第11课　科学计算
图片处理

Pandas
Pillow

pandas

- pandas（Python Data Analysis Library）是基于numpy的数据分析模块，提供了大量标准数据模型和高效操作大型数据集所需要的工具。
- 有人说，**pandas是使得Python能够成为高效且强大的数据分析环境的重要因素之一**。
- pandas主要提供了3种数据结构：
  - **1）Series，带标签的一维数组**
  - **2）DataFrame，带标签且大小可变的二维表格结构（重点）**
  - 3）Panel，带标签且大小可变的三维数组（自学）

Series

- Series是一个一维的类似字典的对象，包含一个数组的数据（任何NumPy的数据类型）和一个与数组关联的数据标签 (索引)。
- Seriers的交互式显示的字符串表示形式是索引在左边，值在右边。

如果不给数据指定索引，创建默认索引包含整数0到 N-1 （N是数据的长度）。

```
import pandas as pd
x=pd.Series([1,3,5,7])
print(x)
```

```
0    1
1    3
2    5
3    7
dtype: int64
```

```
print(x.values)
[1 3 5 7]

print(x.index)
RangeIndex(start=0, stop=4,
step=1)
```

Series

通常，创建一个**指定的索引**来确定Series的每一个数据点：

```
import pandas as pd
x=pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
print(x)
d    4
b    7
a   -5
c    3
dtype: int64

print(x.values)
[ 4  7 -5  3]

print(x.index)
Index(['d', 'b', 'a', 'c'], dtype='object')
```

Series

Series对象可以理解为一个字典。

互相转换的机制：
将series对象的index作为字典的keys，对应的值作为字典的value。

```
import pandas as pd
x=pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
print(dict(x))
{'d': 4, 'b': 7, 'a': -5, 'c': 3}

my_dict = {'a': 3, 'b': 7, 'c': 1, 'd': 5}
series_obj = pd.Series(my_dict)
print(series_obj)
a    3
b    7
c    1
d    5
dtype: int64
```

Series

使用索引检索数据

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e’])

print(s['a'])
1

print(s[['a','c','d']])
a    1
c    3
d    4
dtype: int64

print(s['f’])
KeyError: 'f'
```

Series

可以：（1）通过一个布尔数组过滤，（2）纯量乘法，（3）使用数学函数

```
import pandas as pd
import numpy as np

x=pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
print(x[x>0])
print(x*3)
print(np.exp(x))
```

```
d    4
b    7
c    3
dtype: int64
```

```
d    12
b    21
a   -15
c     9
dtype: int64
```

```
d      54.598150
b    1096.633158
a       0.006738
c      20.085537
dtype: float64
```

Series

在许多应用中Series的一个重要功能是在**算术运算**中它会自动对齐不同索引的数据：

```
x=pd.Series([1, 2, 3, 4], index=['d', 'b', 'a', 'c'])
y=pd.Series([4, 3, 2, 1], index=['d', 'b', 'f', 'a'])

print(x+y)
```

```
d    1
b    2
a    3
c    4
dtype: int64
```

+

```
d    4
b    3
f    2
a    1
dtype: int64
```

=

```
a    4.0
b    5.0
c    NaN
d    5.0
f    NaN
dtype: float64
```

DataFrame

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = pd.DataFrame(data)
print(frame)
```

```
    state  year  pop
0    Ohio  2000  1.5
1    Ohio  2001  1.7
2    Ohio  2002  3.6
3  Nevada  2001  2.4
4  Nevada  2002  2.9
```

产生的DataFrame和Series一样，它的索引会自动分配，并且对列进行了排序

DataFrame是类似表格的数据结构!

DataFrame

如果设定了列的顺序，DataFrame的列将会精确的按照所传递的顺序排列（也可以加入索引）：

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}

frame =pd.DataFrame(data, columns=['year', 'state', 'pop', 'others'],
index=['one', 'two', 'three', 'four', 'five' ])

print(frame)
```

```
       year   state  pop others
one    2000    Ohio  1.5    NaN
two    2001    Ohio  1.7    NaN
three  2002    Ohio  3.6    NaN
four   2001  Nevada  2.4    NaN
five   2002  Nevada  2.9    NaN
```

DataFrame

DataFrame与字典的转换

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],'year':
[2000, 2001, 2002, 2001, 2002], 'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = pd.DataFrame(data, index = [2,3,4,5,6])
print(frame)
```

	state	year	pop
2	Ohio	2000	1.5
3	Ohio	2001	1.7
4	Ohio	2002	3.6
5	Nevada	2001	2.4
6	Nevada	2002	2.9

```
print(dict(frame['year']))
```

```
{2: 2000, 3: 2001, 4: 2002, 5: 2001, 6: 2002}
```

只考虑一列的话，相当于是将Series对象转换成dict

DataFrame

DataFrame与字典的转换

```
   state  year  pop
2   Ohio  2000  1.5
3   Ohio  2001  1.7
4   Ohio  2002  3.6
5 Nevada  2001  2.4
6 Nevada  2002  2.9
```

```
my_dict=dict(frame[['pop', 'year']])
print(my_dict)
```

```
{'pop': 2    1.5
        3    1.7
        4    3.6
        5    2.4
        6    2.9
Name: pop, dtype: float64,

'year': 2    2000
        3    2001
        4    2002
        5    2001
        6    2002
Name: year, dtype: int64}
```

相当于一个嵌套的字典

```
print(my_dict['pop'][2])
1.5
```

属性

```
print(my_dict['pop'].name)
pop
print(my_dict['pop'].dtype)
float64
```

DataFrame

也可以从列表中创建DataFrame

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print(df)
```

```
   0
0  1
1  2
2  3
3  4
4  5
```

```
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print(df)
```

```
      Name  Age
0     Alex   10
1      Bob   12
2   Clarke   13
```

```
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'],dtype=float)
print(df)
```

```
      Name   Age
0     Alex  10.0
1      Bob  12.0
2   Clarke  13.0
```

DataFrame

从字典的列表中创建DataFrame

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data, index=['first', 'second'])
print(df)
```

```
            a   b     c
first       1   2   NaN
second      5  10  20.0
```

```
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

df1 = pd.DataFrame(data, index=['first', 'second'],
columns=['a', 'b' ])

df2 = pd.DataFrame(data, index=['first', 'second'],
columns=['a', 'b1'])

print(df1) ? ? ?
print(df2) ? ? ?
```

DataFrame

从字典的列表中创建DataFrame

```
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

df1 = pd.DataFrame(data, index=['first', 'second'],
columns=['a', 'b'])

df2 = pd.DataFrame(data, index=['first', 'second'],
columns=['a', 'b1'])

print(df1)
print(df2)
```

```
        a   b
first   1   2
second  5  10
```

```
        a  b1
first   1 NaN
second  5 NaN
```

DataFrame

添加列

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print(df)
```

```
   one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4
```

```
df['three']=pd.Series([10,20,30],index=['a','b','c'])
print(df)
```

```
   one  two  three
a  1.0    1   10.0
b  2.0    2   20.0
c  3.0    3   30.0
d  NaN    4    NaN
```

```
df['four']=df['one']+df['three']
print(df)
```

```
   one  two  three  four
a  1.0    1   10.0  11.0
b  2.0    2   20.0  22.0
c  3.0    3   30.0  33.0
d  NaN    4    NaN   NaN
```

DataFrame

删除列

```
print(df)
```

```
    one  two  three
a   1.0    1   10.0
b   2.0    2   20.0
c   3.0    3   30.0
d   NaN    4    NaN
```

```
del df['one']
print(df)
```

```
   two  three
a    1   10.0
b    2   20.0
c    3   30.0
d    4    NaN
```

```
df.pop('two')
print(df)
```

```
   three
a   10.0
b   20.0
c   30.0
d    NaN
```

DataFrame

处理完了列，接着轮到行...

按**索引选择**:可以通过将行索引传递给**loc函数**来选择行。

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)

print(df)
```

```
   one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4
```

```
print(df.loc['b'])
```

```
one    2.0
two    2.0
Name: b, dtype: float64
```

DataFrame

添加/删除行

```
import pandas as pd
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b' ])
```

```
print(df)
print(df2)
```

```
   a  b
0  1  2
1  3  4
```

```
   a  b
0  5  6
1  7  8
```

```
df = df.append(df2)
print(df)
```

```
   a  b
0  1  2
1  3  4
0  5  6
1  7  8
```

```
df = df.drop(0)
print(df)
```

```
   a  b
1  3  4
1  7  8
```

DataFrame

迭代-iteritems(): 每列作为键-值对

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(4,3),columns=['col1','col2','col3'])
for key,value in df.iteritems():
   print(key,value)
```

```
col1 0   -0.439642
     1    0.278907
     2   -0.720401
     3   -0.889938
     Name: col1, dtype: float64
col2 0    0.554922
     1    1.485924
     2   -0.012565
     3   -0.432469
     Name: col2, dtype: float64
col3 0    1.039062
     1   -1.474022
     2    2.116732
     3   -0.174312
     Name: col3, dtype: float64
```

DataFrame

迭代-iterrows()：产生每个索引值以及包含每行数据

```
df = pd.DataFrame(np.random.randn(4,3),columns = ['col1','col2','col3'])
for row_index,row in df.iterrows():
   print(row_index,row)
```

```
0 col1   -1.558902
  col2    0.431634
  col3   -0.376704
  Name: 0, dtype: float64
1 col1   -0.409094
  col2    0.560735
  col3    0.114407
  Name: 1, dtype: float64
2 col1    1.029609
  col2    0.338889
  col3   -1.374956
  Name: 2, dtype: float64
3 col1   -0.443590
  col2    1.217335
  col3    0.970800
  Name: 3, dtype: float64
```

DataFrame

排序：

```
import pandas as pd
import numpy as np
unsorted_df =
pd.DataFrame(np.random.randn(10,2),index=[1,4,6,2,3,5,9,8,0,7],columns =
['col2','col1'])
print(unsorted_df)

sorted_df=unsorted_df.sort_index()
print(sorted_df)
```

(降序：`unsorted_df.sort_index(ascending=False)`)

```
       col2      col1
1 -0.169437 -0.818017
4  0.203346  0.576408
6 -0.377205 -2.457092
2  0.827508  1.100445
3  0.137887  0.939092
5  0.103728  0.700505
9  0.192438  1.147378
8 -0.173740 -0.813452
0 -0.582736 -0.237847
7 -0.471654  1.100396
```

```
       col2      col1
0 -0.582736 -0.237847
1 -0.169437 -0.818017
2  0.827508  1.100445
3  0.137887  0.939092
4  0.203346  0.576408
5  0.103728  0.700505
6 -0.377205 -2.457092
7 -0.471654  1.100396
8 -0.173740 -0.813452
9  0.192438  1.147378
```

DataFrame

排序：按列（columns）排序

```
sorted_df=unsorted_df.sort_index(axis=1)
print(sorted_df)
```

	col2	col1
1	-1.415323	-1.204942
4	2.144845	0.223775
6	3.407945	0.237890
2	-1.826759	0.135543
3	-0.116241	0.684461
5	0.529598	-1.730282
9	-0.942271	-0.574648
8	1.399938	0.836752
0	1.430344	-0.593297
7	-1.726375	-1.701347

	col1	col2
1	-1.204942	-1.415323
4	0.223775	2.144845
6	0.237890	3.407945
2	0.135543	-1.826759
3	0.684461	-0.116241
5	-1.730282	0.529598
9	-0.574648	-0.942271
8	0.836752	1.399938
0	-0.593297	1.430344
7	-1.701347	-1.726375

DataFrame

排序：按数值排序sort_values()

```
unsorted_df = pd.DataFrame({'col1':[2,1,1,1],'col2':[1,3,2,4]})
print(unsorted_df)

sorted_df = unsorted_df.sort_values(by='col1')
print(sorted_df)
```

```
   col1  col2
0     2     1
1     1     3
2     1     2
3     1     4
```

```
   col1  col2
1     1     3
2     1     2
3     1     4
0     2     1
```

```
sorted_df = unsorted_df.sort_values(by=['col1','col2'])
print(sorted_df)
```

```
   col1  col2
2     1     2
1     1     3
3     1     4
0     2     1
```

```
kind='mergesort'
     'heapsort'
     'quicksort'
```

DataFrame

小应用：与日期相关、生成日期序列

```
dates = pd.date_range('20200601', periods=7)
print(dates)
```

```
DatetimeIndex(['2020-06-01', '2020-06-02', '2020-06-03', '2020-06-04',
               '2020-06-05', '2020-06-06', '2020-06-07'],
              dtype='datetime64[ns]', freq='D')
```

freq='M’ 月，'D’ 天， ‘W’ ，周，'Y’ 年

```
calendar = pd.DataFrame(np.zeros((7,4)), index=dates,
                        columns=list('ABCD'))
print(calendar)
```

	A	B	C	D
2020-06-01	0.0	0.0	0.0	0.0
2020-06-02	0.0	0.0	0.0	0.0
2020-06-03	0.0	0.0	0.0	0.0
2020-06-04	0.0	0.0	0.0	0.0
2020-06-05	0.0	0.0	0.0	0.0
2020-06-06	0.0	0.0	0.0	0.0
2020-06-07	0.0	0.0	0.0	0.0

DataFrame

小应用：与日期相关、生成日期序列

```
print(calendar)
```

	A	B	C	D
2020-06-01	0.0	0.0	0.0	0.0
2020-06-02	0.0	0.0	0.0	0.0
2020-06-03	0.0	0.0	0.0	0.0
2020-06-04	0.0	0.0	0.0	0.0
2020-06-05	0.0	0.0	0.0	0.0
2020-06-06	0.0	0.0	0.0	0.0
2020-06-07	0.0	0.0	0.0	0.0

对表格转置　`calendar=calendar.T`

	2020-06-01	2020-06-02	2020-06-03	...	2020-06-05	2020-06-06	2020-06-07
A	0.0	0.0	0.0	...	0.0	0.0	0.0
B	0.0	0.0	0.0	...	0.0	0.0	0.0
C	0.0	0.0	0.0	...	0.0	0.0	0.0
D	0.0	0.0	0.0	...	0.0	0.0	0.0

DataFrame

小应用：与日期相关、生成日期序列

```
              A    B    C    D
2020-06-01  0.0  0.0  0.0  0.0
2020-06-02  0.0  0.0  0.0  0.0
2020-06-03  0.0  0.0  0.0  0.0
2020-06-04  0.0  0.0  0.0  0.0
2020-06-05  0.0  0.0  0.0  0.0
2020-06-06  0.0  0.0  0.0  0.0
2020-06-07  0.0  0.0  0.0  0.0
```

```
print(calendar.shape)
#返回表示DataFrame维度的元组(a,b)其中a表示行数，b表示列数。
```

```
(7, 4)
```

```
print(calendar.size)
#返回DataFrame中的元素数。
```

```
28
```

```
print(calendar.head(2))
#返回前n行
```

```
              A    B    C    D
2020-06-01  0.0  0.0  0.0  0.0
2020-06-02  0.0  0.0  0.0  0.0
```

```
print(calendar.tail(2))
#返回最后n行
```

```
              A    B    C    D
2020-06-06  0.0  0.0  0.0  0.0
2020-06-07  0.0  0.0  0.0  0.0
```

DataFrame

应用：描述性统计(Descriptive Statistics)

	Name	Age	Rating
0	Tom	25	4.23
1	James	26	3.24
2	Ricky	25	3.98
3	Vin	23	2.56
4	Steve	30	3.20
5	Smith	29	4.60
6	Jack	23	3.80
7	Lee	34	3.78
8	David	40	2.98
9	Gasper	30	4.80
10	Betina	51	4.10
11	Andres	46	3.65

sum(), mean(), std()

DataFrame

应用：描述性统计(Descriptive Statistics)

```
print(df.sum())   #axis=0
```

```
Name      TomJamesRickyVinSteveSmithJackLeeDavidGasperBe...
Age                                                     382
Rating                                                44.92
dtype: object
```

```
print(df.sum(1)) #axis=1
```

```
0     29.23
1     29.24
2     28.98
3     25.56
4     33.20
5     33.60
6     26.80
7     37.78
8     42.98
9     34.80
10    55.10
11    49.65
dtype: float64
```

```
print(df.mean())
```

```
Age       31.833333
Rating     3.743333
dtype: float64
```

```
print(df.std())
```

```
Age       9.232682
Rating    0.661628
dtype: float64
```

DataFrame

应用：描述性统计(Descriptive Statistics)

总结数据

```
      Name   Age   Rating
0      Tom    25     4.23
1    James    26     3.24
2    Ricky    25     3.98
3      Vin    23     2.56
4    Steve    30     3.20
5    Smith    29     4.60
6     Jack    23     3.80
7      Lee    34     3.78
8    David    40     2.98
9   Gasper    30     4.80
10  Betina    51     4.10
11  Andres    46     3.65
```

```
print(df.describe())
```

```
              Age      Rating
count   12.000000   12.000000
mean    31.833333    3.743333
std      9.232682    0.661628
min     23.000000    2.560000
25%     25.000000    3.230000
50%     29.500000    3.790000
75%     35.500000    4.132500
max     51.000000    4.800000
```

DataFrame

应用：描述性统计(Descriptive Statistics)

总结数据

print(df.describe(include=['number'])) #**number** - 汇总数字列
print(df.describe(include=['object'])) **#object** - 汇总String列

```
                Name
count             12
unique            12
top            Ricky
freq               1
```

print(df.describe(include='all'))#将所有列汇总在一起

```
                Name        Age     Rating
count             12  12.000000  12.000000
unique            12        NaN        NaN
top            Ricky        NaN        NaN
freq               1        NaN        NaN
mean             NaN  31.833333   3.743333
std              NaN   9.232682   0.661628
min              NaN  23.000000   2.560000
25%              NaN  25.000000   3.230000
50%              NaN  29.500000   3.790000
75%              NaN  35.500000   4.132500
max              NaN  51.000000   4.800000
```

DataFrame

功能应用(Function Application)
- 逐表函数:pipe()
- 行或列智能函数:apply ()
- 元素智能函数:applymap ()

逐表函数

可以通过将函数和适当数量的参数作为pipe()参数传递来执行自定义操作，对整个DataFrame执行操作。

例子：为DataFrame中的所有元素添加2。

```
def adder(ele1,ele2):
    return ele1+ele2
```

DataFrame

```
import pandas as pd
import numpy as np

def adder(ele1,ele2):
    return ele1+ele2

df = pd.DataFrame(np.random.randint(5, size=(3,3)),
     columns=['col1','col2','col3' ])
```

```
print(df)
```

	col1	col2	col3
0	3	4	3
1	3	3	0
2	2	3	4

```
print(df.pipe(adder,2))
```

	col1	col2	col3
0	5	6	5
1	5	5	2
2	4	5	6

DataFrame

元素智能函数
对DataFrame的每一个数据进行操作

```
def adder(ele1,ele2):
    return ele1+ele2

df = pd.DataFrame(np.random.randint(5, size=(3,
3)),columns=['col1','col2','col3'])

print(df.applymap(lambda x:x+100))
print(df.pipe(adder,100))
```

```
   col1  col2  col3
0   102   100   101
1   102   102   103
2   103   104   103
```

DataFrame

行或列智能函数
可以使用apply()方法沿DataFrame的轴应用任意函数，该方法与描述性统计方法一样，采用可选的轴参数。 默认情况下，操作按列方式执行，将每列作为类似数组。

```
    col1  col2  col3
0      2     0     0
1      1     4     4
2      1     1     4
print(df.apply(np.mean))
print(df.apply(np.mean,axis=1))
print(df.apply(lambda x: x.max() - x.min()))
```

```
col1    1.333333
col2    1.666667
col3    2.666667
dtype: float64
```

```
0    0.666667
1    3.000000
2    2.000000
dtype: float64
```

```
col1    1
col2    4
col3    4
dtype: int64
```

PIL、pillow

Python Imaging Library (PIL)是python下的图像处理模块，支持多种格式，并提供强大的图像处理功能，可以通过pip进行安装后使用。

PIL模块的官方版本只支持Python2，pillow支持Python3。

from PIL import Image

```
im = Image.open('view.jpg')
im.show()

print(im.format) # JPEG
print(im.size) # (533, 300)
```

基本操作

```
#基本操作
#读取像素值
print(im.getpixel((100,50)))
#(29, 66, 146) 返回格式为（R，G，B）的元组

#设置像素值：指定目标像素的颜色值
im.putpixel((100,50),(128,30,120))

#保存图片文件
im.save('view.jpg’)

#转换图片格式
im.save('view.bmp’)

#图像缩放
im1 = im.resize((100,100))
```

基本操作

旋转图像

```
im2 = im.rotate(90)
```

```
im3 = im.transpose(Image.ROTATE_180)
```

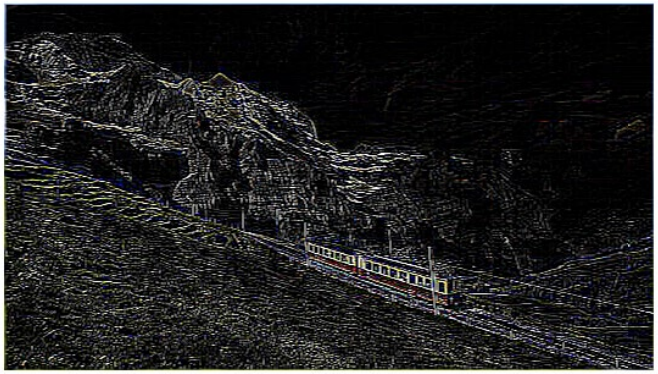
```
im4 = im.transpose(Image.FLIP_LEFT_RIGHT)
```

ImageFilter

```
from PIL import ImageFilter
im5 = im.filter(ImageFilter.DETAIL)
#图像增强
```

```
im6 = im.filter(ImageFilter.BLUR)
#图像模糊
```

```
im7 = im.filter(ImageFilter.FIND_EDGES)
#图像边缘提取
```



图像点运算

```
#调整亮度

im8 = im.point(lambda i:i*1.3)


im9 = im.point(lambda i:i*0.7)
```

```
#调整亮度2
from PIL import ImageEnhance
enh = ImageEnhance.Brightness(im)
enh.enhance(1.3).show()
```

图像点运算

```
#调整亮度2
from PIL import ImageEnhance
enh = ImageEnhance.Brightness(im)
enh.enhance(1.3).show()
```

```
#调整对比度（增强）
enh = ImageEnhance.Contrast(im)
enh.enhance(1.3).show()
```

图像冷暖色调

```
r,g,b = im.split()
#先将彩色图片分离为RGB三个通道，分离后每个通道的大小和原图像一样，但只包含一个颜色分量
#然后改变每个通道的亮度
r = r.point(lambda i:i*1.3)
g = g.point(lambda i:i*0.9)
b = b.point(lambda i:0)
#最后将三个通道组合
im10 = Image.merge(im.mode,(r,g,b))
im10.show()
```

RGB

Pillow应用

- 使用pillow生成验证码图片
- 验证码在网络应用开发中占有重要的地位，广泛应用于用户注册、登录、留言、购物等场合，可以有效地阻止恶意用户频繁地提交非法数据。
- 图片验证码是比较传统的验证码形式，图片中除了经过平移、旋转、错切、缩放等基本变换的字母和数字之外，还有一些随机线条或其他干扰因素。用户需要输入正确的答案才能进行后续的操作。
  - 需要pillow的哪些模块？
  - 还需要什么模块？

Pillow应用

- 面向过程的写法

- 我们需要pillow的哪些模块？
  - ✓ 需要创建图片对象：Image
  - ✓ 需要对图像对象的简单2D绘制： ImageDraw
  - ✓ 需要用什么字体文件格式？ImageFont
  - ✓ 有些验证码比较模糊：ImageFilter

- 我们需要哪些额外的模块？
  - ✓ 一些字符：string
  - ✓ 验证码一般随机生成（包括颜色和字符）：random

```
from PIL import Image, ImageDraw, ImageFont, ImageFilter
import random
import string
```

Pillow应用

- 面向过程的写法
- 先定义一些辅助的常量和函数

```
characters = string.ascii_letters+string.digits

# 随机字母:
def rndChar():
    return random.choice(characters)

# 随机颜色:返回RGB元组
def rndColor():
    return (random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255))
```

- 过程:
  - ✓ 创建图片对象→创建字体对象→创建绘图对象
  - ✓ 绘图→模糊化

Pillow应用

- 面向过程的写法

```
# 创建Image对象:
width = 60 * 4      #这里假设验证码有4个字符
height = 60
image = Image.new('RGB', (width, height), (255, 255, 255))
```

im.mode　　im.size　　颜色

```
# 创建Font对象:
font = ImageFont.truetype('c:/windows/fonts/TIMESBD.TTF', 36)
```

PIL无法定位到字体文件的位置，可以根据操作系统提供绝对路径　　字体大小

```
# 创建Draw对象:
draw = ImageDraw.Draw(image)
```

Pillow应用

- 面向过程的写法

```
# 绘制验证码字符:
for t in range(4):
    draw.text((60 * t + 10, 10), rndChar(), font=font,
fill=rndColor())

# 简单模糊化一下:
image = image.filter(ImageFilter.BLUR)
image.show()
```

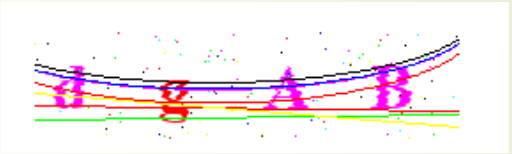
Pillow应用

- 为了安全，有些验证码的字体不那么标准
- 也可以添加一些干扰像素的噪点、线条、弧线

```
imageFinal = Image.new('RGB', (width, height), (255, 255, 255))

pixelsFinal = imageFinal.load()
pixelsTemp = image.load()

#进行简单的像素点运算，像素点位置的变换
for y in range(0, height):
    offset = random.randint(-1,1)
    for x in range(0, width):
        newx = x+offset
        #处理一下边界
        ？？？（实验）
        pixelsFinal[newx,y] = pixelsTemp[x,y]
```

Pillow应用

- 为了安全，有些验证码的字体不那么标准
- 也可以添加一些干扰像素的噪点、线条、弧线



```
draw = ImageDraw.Draw(imageFinal)
#添加干扰噪点像素
for i in range(int(width*height*0.01)):
    draw.point((random.randint(0,width), random.randint(0,height)),
fill=rndColor())

#添加干扰线条
for i in range(4):
    ？？？（实验）
    draw.line(？？？)

#添加干扰弧线
for i in range(4):
    ？？？（实验）
    draw.arc(？？？)
```

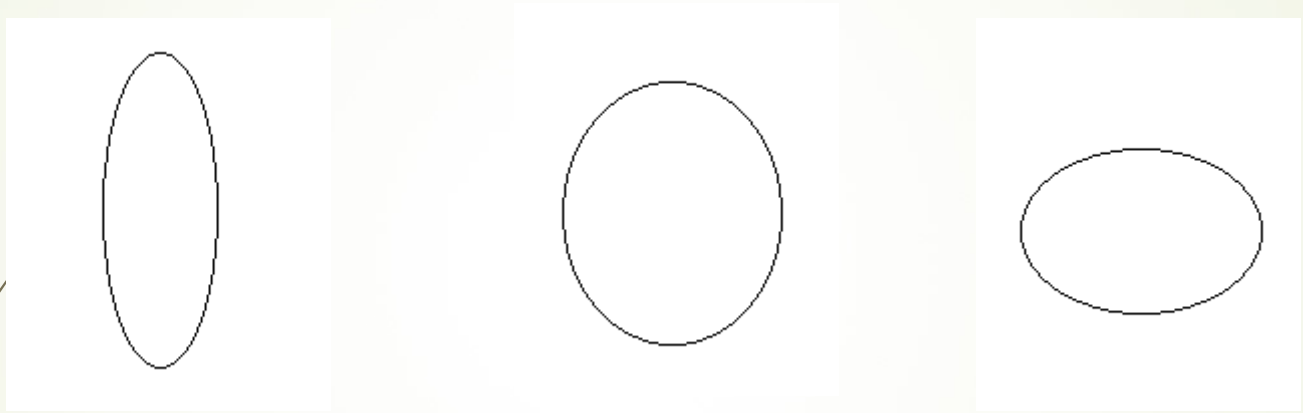
Pillow应用

- 我们刚才看到可以使用ImageDraw模块来绘制图像：直线和弧线
- 还可以绘制矩形，椭圆、多边形和文本等。


```
from PIL import Image,ImageDraw,ImageFont

blank_image = Image.new('RGB',(400,300),'white')

# 在blank_image 图像上绘图
img_draw = ImageDraw.Draw(blank_image)

# 画一个矩形
img_draw.rectangle((70,50,270,200),outline='red',fill='blue')

# 设置字体
fnt = ImageFont.truetype('c:/windows/fonts/TIMESBD.TTF',40) # 修改电脑上的字型,字体可自行下载

# 放上文字信息到图像上
img_draw.text((70,250),'hello world',font=fnt,fill='green')

blank_image.show()
```

Pillow应用

使用pillow计算任意椭圆的中心

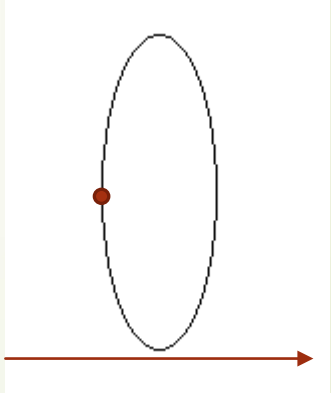


- 只有两种像素点
  - ✓ 背景为白色（255,255,255）
  - ✓ 椭圆为黑色（0,0,0）

- 方法：黑色像素点的搜索，找出椭圆最左端，最右端，最上端，最下端的像素点的xy坐标，然后计算椭圆的中心

Pillow应用

```
def searchLeft(width, height, im):
    for w in range(width): #从左向右扫描
        for h in range(height): #从下向上扫描
            color = im.getpixel((w, h)) #获取图像指定位置的像素颜色
            if color != (255, 255, 255):
                return w #遇到并返回椭圆边界最左端的x坐标


def searchRight(width, height, im):
    #实验

def searchTop(width, height, im):
    #实验

def searchBottom(width, height, im):
    #实验
```



Pillow应用

```
images = [f for f in os.listdir('testimages') if f.endswith('.bmp')]
for f in images:
    f = 'testimages\\'+f #假设所有椭圆图片放到testimages文件夹
    im = Image.open(f)
    width, height = im.size #获取图像大小

    #获取坐标
    x0 = searchLeft(width, height, im)
    x1 = searchRight(width, height, im)
    y0 = searchBottom(width, height, im)
    y1 = searchTop(width, height, im)

    center = ？？#实验

    im.putpixel(center, (255,0,0)) #把椭圆中心像素画成红色
    im.save(f[0:-4]+'_center.bmp') #保存为新图像文件（或者直接plot出来）
    im.close()
```