>abc</code>
<code>\Z</code>	仅匹配字符串末尾。	<code>abc\Z</code>	<code>abc</code>
<code>\b</code>	匹配\w和\W之间。	<code>a\b!bc</code>	<code>a!bc</code>
<code>\B</code>	<code>[^\b]</code>	<code>a\Bbc</code>	<code>abc</code>



# 元字符

逻辑、分组			
	代表左右表达式任意匹配一个。 它总是先尝试匹配左边的表达式，一旦成功匹配则跳过匹配右边的表达式。 如果 没有被包括在()中，则它的范围是整个正则表达式。	abc def	abc def
(...)	被括起来的表达式将作为分组，从表达式左边开始每遇到一个分组的左括号'('，编号+1。 另外，分组表达式作为一个整体，可以后接数量词。表达式中的 仅在该组中有效。	(abc){2} a(123 456)c	abccabc a456c
(?P<name>...)	分组，除了原有的编号外再指定一个额外的别名。	(?P<id>abc){2}	abccabc
\<number>	引用编号为<number>的分组匹配到的字符串。	(\d)abc\1	1abc1 5abc5
(?P=name)	引用别名为<name>的分组匹配到的字符串。	(?P<id>\d)abc(?P=id)	1abc1 5abc5





# 元字符

特殊构造 ( 不作为分组 )			
(?:...)	(...)的不分组版本, 用于使用' '或后接数量词。	(?:abc){2}	abccabc
(?iLmsux)	iLmsux的每个字符代表一个匹配模式, 只能用在正则表达式的开头, 可选多个。匹配模式将在下文中介绍。	(?i)abc	AbC
(?#...)	#后的内容将作为注释被忽略。	abc(?#comment)123	abc123
(?=...)	之后的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	a(=?\d)	后面是数字的a
(?!...)	之后的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	a(?!\d)	后面不是数字的a
(?<=...)	之前的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	(?<=\d)a	前面是数字的a
(?<!...)	之前的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	(?<!\d)a	前面不是数字的a
(?(id/name) yes-pattern  no-pattern)	如果编号为id/别名为name的组匹配到字符, 则需要匹配yes-pattern, 否则需要匹配no-pattern。  no-pattern可以省略。	(\d)abc(?:\d abc)	1abc2 abcabc <a href="http://www.cnblogs.com/huxi">http://www.cnblogs.com/huxi</a>





# 文本数据分析及其任务





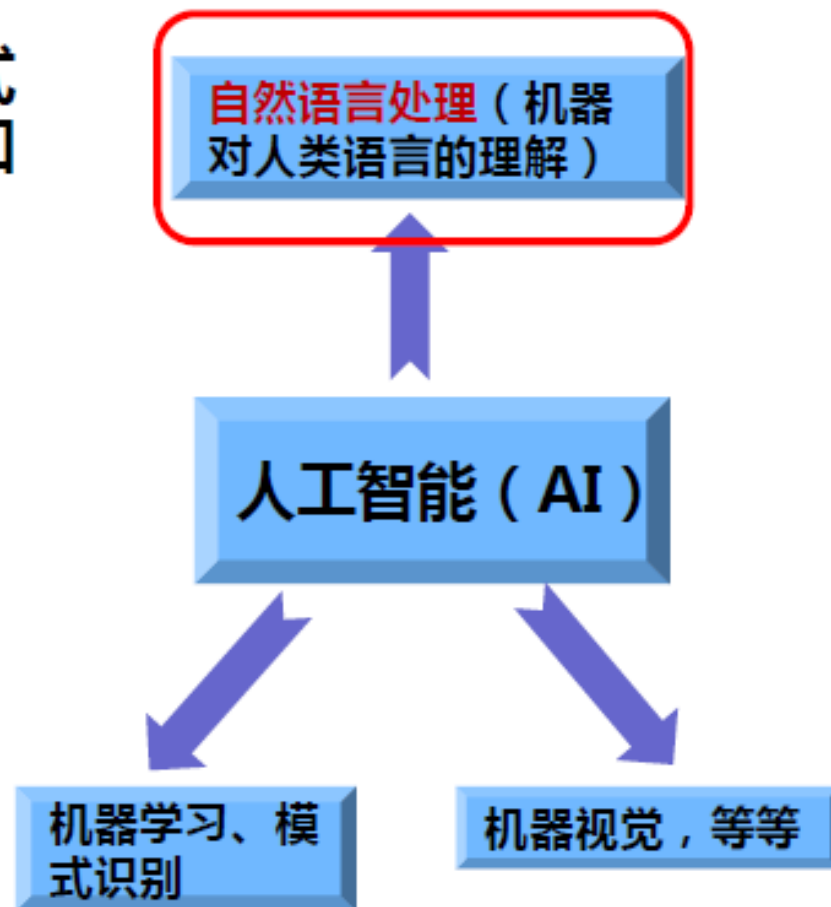
# 自然语言处理 (Natural Language Processing, NLP)

- 自然语言处理是通过建立形式化的计算模型来分析、理解和处理自然语言

- 什么是自然语言

- 其它术语

- 计算语言学(Computational Linguistics)
  - 自然语言理解(Natural Language Understanding)
  - 人类语言技术(Human Language Technology)





# 词性标注

词类： 对一种语言全部词汇的语法分类。  
part of speech (pos)    word class

词性标注： 对文本中兼类词的实际词类归属进行判定。

例：

- 把这篇报道编辑一下

把/q-p-v-n 这/r 篇/q 报道/v-n 编辑/v-n 一/m-c 下/f-q-v

- Time flies like an arrow.

Time/n-v flies/v-n like/p-v an/Det arrow/n

c	连词
Det	冠词
f	方位词
m	数词
n	名词
p	介词
q	量词
r	代词
v	动词



# 词义标注

---

- 异性异义型
  - book:n, 书
  - book:v, 预定
- 同性异义型
  - bank:n, 银行
  - bank:n, 河岸
- 同性近义型
  - 材料: 文字材料
  - 材料: 建筑材料
  - 材料: 某人适合某种工作



# 未登录词

---

- |                |                     |
|----------------|---------------------|
| 1. 汉族人名、地名     | 雪村 老张 中关村           |
| 2. 外族人名、地名     | 横路静二 突尼斯            |
| 3. 中外组织机构单位名称  | 联合国教科文组织            |
| 4. 商品品牌名       | 非常可乐 苹果iPad         |
| 5. 专业术语        | 有限状态自动机 三分球         |
| 6. 新词语         | 秒杀 蚁族 羊羔体           |
| 7. 缩略语         | 人影办 两会 北医三院         |
| 8. 汉语重叠形式、离合词等 | 高高兴兴 幽了他一默          |
| 9. 含数字，非汉字字符的词 | 2014年3月3日 IC卡 D座 T台 |





# 未登录词

---

## □ 未登录词识别困难

- 未登录词没有明确边界
- 许多未登录词的构成单元本身都可以独立成词

## □ 较成熟

- -中国人名、译名
- -中国地名

## □ 较困难

- -商标字号
- -机构名

## □ 很困难

- -专业术语
- -缩略语
- -新词语





# 共指分析

---

- 倪润峰从著名的电器制造公司长虹电子集团有限公司董事长的位置上退休了，赵勇将接替他。
  - 他？





# 【中文】分词

---

输入：字符串

**学生人数多又能保证质量的才是好学校。**

输出：词串

学 生 人 数 多 又 能 保 证 质 量 的 才 是 好 学 校 。

汉语的自然书面文本词与词之间无空格分开，因此，在汉语书面语的处理中（比如词频统计、句子结构分析、语义理解等），首先碰到的就是词的切分问题。

Word Segmentation





# 【中文】分词

- 交集型歧义

例1：张店区大学生不看重大城市的户口本

张店区 大学生 不 看 重大 城市 的 户口本  
张店区 大学生 不 看重 大 城市 的 户口本

- 组合型歧义

例2：你认为学生会听老师的吗

你 认为 学生会 听 老师 的 吗  
你 认为 学生会 听 老师 的 吗

- 混合型歧义

例3：只有雷人才能吸引人

只有 雷人 才能 吸引 人  
只有 雷人才 能 吸引 人  
只有 雷人 才 能 吸引 人

交集型歧义：组合型歧义 = 1: 22<sup>[1]</sup>      语料规模：17,547字

[1] 刘挺、王开铸，1998，关于歧义字段切分的思考与实验。《中文信息学报》第2期，63-64页。





# 【中文】分词

---

- FMM（正向最大匹配）
  - 从左到右，逐步去掉右部（底部）的字进行新一轮匹配
- BMM（逆向最大匹配）
  - 从右到左，逐步去掉左部（底部）的字进行新一轮匹配
- 长词优先原则



# 【中文】分词：示例“研究生命的起源”（正向）

- 字典：

研究

研究生

生命

命

的

起源

命的起源

命的起

命的

命 #第二个词匹配成功，一个单字

的起源

的起

的 #第三个词匹配成功

- 最大匹配字数 = 5

- 匹配过程：

研究生命的

研究生命

研究生 #第一个词匹配成功

起源 #第四个词匹配成功

那么正向最大匹配的结果就是

研究生 命 的 起源





# 【中文】分词：示例“研究生命的起源”（逆向）

- 字典：

研究

研究生

生命

命的

的

起源

研究生命的

究生命的

生命的

命的

的 #第二个词匹配成功

- 最大匹配字数 = 5

研究生命

究生命

生命 #第三个词匹配成功

- 匹配过程：

生命的起源

研究 #第四个词匹配成功

命的起源

的起源

所以逆向最大匹配后的结果为

起源 #第一个词匹配成功

研究 生命 的 起源





# 【中文】分词

- 因为中文比较复杂以及中文的特殊性，逆向最大匹配大多数时候往往会比正向要准确。
- 分词算法设计中的几个基本原则
  - 颗粒度越大越好：“公安局长”比“公安”+“局长”好
  - 切分结果中非词典词越少越好，单字字典词数越少越好
    - “技术和服务”可以分为“技术”、“和”、“服务”，或者“技术”、“和服”、“务”
  - 总体词数越少越好



# 【中文】分词

---

- 最大匹配法的问题

- **存在分词错误** → 增加知识，局部修改

- **无法发现分词歧义** → 从单向最大匹配改为双向最大匹配

正向最大匹配和逆向最大匹配结果不同，意味着存在分词歧义。

FMM 有意/ 见/ 分歧/

BMM 有/ 意见/ 分歧/





# 【中文】分词的预处理

---

- 标点符号，如书名号、引号等
- 特殊标志字符串，如数字、拉丁字母
- 闭锁词，如连词、助词、叹词，了、和、的、与等
- 停用词，如是、太等





# 【中文】分词工具：jieba

```
# encoding=utf-8
```

```
import jieba
```

```
seg_list = jieba.cut("我来到北京大学信息管理系", cut_all=False)
```

```
print("Default Mode: " + "/ ".join(seg_list)) # 精确模式
```

我/ 来到/ 北京大学/ 信息管理/ 系

```
seg_list = jieba.cut("我来到北京大学信息管理系", cut_all=True)
```

```
print("Full Mode: " + "/ ".join(seg_list)) # 全模式
```

我/ 来到/ 北京/ 北京大学/ 大学/ 信息/ 信息管理/ 管理/ 管理系

```
Default Mode: seg_list = jieba.cut("他来到了中关村图书大厦") # 默认是精确模式
```

```
print(", ".join(seg_list))
```

```
seg_list = jieba.cut_for_search("李彦宏本科毕业北京大学信息管理系，后在美国布法罗纽约州立大学深造")  
# 搜索引擎模式
```

```
print(", ".join(seg_list))
```





# 【中文】简繁转换

- (1) 明成皇后，她是一个世纪前北韩王朝的最后<sub>后</sub>一位皇后<sub>后</sub>。  
明成皇后，她是一個世紀前北韓王朝的最後<sub>後</sub>一位皇后<sub>后</sub>。
- (2) 负离子陶瓷烫<sub>发</sub>机，内置负离子<sub>发</sub>射器。  
負離子陶瓷燙<sub>髮</sub>機，內置負離子<sub>發</sub>射器。
- (3) 每个战<sub>斗</sub>单位只有一<sub>斗</sub>米。  
每個戰<sub>鬥</sub>單位只有一<sub>斗</sub>米。



# 【中文】文语转换

---

1. 为达到赢球的[的][目的]，一定要注意比赛时的情绪[调]动与心理[调]节
2. 他们村有三百多人[种]树
3. 他喜欢[展]览馆门口播放的“[小]老鼠[上]灯台”这类欢快的儿歌
4. 这些大学生为什么不[看]重[大]城市户口
5. 这些大学生为什么不[看]重[大]赛事预告
6. 可敬[的]哥[争]分夺秒送病人 (《北京晚报》2000/02/20)
7. 我国首艘航母临近服役[84]年[航]母梦成真 (新浪网新闻标题 2012-09-21)

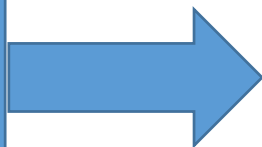
好难 ☹️





# 信息抽取

2010年4月14日7时49分，青海省玉树藏族自治州玉树县发生7.1级地震，此后余震不断，地震造成大量人员伤亡和房屋倒塌。截至2010年4月25日下午17时，玉树地震已造成2220人遇难，失踪70人。



字段	值
事件	地震
时间	2010年4月14日7时49分
地点	青海省玉树藏族自治州玉树县
死亡人数	2220人
失踪人数	70
受伤人数	-



# 信息抽取

- 命名实体识别

We showed that interleukin-1 IL-1 and IL-2 receptor alpha gene ..

Protein

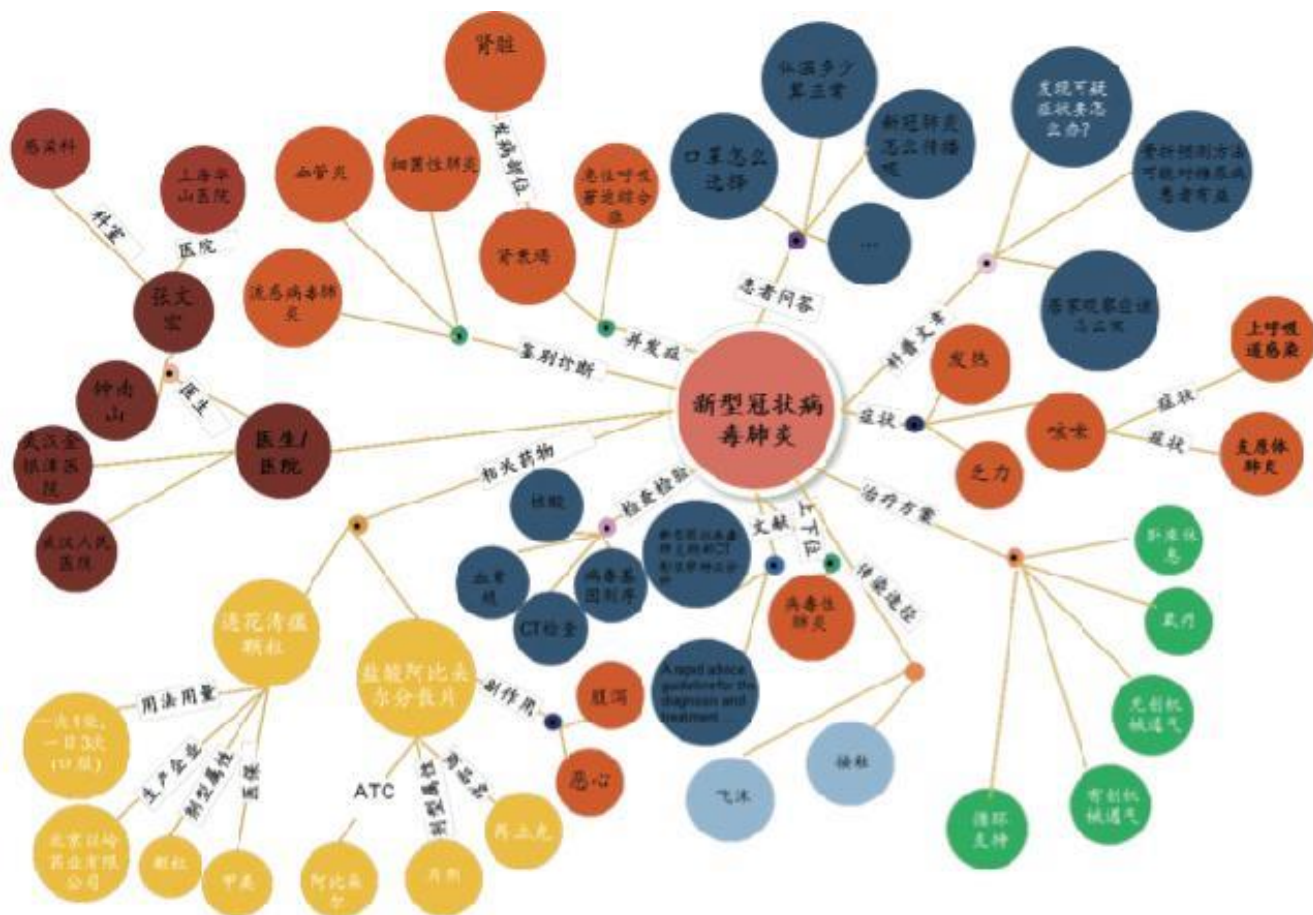
DNA

- 实体消歧 (named entity recognition, NER)
  - 命名实体的歧义指的是一个实体指称项可对应到多个真实世界实体，确定一个实体指称项所指向的真实世界实体，这就是命名实体消歧。



# 信息抽取与知识图谱

- 知识图谱
- 如何构建?
  - 实体识别
  - 实体消歧
  - 关系抽取
  - .....





# 自动文摘

---

- 抽取式摘要 (extraction)
  - 假设：一篇文档的核心思想可以用文档中的某一句或几句话来概括
  - 任务就变成了找到文档中最重要的几句话，也就是一个排序问题
- 生成式摘要 (abstraction)
  - 要求：理解文档、可读性强、简练总结





# 信息检索

- 谷歌、百度
- 基于语义的搜索
- 检索中的自动摘要
- 检索优化
  - Query correction
  - Query expansion
  - Query suggestion



bing

WEB IMAGES VIDEOS MAPS NEWS MORE

query optimitiom

1,650,000 RESULTS Any time ▾

Including results for *query optimization*.  
Do you want results only for query optimitiom?


[Query optimization - Wikipedia, the free encyclopedia](https://en.wikipedia.org/wiki/Query_optimization)  
en.wikipedia.org/wiki/Query\_optimization ▾  
Query optimization is a function of many relational database management systems. The query optimizer attempts to determine the most efficient way to execute a given ...

[How To: Optimize SQL Queries - MSDN – the Microsoft Developer ...](https://msdn.microsoft.com/en-us/library/ff650689)  
msdn.microsoft.com/en-us/library/ff650689 ▾  
If the Physical operation in the query step details is in red, then it indicates that the query optimizer has chosen a less efficient query plan.

[Query Optimization - SQL Query Optimizer](https://queryoptimization.com)  
queryoptimization.com  
What is SQL? SQL (Structured Query Language) is a universal language for reading, writing, updating, deleting and managing data form a relational database.







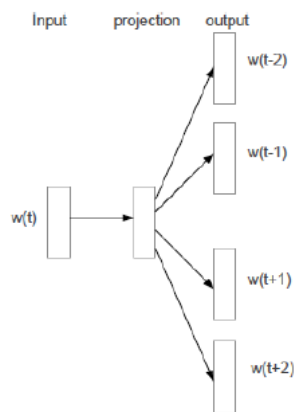
# 文本的表示学习 (representation learning)

- 词嵌入 (word embedding)

- 例如 Word2vec

$$\vec{King} - \vec{Man} + \vec{Woman} = \vec{Queen}$$

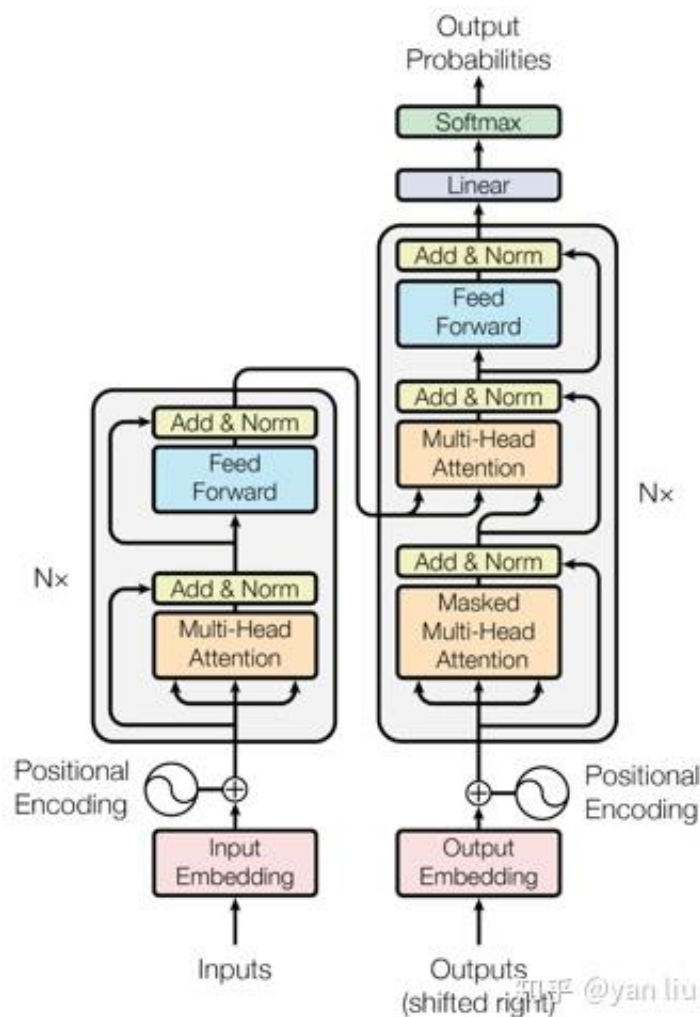
- Goal: represent each word  $i$  with a vector  $\vec{v}_i \in R^d$  by training from a sequence  $(w_1, w_2, \dots, w_T)$
- Distributional hypothesis (John Rupert Firth): *You know a word by the company it keeps*
- Skip-gram: learning word representations by predicting the nearby words



$$p(w_O | w_I) = \frac{\exp(v'_{w_O} \top v_{w_I})}{\sum_{w=1}^W \exp(v'_w \top v_{w_I})}$$

# 文本的表示学习 (representation learning)

- 预训练模型 (pre-trained models)
  - BERT (Bidirectional Encoder Representations from Transformers) : 通过在海量的语料的基础上运行自监督学习方法为单词学习一个好的特征表示
    - 预训练1: Masked Language Model (在训练的时候随即从输入预料上mask掉一些单词, 然后通过上下文预测该单词)
    - 预训练2: Next Sentence Prediction (判断句子B是否是句子A的下文)





# 机器翻译

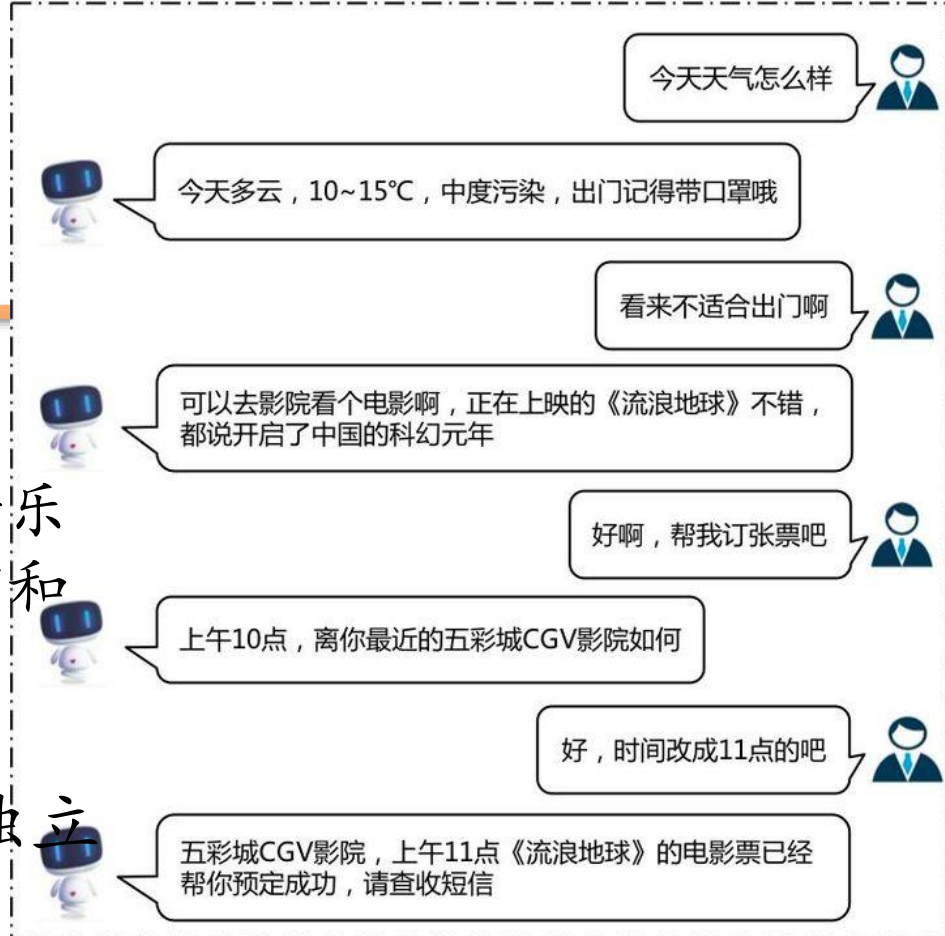
---

- 目标是研制能把一种自然语言翻译成另外一种自然语言的计算机软件系统
- 相关研究始于四十年代末
  - 计算机诞生不久
- 目前市场上有不少翻译产品
  - 已经具有较强实用价值，例如谷歌翻译等



# 人机对话

- 语音助手
  - 内容搜索、信息查询、音乐播放、闹铃设定以及餐馆和票务的预定等功能
  - 百度小度、苹果Siri
- 智能音箱是为对话系统独立设计的音箱产品
  - 家居环境下家电设备的控制
  - 百度小度音箱和小度在家、亚马逊Echo、Google Home、阿里天猫精灵、小米小爱等
- 闲聊软件主要是借助情感计算技术和用户进行情感交流





# 文本数据分析的难点

---

- 原因：自然语言中有大量的歧义现象
- 本质原因：知识体系的缺乏
  - 自然语言的理解不仅和语言本身的规律有关，还和语言之外的知识（例如常识）有关
  - 语言处理涉及的常是海量知识，知识库的建造维护难以进行
  - 场景/背景的建立问题





# 文本数据分析的难点

---

- 常见对策一：建立“知识”
  - 规则方法(rule-based methods)
    - 通过语言学知识编写规则
    - 通过规则引入知识
  - 经验方法(empirical methods)
    - 训练数据+机器学习
    - 通过训练数据引入知识，通过机器学习消歧
  - 规则和经验方法的结合
  - 交互式处理
    - 人机互助进行处理



# 文本数据分析的难点

---

- 常见对策二：减少“未知知识”
  - 限定语言
  - 限定领域
    - 限定处理文本的领域
  - 限定任务
  - 限定复杂度
    - 限定语言的词汇和句法，降低复杂度





# NLP的前沿研究

---

- 基于网络大数据的自然语言理解成为新热点
  - 信息提取、自动文摘、情感分析、观点挖掘、主题跟踪、自动知识库抽取等
- 大规模自然语言处理模型、算法
  - 多领域自适应学习 + 自然语言处理
  - 高速并行计算 + 大规模自然语言处理



# 文本可视化分析





# 文本可视化

---

- 文本可视化技术综合了文本分析、数据挖掘、数据可视化、计算机图形学、人机交互、认知科学等学科的理论和方法，为人们理解复杂的文本内容、结构和内在的规律等信息的有效手段。
- 文本可视化技术将文本中复杂的或者难以通过文字表达的内容和规律以视觉符号的形式表达出来，
- 同时向人们提供与视觉信息进行快速交互的功能，使人们能够利用与生俱来的视觉感知的并行化处理能力快速获取大数据中所蕴含的关键信息。



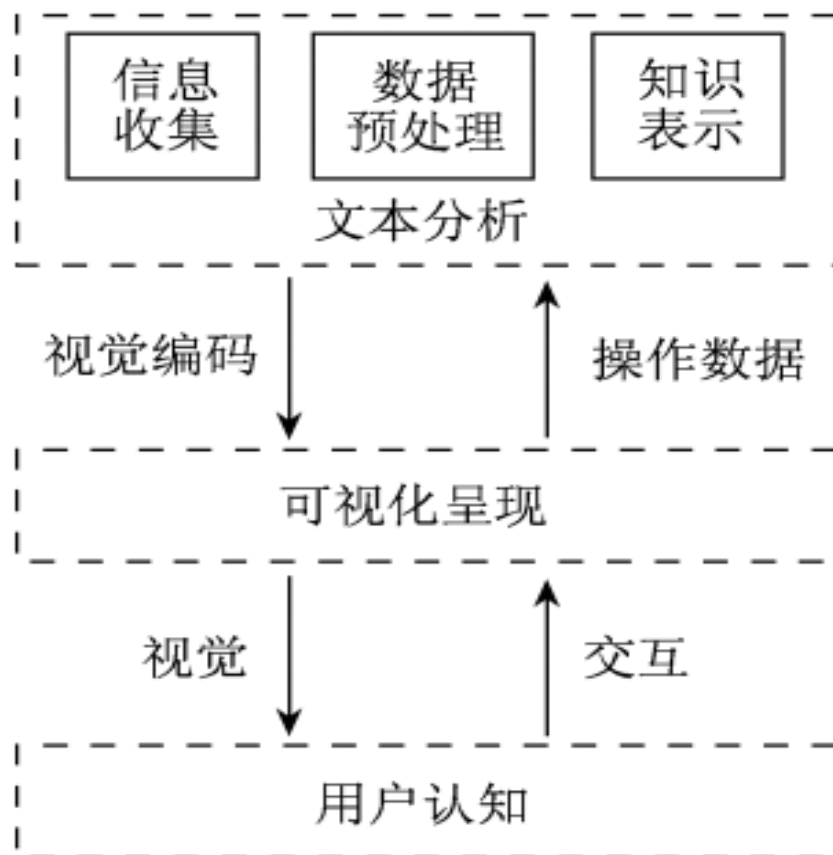
# 文本可视化

---

- 文本可视化技术涵盖了信息收集、数据预处理、知识表示、视觉呈现和交互等过程。
- 其中，数据挖掘和自然语言处理等技术充分发挥计算机的自动处理能力，将无结构的文本信息自动转换为可视的有结构信息。
- 而可视化呈现使人类视觉认知、关联、推理的能力得到充分的发挥。



# 文本可视化的基本框架





# 政府工作报告词云绘制

---

- 来源：
  - <http://www.china-cer.com.cn/guwen/2021021911402.html>
  - <http://www.gov.cn/guowuyuan/zfgzbg.htm>
- 数据预处理：
  - 分词
  - 去停用词
- 设定参数进行绘制：
  - 词云形状
  - 字体
  - 词频阈值
  - 最大显示次数
  - .....



# 政府工作报告词云绘制

- 安装工具包:

```
>>> pip install wordcloud
```

```
import jieba
from wordcloud import WordCloud
from skimage import io
import os
import numpy as np
```

```
def chinese_jieba(txt):
    wordlist_jieba = jieba.cut(txt) # 将文本分割, 返回列表
    txt_jieba = " ".join(wordlist_jieba) # 将列表拼接为以空格为间断的字符串
    return txt_jieba
```



# 政府工作报告词云绘制

- 工具包:

```
stopwords = {'这些', '那些', '因为', '所以', '和', '的'}
cloud_mask = np.array(PIL.Image.open('air.jpg'))
with open('2021政府工作报告.txt', encoding='utf8') as fp:
    txt = fp.read()
    cutted_text = chinese_jieba(txt)

# print(txt)
wordcloud = WordCloud(font_path = r'C:\Windows\Fonts\STKAITI.TTF', # 字体
                      background_color = 'white', # 背景色
                      max_words = 80, # 最大显示单词数
                      max_font_size = 80, # 频率最大单词字体大小
                      mask = cloud_mask, # 词云形状mask
                      stopwords = stopwords # 过滤停用词
                      ).generate(cutted_text)
image = wordcloud.to_image()
image.show()
```







# 政府工作报告词云绘制

- 效果展示





# 历年政府工作报告词云



2016 年



2017 年



2018 年



2019 年



# 布置第四次作业

---





谢谢！

