QBUS6840 TUT1

加载库文件

python 最大的优点在于有大量功能强大的库文件（library）。有了这些库文件，python 可以在各种应用场景中使用。比如 pandas 用来处理表格，numpy 用来处理矩阵运算，seaborn 用来数据可视化等等。python 甚至可以通过加载库文件的方式使用高效率的 C/C++ 代码。

有两种方式将库文件加载在 python 文件中：

- 使用 `import... as` 语句

`import` *library_name（.function_name）* `as` *nickname*

导入一个模块或者模块的某个函数,并用 nickname 代替模块（函数）原来的名字

> 如果不起 nickname ,以后每次要调用模块(函数) 时就要将模块和函数的名字写出来。

- 使用`from library_name import function_name`语句

导入了一个模块中的一个函数，之后直接使用函数名使用

```
import matploylib.pyplot as plt

from matploylib import pyplot
```

python原生时间类型变量

datetime 类型变量

原生的`python`用 datetime 表示时间类型变量

基本操作

构建 datetime

`datetime.datetime(*year*, *month*, *day*, *hour=0*, *minute=0*, *second=0*, *microsecond=0*, )`

```
import datetime
a = datetime.datetime(2008, 8, 8, 20, 0, 0, 946118)
print(a)
print(type(a))
```

> 要注意参数范围，如果参数不符合规范会报`ValueError`其中：

> `MINYEAR <= year <= MAXYEAR`
>
> `0 <= microsecond < 1000000`

MINYEAR 和 MAXYEAR 可以自己试一试：

```
print(datetime.MINYEAR)
print(datetime.MAXYEAR)
```

datetime 有 year month day 等参数

```
print(a.year)
print(a.month)
print(a.day)
```

构建 date 和 time

`datetime.date(year, month, day)`

`datetime.time(*hour=0*, *minute=0*, *second=0*, *microsecond=0*)`

```
a = datetime.date(2008, 8, 8)
b = datetime.time(2,1,2,100)
print(a)
print(b)
```

用 日期 和 时间 构建 datetime

`datetime.datetime.combine(*date*, *time*)`

```
c = datetime.datetime.combine(a,b)
print(type(c))
```

获取当前datetime

`datetime.datetime.now()` 获取当前时间

```
import datetime
a = datetime.datetime.now()
print(a)
```

~~获取当天date~~

~~`datetime.datetime.today()`~~

构建 timedelta

timedelta 是用来代表两个时间之间的的时间差，两个date或datetime对象相减时可以返回一个timedelta对象。

`datetime.timedelta([*days*[, *seconds*[, *microseconds*[, *milliseconds*[, *minutes*[, *hours*[, *weeks*]]]]]])`

所有参数可选，且默认都是0，参数的值可以是整数，浮点数，正数或负数

timedelta和可以和date,datetime对象进行加减操作

```
from datetime import timedelta
a = timedelta(days = 1, hours = 10, minutes = 8, seconds = 10)
print(a)
```

获取明天/前N天

```
tomorrow = datetime.date.today() + datetime.timedelta(days=1)
print(tomorrow)
```

前两天：

```
time = datetime.date.today() - datetime.timedelta(days=2)
print(time)
```

获取当天开始和结束时间(00:00:00 23:59:59)

```
d = datetime.datetime.combine(datetime.date.today(), datetime.time.min)
e = datetime.datetime.combine(datetime.date.today(),datetime.time.max)
print(d)
print(e)
```

获取两个datetime的时间差(秒数)

`timedelta.total_seconds()`

```
gap = (e-d).total_seconds()
print(gap)
```

datetime => string

`datetime.datetime.strftime(*format*)`

```
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
```

`%Y` : Year with century as a decimal number.

`%A` : Weekday as locale’s full name.

`%m`: **Month** as a zero-padded decimal number.

`%M`: **Minute** as a zero-padded decimal number.

其他规范

datetime <= string

`datetime.datetime.strptime(string ,*format*)`

```
e = datetime.datetime.strptime("2014-12-31 18:20:10", "%Y-%m-%d %H:%M:%S")
print(type(e))
```

dateutil 库

`dateutil` 库是对原生的 `datetime`库的扩展和增强

其中最有用的是

- `parser`根据字符串解析成datetime
- rrule根据定义的规则来生成datetime

parser

`parser.parse(timestr)`

根据字符串解析成datetime

字符串可以很随意，可以用时间日期的英文单词，可以用横线、逗号、空格等做分隔符。

- 没指定时间默认是0点，没指定日期默认是今天，没指定年份默认是今年
- fuzzy开启模糊匹配，过滤掉无法识别的时间日期字符)

```
print(parse("Wed, Nov 12"))
print(parse("2013-08-20"))
print(parse("20130820"))

print(parse("this is the wonderful moment 12:00:00,I feel
good",fuzzy=True))
```

rrule

`dateutil.rrule.rrule(freq, dtstart=None, interval=1, wkst=None, count=None, until=None, bysetpos=None, bymonth=None, bymonthday=None, byyearday=None, byeaster=None, byweekno=None, byweekday=None, byhour=None, byminute=None, bysecond=None, cache=False)`

函数生成一系列时间点

`freq`:时间单位。可以是 YEARLY, MONTHLY, WEEKLY, DAILY, HOURLY, MINUTELY, SECONDLY。

`dtstart``until`:开始和结束时间

`wkst`:周开始时间

`interval`:间隔

`count`:指定生成多少个

`byxxx`:指定匹配的周期

```
Recurrence rules may generate recurrence instances with an invalid date (e.g., February 30) or nonexistent local time (e.g., 1:30 AM on a day where the local time is moved forward by an hour at 1:00 AM). Such recurrence instances MUST be ignored and MUST NOT be counted as part of the recurrence set.
```

比如byweekday=(MO,TU)则只有周一周二的匹配。byweekday可以指定MO,TU,WE,TH,FR,SA,SU。即周一到周日。

```
from dateutil.rrule import rrule, MONTHLY
from datetime import datetime
start_date = datetime(2014, 12, 31)
a = list(rrule(freq=MONTHLY, count=4, dtstart=start_date))
print(a)
```

从 2014年12月31日开始按月生成4个 `datetime`

```
[datetime.datetime(2014, 12, 31, 0, 0), datetime.datetime(2015, 1, 31, 0, 0), datetime.datetime(2015, 3, 31, 0, 0), datetime.datetime(2015, 5, 31, 0, 0)]
```

Numpy 时间变量numpy.datetime64

Numpy 中存储日期时间变量的最基本方法是使用符合日期时间格式的字符串。 内部存储单元自动从字符串的形式中选择，可以是日期单位或时间单位，比如年（'Y'），月（'M'），周（'W'）和天（'D'），小时（'h'），分钟（'m'）），秒（'s'），毫秒（'ms'）

构建时间变量

`np.datetime64(str)`

string 需要符合日期时间格式

```
np.datetime64('2005-02-25')
np.datetime64('2005-02-25T03:30')
```

构建时间 array

```
np.array(['2007-07-13', '2006-01-13', '2010-08-13'], dtype='datetime64')
```

结果：

自动的识别 array 里的时间单位

```
In [13]:np.array(['2010-09-02','2005-02-25T03:30'],dtype='datetime64')
Out[13]: array(['2010-09-02T00:00', '2005-02-25T03:30'],
dtype='datetime64[m]')
```

用 np.arange 函数生成连续时间 array

`np.arange( str1, str2, dtype)`

str1 和 str2 是时间范围的上下界，dtype 是指生成的时间数据类型（是天（D）还是小时（H）等）

- dtype 应该是string 类型
- dtype 不写会默认是 str1 和 str2 中匹配到的最小时间单位

```
In [18]:
np.arange("2010-05-19","2010-05-24", step = 2, dtype = 'datetime64[D]')
Out[18]:
array(['2010-05-19', '2010-05-20', '2010-05-21', '2010-05-22',
       '2010-05-23'], dtype='datetime64[D]')
```

`datetime64 + np.arange( int )`

```
In [22]: np.datetime64('2009-01-01') + np.arange(10)
Out[22]:
array(['2009-01-01', '2009-01-02', '2009-01-03', '2009-01-04',
       '2009-01-05', '2009-01-06', '2009-01-07', '2009-01-08',
       '2009-01-09', '2009-01-10'], dtype='datetime64[D]')
```

Timedelta

numpy.timedelta64 和原生 datetime 一样是用来代表两个时间之间的的时间差，两个date或datetime对象相减时可以返回一个 numpy.timedelta64 对象

```
In [20]:
np.datetime64('2009-01-01') - np.datetime64('2008-01-01')
Out[20]: numpy.timedelta64(366,'D')
```

其他的计算和原生的 datetime 一样

```
>>>np.datetime64('2009') + np.timedelta64(20, 'D')
numpy.datetime64('2009-01-21')
```

需要注意的是 年和月的日期数不定，所以不能直接让日期和 单位为月和年的 timedelta 相运算

```
np.datetime64('2009-01-01')+ np.timedelta64(1,'Y')
Traceback (most recent call last):

  File "<ipython-input-21-933e9d1cc22f>", line 1, in <module>
    np.datetime64('2009-01-01')+ np.timedelta64(1,'Y')

TypeError: Cannot get a common metadata divisor for NumPy datetime metadata [D] and [Y] because they have incompatible nonlinear base time units
```

busday

numpy 的 datetime64 还有几个关于工作日的有趣函数

np.is_busday(datetime64):

返回是否是工作日

```
np.is_busday(np.datetime64('2011-07-15'))
True
```

np.busday_count(datetime64, datetime64):

返回一段时间内有多少个工作日

```
np.busday_count(np.datetime64('2011-07-11'), np.datetime64('2011-07-18'))
5
```

Pandas 时间变量

Pandas 用 `Timestamp` 类型数据 存储时间，用 `DatetimeIndex` 储存时间 array

=> Timestamp

`pd.Timestamp( str )`

```
friday = pd.Timestamp('2018-01-05')
```

`pd.to_datetime( str )`

```
pd.to_datetime('1/1/2018')
Out[28]: Timestamp('2018-01-01 00:00:00')

pd.to_datetime(np.datetime64('2018-01-01'))
Out[29]: Timestamp('2018-01-01 00:00:00')
```

=>Timedelta

```
pd.to_timedelta(1,'D')
Out[33]: Timedelta('1 days 00:00:00')
```

运算和原生 datetime 一样

生成连续时间 array

`pd.date_range(start=None, end=None, periods=None, freq=None)`

start/end : 起点和终点

period：生成多少个时间点

freq: 间隔

```
#month end frequency
pd.date_range(start='1/1/2018', periods=5, freq='M')
Out[30]:
DatetimeIndex(['2018-01-31', '2018-02-28', '2018-03-31', '2018-04-30',
               '2018-05-31'],
              dtype='datetime64[ns]', freq='M')
```

生成包含多个时间的时间series类型： DatetimeIndex, 其中每一个元素都是 timestamp 类型。

组成索引为时间点的 series

pd.series(array, index = DatetimeIndex)

```
idx = pd.date_range('2018-01-01', periods=5, freq='H')
ts = pd.Series(range(len(idx)), index=idx)
```

这样我们就可以用 series 的方法处理涉及时间的数据了

从 csv 中读取 TS 数据

以往我们的 csv 的 index 一般都是 从1开始的数字序列，但我们处理时间相关的数据时，往往要将csv中的时间数据作为 dataframe 的 index

lambda 匿名函数

lambda函数是一个很小的匿名函数。

lambda函数可以使用任意数量的参数，但只能有一个表达式。

`lambda *arguments* : *expression*`

```
x = lambda a : a + 10
x(5)

y = lambda a, b : a * b
y(5, 6)
```

x 成为这个lambda函数的名字,供之后调用。

读取 csv 数据并以时间作为 index

`pd.read_csv( filename, parse_dates, index_col,date_parser,keep_date_col=True)`

`filename`： csv 文件名

`parse_dates`: 需要作为 index 的时间数据所在列，可以是列所在的 index 或者列名，亦是 index 或列名组成的list 这是因为有可能日期和时间放在了两列，例如有 `date` 和 `time` 两列，并将新列命名为 actual 就要传参{'actual': ['date','time']}，如果只有 `time` 列，就传参['time']。

`index_col`：将 dataframe 的 index 设为某个列，一般设为通过 parse_dates 参数生成的列

`date_parser`：确定用哪一个函数分析存储时间信息的字符串 成 时间类型，一般是根据数据源的格式手动添加一个匿名函数

`keep_date_col`: 如果在 parse_dates 参数中将多个列合并成时间列，但是想保留原先的列，赋值 keep_date_col为 TRUE。

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m')
data_time= pd.read_csv('AirPassengers.csv', parse_dates=['Month'],
index_col='Month',date_parser=dateparse)
```

Time series 数据可视化

还是用我们熟悉的 matplotlib.pyplot 库

```
plt.plot(ts)
```

Time series 操作

提取某个年的数据：

```
ts['1949']
```

提取某个日期的数据：

```
ts['1949-01-01']
```

提取某段时间的数据：

```
ts2 = ts['1949-01-01':'1949-05-01']
```