Python3时间日期

- 获取当前日期和时间
- 获得当前日期
- 时间对象来表示日期
- 获得当前日期
- 从时间戳获取日期
- 打印今天的年、月和日
- 表示时间的时间对象
- 打印小时、分钟、秒和微秒
- Python datetime对象
- 打印年份、月份、时间、分钟和时间戳
- 两个日期和时间的区别
- timedelta 两个时间增量对象之间的差异
- 打印负时间增量对象
- 持续时间(秒)
- Python datetime格式
- 使用strftime()格式化日期
- strptime()

Python Datetime
- 当前时间
- 指定格式
- 创建时间对象
- strftime()方法
- Mastering Python Datetime (With Examples)
- 获得当前日期时间
- 将日期和时间分割为多个组件
- 创建手动日期和时间对象
- Timedelta
- datetime.strftime ()
- datetime.strptime()
- Datetime到Unix时间戳
- 时区处理

How to Format Dates in Python

Python, Datum und Zeit

Python标准库 时间与日期的区别 (time, datetime包)

Python获取时间范围内日期列表和周列表的函数
- 获取日期列表
- python给定起始和结束日期，如何得到中间所有日期
- Python获取两时段内日期、月份、小时的列表清单
- Python计算出给定的时间段的具体日期列表-大全
    - 计算昨天和明天的日期
    - 计算某一月有多少天
    - 计算周的日期函数。包含某一周开始、结束日期，周的详细日期列表
    - 计算自定义时间的日期函数。（比如计算20150811-20150922之间的日期列表）
    - 将自定义时间分割成星期来进行取值，不足一星期的按天来取值
    - 算出某一日期所属当前月的所有日期列表，dates是一个具体日期
- python获取任意时间段的日期(年月日)

python判断当前时间是否在某个时间段内
- python判断当前时间是否在某个时间段里
- python判断时间是否落在两个时区之间（只比较时刻不比较日期）
- 如何使用Python的datetime模块确定当前时间是否在指定范围内?
- python 如何判断两个时间是否在同一个5分钟时段内？ - 知乎

python 时间戳、时间字符格式化、判断时间在某个时间段内
- 获得当天时间的前一天、后一天（注意时间戳是以秒s为单位的，当将时间戳再转为格式化的时间字符串时，注意不能再用%f 毫秒）
- 判断某个时间点是否在两个时间点内
- 获得一分钟之前的时间，一小时之后的时间

时间分段--划分为等间隔的时间间隔

DateTimeRange 时间库
- 按天间隔
- 按月间隔

Python3 日期和时间

Python 程序能用很多方式处理日期和时间，转换日期格式是一个常见的功能。

Python 提供了一个 time 和 calendar 模块可以用于格式化日期和时间。

时间间隔是以秒为单位的浮点小数。

每个时间戳都以自从1970年1月1日午夜（历元）经过了多长时间来表示。

Python 的 time 模块下有很多函数可以转换常见日期格式。如函数time.time()用于获取当前时间戳, 如下实例:

```
import time;  # 引入time模块

ticks = time.time()
print ("当前时间戳为:", ticks)
```

```
当前时间戳为: 1568876848.039
```

时间戳单位最适于做日期运算。但是1970年之前的日期就无法以此表示了。太遥远的日期也不行，UNIX和Windows只支持到2038年。

什么是时间元组?

很多Python函数用一个元组装起来的9组数字处理时间:

序号	字段	值
0	4位数年	2008
1	月	1 到 12
2	日	1到31
3	小时	0到23
4	分钟	0到59
5	秒	0到61 (60或61 是闰秒)
6	一周的第几日	0到6 (0是周一)
7	一年的第几日	1到366 (儒略历)
8	夏令时	-1, 0, 1, -1是决定是否为夏令时的旗帜

上述也就是struct_time元组。这种结构具有如下属性：

|序号 |属性 |值| |----|----|----| |0 |tm_year |2008 |1 |tm_mon |1 到 12 |2 |tm_mday |1 到 31 |3 |tm_hour |0 到 23 |4 |tm_min |0 到 59 |5 |tm_sec |0 到 61 (60或61 是闰秒) |6 |tm_wday |0到6 (0是周一) |7 |tm_yday |一年中的第几天，1 到 366 |8 |tm_isdst |是否为夏令时，值有：1(夏令时)、0(不是夏令时)、-1(未知)，默认 -1

获取当前时间 time.localtime()

从返回浮点数的时间戳方式向时间元组转换，只要将浮点数传递给如localtime之类的函数。

```
import time

localtime = time.localtime(time.time())
print ("本地时间为 :", localtime)
```

```
本地时间为 : time.struct_time(tm_year=2019, tm_mon=9, tm_mday=19, tm_hour=15, tm_min=18,
tm_sec=28, tm_wday=3, tm_yday=262, tm_isdst=0)
```

获取格式化的时间 time.asctime()

你可以根据需求选取各种格式，但是最简单的获取可读的时间模式的函数是asctime():

```
import time

localtime = time.asctime( time.localtime(time.time()) )
print ("本地时间为 :", localtime)
```

```
本地时间为 : Thu Sep 19 15:19:07 2019
```

格式化日期 time.strftime()

我们可以使用 time 模块的 strftime 方法来格式化日期，：

```
time.strftime(format[, t])
```

```
import time

# 格式化成2016-03-20 11:45:39形式
print (time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))

# 格式化成Sat Mar 28 22:24:24 2016形式
print (time.strftime("%a %b %d %H:%M:%S %Y", time.localtime()))

# 将格式字符串转换为时间戳
a = "Sat Mar 28 22:24:24 2016"
print (time.mktime(time.strptime(a,"%a %b %d %H:%M:%S %Y")))
```

```
2019-09-19 15:21:18
Thu Sep 19 15:21:18 2019
1459175064.0
```

python中时间日期格式化符号：

```
%y 两位数的年份表示（00-99）
%Y 四位数的年份表示（000-9999）
%m 月份（01-12）
%d 月内中的一天（0-31）
%H 24小时制小时数（0-23）
%I 12小时制小时数（01-12）
%M 分钟数（00=59）
%S 秒（00-59）
%a 本地简化星期名称
%A 本地完整星期名称
%b 本地简化的月份名称
%B 本地完整的月份名称
%c 本地相应的日期表示和时间表示
%j 年内的一天（001-366）
%p 本地A.M.或P.M.的等价符
%U 一年中的星期数（00-53）星期天为星期的开始
%w 星期（0-6），星期天为星期的开始
%W 一年中的星期数（00-53）星期一为星期的开始
%x 本地相应的日期表示
%X 本地相应的时间表示
%Z 当前时区的名称
%% %号本身
```

获取某月日历

Calendar模块有很广泛的方法用来处理年历和月历，例如打印某月的月历：

```
import calendar

cal = calendar.month(2016, 1)
print ("以下输出2016年1月份的日历:")
print (cal)
```

```
以下输出2016年1月份的日历:
    January 2016
Mo Tu We Th Fr Sa Su
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

Time 模块

Time 模块包含了以下内置函数，既有时间处理的，也有转换时间格式的：

1-time.altzone

返回格林威治西部的夏令时地区的偏移秒数。如果该地区在格林威治东部会返回负值（如西欧，包括英国）。对夏令时启用地区才能使用。 以下实例展示了 altzone()函数的使用方法：

```
>>> import time
>>> print ("time.altzone %d " % time.altzone)
time.altzone -28800
```

2 time.asctime([tupletime])

接受时间元组并返回一个可读的形式为"Tue Dec 11 18:07:14 2008"（2008年12月11日 周二18时07分14秒）的24个字符的字符串。 以下实例展示了 asctime()函数的使用方法：

```
>>> import time
>>> t = time.localtime()
>>> print ("time.asctime(t): %s " % time.asctime(t))
time.asctime(t): Thu Apr  7 10:36:20 2016
```

3 time.clock()

用以浮点数计算的秒数返回当前的CPU时间。用来衡量不同程序的耗时，比time.time()更有用。 实例

由于该方法依赖操作系统，在 Python 3.3 以后不被推荐，而在 3.8 版本中被移除，需使用下列两个函数替代。

```
time.perf_counter()  # 返回系统运行时间
time.process_time()  # 返回进程运行时间
```

4 time.ctime([secs])

作用相当于asctime(localtime(secs))，未给参数相当于asctime() 以下实例展示了 ctime()函数的使用方法：

```
>>> import time
>>> print ("time.ctime() : %s" % time.ctime())
time.ctime() : Thu Apr  7 10:51:58 2016
```

5 time.gmtime([secs])

接收时间戳（1970纪元后经过的浮点秒数）并返回格林威治天文时间下的时间元组t。注：t.tm_isdst始终为0 以下实例展示了 gmtime()函数的使用方法：

```
>>> import time
>>> print ("gmtime :", time.gmtime(1455508609.34375))
gmtime : time.struct_time(tm_year=2016, tm_mon=2, tm_mday=15, tm_hour=3, tm_min=56, tm_sec=49,
tm_wday=0, tm_yday=46, tm_isdst=0)
```

6 time.localtime([secs]
接收时间戳（1970纪元后经过的浮点秒数）并返回当地时间下的时间元组t（t.tm_isdst可取0或1，取决于当地当时是不是夏令时）。 以下实例展示了 localtime()函数的使用方法：

```
>>> import time
>>> print ("localtime(): ", time.localtime(1455508609.34375))
localtime():  time.struct_time(tm_year=2016, tm_mon=2, tm_mday=15, tm_hour=11, tm_min=56,
tm_sec=49, tm_wday=0, tm_yday=46, tm_isdst=0)
```

7 time.mktime(tupletime)
接受时间元组并返回时间戳（1970纪元后经过的浮点秒数）。
8 time.sleep(secs)
推迟调用线程的运行，secs指秒数。 以下实例展示了 sleep()函数的使用方法：

```
#!/usr/bin/python3
import time

print ("Start : %s" % time.ctime())
time.sleep( 5 )
print ("End : %s" % time.ctime())
```

9 time.strftime(fmt[,tupletime])
接收以时间元组，并返回以可读字符串表示的当地时间，格式由fmt决定。
以下实例展示了 strftime()函数的使用方法：

```
>>> import time
>>> print (time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))
2016-04-07 11:18:05
```

10 time.strptime(str,fmt='%a %b %d %H:%M:%S %Y')
根据fmt的格式把一个时间字符串解析为时间元组。 以下实例展示了 strptime()函数的使用方法：

```
>>> import time
>>> struct_time = time.strptime("30 Nov 00", "%d %b %y")
>>> print ("返回元组: ", struct_time)
返回元组:  time.struct_time(tm_year=2000, tm_mon=11, tm_mday=30, tm_hour=0, tm_min=0, tm_sec=0,
tm_wday=3, tm_yday=335, tm_isdst=-1)
```

11 time.time( )
返回当前时间的时间戳（1970纪元后经过的浮点秒数）。 以下实例展示了 time()函数的使用方法：

```
>>> import time
>>> print(time.time())
1459999336.1963577
```

12 time.tzset()
根据环境变量TZ重新初始化时间相关设置。
13 time.perf_counter()
返回计时器的精准时间（系统的运行时间），包含整个系统的睡眠时间。由于返回值的基准点是未定义的，所以，只有连续调用的结果之间的差才是有效的。 实例

14 time.process_time() 返回当前进程执行 CPU 的时间总和，不包含睡眠时间。由于返回值的基准点是未定义的，所以，只有连续调用的结果之间的差才是有效的。

Time模块包含了以下2个非常重要的属性：

1 time.timezone

属性time.timezone是当地时区（未启动夏令时）距离格林威治的偏移秒数（>0，美洲;<=0大部分欧洲，亚洲，非洲）。

2 time.tzname

属性time.tzname包含一对根据情况的不同而不同的字符串，分别是带夏令时的本地时区名称，和不带的。

日历（Calendar）模块

此模块的函数都是日历相关的，例如打印某月的字符月历。

星期一是默认的每周第一天，星期天是默认的最后一天。更改设置需调用calendar.setfirstweekday()函数。模块包含了以下内置函数：

1 calendar.calendar(year,w=2,l=1,c=6)

返回一个多行字符串格式的year年年历，3个月一行，间隔距离为c。 每日宽度间隔为w字符。每行长度为21* W+18+2* C。l是每星期行数。

2 calendar.firstweekday( )

返回当前每周起始日期的设置。默认情况下，首次载入caendar模块时返回0，即星期一。

3 calendar.isleap(year)

是闰年返回 True，否则为 false。

```
>>> import calendar
>>> print(calendar.isleap(2000))
True
>>> print(calendar.isleap(1900))
False
```

4 calendar.leapdays(y1,y2)

返回在Y1，Y2两年之间的闰年总数。

5 calendar.month(year,month,w=2,l=1)

返回一个多行字符串格式的year年month月日历，两行标题，一周一行。每日宽度间隔为w字符。每行的长度为7* w+6。l是每星期的行数。

6 calendar.monthcalendar(year,month)

返回一个整数的单层嵌套列表。每个子列表装载代表一个星期的整数。Year年month月外的日期都设为0;范围内的日子都由该月第几日表示，从1开始。

7 calendar.monthrange(year,month)

返回两个整数。第一个是该月的星期几，第二个是该月有几天。星期几是从0（星期一）到 6（星期日）。

```
>>> import calendar
>>> calendar.monthrange(2014, 11)
(5, 30)
(5, 30)解释：5 表示 2014 年 11 月份的第一天是周六，30 表示 2014 年 11 月份总共有 30 天。
```

8 calendar.prcal(year,w=2,l=1,c=6)

相当于 print calendar.calendar(year,w,l,c).

9 calendar.prmonth(year,month,w=2,l=1)

相当于 print calendar.calendar（year，w，l，c）。

10 calendar.setfirstweekday(weekday)

设置每周的起始日期码。0（星期一）到6（星期日）。

11 calendar.timegm(tupletime)

和time.gmtime相反：接受一个时间元组形式，返回该时刻的时间戳（1970纪元后经过的浮点秒数）。

12 calendar.weekday(year,month,day)

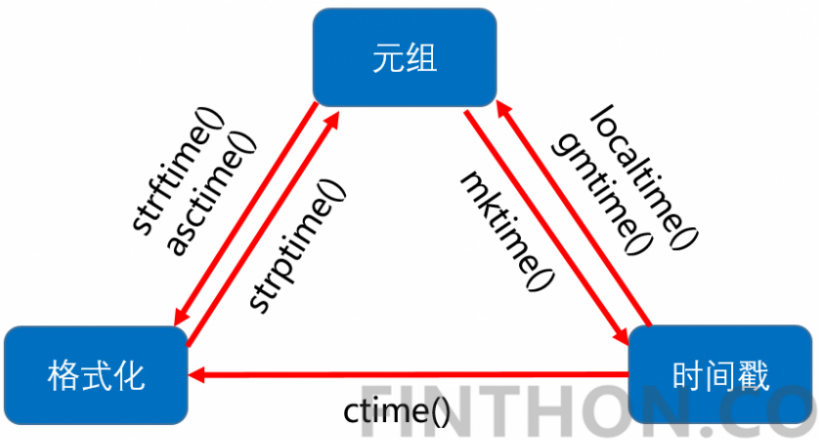
返回给定日期的日期码。0（星期一）到6（星期日）。月份为 1（一月） 到 12（12月）。

time 模块

datetime模块

Python时间模块——time模块

time模块中常用的函数可以总结成下图，基本上就是三者如何转换：



time.gmtime()

该函数和localtime()的功能一样，只是它返回的时间是格林威治天文时间（UTC），也就是世界标准时间。中国时间为UTC+8。

```
import time
print(time.gmtime())
print(time.localtime())
```

```
time.struct_time(tm_year=2019, tm_mon=9, tm_mday=19, tm_hour=7, tm_min=53, tm_sec=15, tm_wday=3,
tm_yday=262, tm_isdst=0)
time.struct_time(tm_year=2019, tm_mon=9, tm_mday=19, tm_hour=15, tm_min=53, tm_sec=15,
tm_wday=3, tm_yday=262, tm_isdst=0)
```

time.mktime()

该函数将一个元组转换成时间戳。

```
import time
print(time.mktime(time.localtime()))
```

```
1568879650.0
```

time.sleep()

该函数能让程序线程暂停休息，传入几秒，休息几秒。

```
import time
print(time.time())
time.sleep(3)  # 延迟3秒
print(time.time())
```

```
1568879687.072
1568879690.073
```

time.asctime()

该函数将一个元组转换成格式化时间。如果没有传入参数，默认传入time.localtime()。

```
import time
print(time.asctime())
```

```
Thu Sep 19 15:55:56 2019
```

time.ctime()

该函数将一个时间戳转换成格式化时间。如果没有传入参数，默认传入time.time()。

```
import time
print(time.ctime())
```

```
Thu Sep 19 15:56:42 2019
```

time.strptime()

该函数按照格式化字符把一个格式化时间字符串转成元组。

```
import time
print(time.strptime('2018-10-29 18:46:14', '%Y-%m-%d %X'))
```

```
time.struct_time(tm_year=2018, tm_mon=10, tm_mday=29, tm_hour=18, tm_min=46, tm_sec=14,
tm_wday=0, tm_yday=302, tm_isdst=-1)
```

time.strftime()

该函数按照格式化字符把一个元组转换成格式化时间字符串。如果没有传入参数，默认传入time.localtime()。

```
import time
print(time.strftime("%Y-%m-%d %X", time.localtime()))
```

```
2019-09-19 15:57:31
```

你可能会发现，ctime()和asctime()的字符串格式不适合您的应用程序。相反，你可能希望以对用户更有意义的方式格式化字符串。

这方面的一个例子是，如果你希望将你的时间显示在一个考虑地区信息的字符串中。

要格式化字符串，给定struct_time或Python time元组，可以使用strftime()，它代表“字符串格式化时间”。

strftime()接受两个参数:
format指定字符串中时间元素的顺序和形式。

t是一个可选的时间元组。

要格式化字符串，可以使用指令。指令是以%开头的字符序列，它指定一个特定的时间元素，例如:

```
%d:每月的哪一天
%m:一年中的一个月
% Y:年
```

例如，您可以使用ISO 8601标准输出本地时间的日期，如下所示:

```
import time
print(time.strftime('%Y-%m-%d', time.localtime()))
```

```
2019-09-19
```

需要注意的是，当传入的时间中包括周数和天数（%U和%W），该函数才能使用。

更重要的是:虽然使用Python时间表示日期是完全有效和可接受的，但是你还应该考虑使用Python的datetime模块，它提供了快捷方式和更健壮的框架，可以同时处理日期和时间。 例如，可以使用datetime简化ISO 8601格式的日期输出:

```
from datetime import date
print(date(year=2019, month=3, day=1).isoformat())
```

正如之前看到的，使用strftime()而不是asctime()的一大好处是它能够呈现使用特定于语言环境的信息的时间戳。

例如，如果你想以对语言环境敏感的方式表示日期和时间，则不能使用asctime():

```
from time import asctime
print(asctime())


import locale
print(locale.setlocale(locale.LC_TIME, 'zh_HK'))  # Chinese - Hong Kong

print(asctime())
```

```
Thu Sep 19 18:32:34 2019
```

```
---------------------------------------------------------------------------

Error                                     Traceback (most recent call last)

<ipython-input-50-3e12093e315b> in <module>()
      4
      5 import locale
----> 6 print(locale.setlocale(locale.LC_TIME, 'zh_HK'))  # Chinese - Hong Kong
      7
      8 print(asctime())
```

```
d:\Anaconda3\lib\locale.py in setlocale(category, locale)
    596         # convert to string
    597         locale = normalize(_build_localename(locale))
--> 598     return _setlocale(category, locale)
    599
    600 def resetlocale(category=LC_ALL):
```

```
Error: unsupported locale setting
```

请注意，即使以编程方式更改了区域设置，asctime()仍然以与以前相同的格式返回日期和时间。

技术细节:LC_TIME是用于日期和时间格式化的locale类别。地区参数'zh_HK'可能不同，这取决于你的系统。

但是，当使用strftime()时，将看到它使用了locale时间:

```
from time import strftime, localtime
print(strftime('%c', localtime()))


import locale
print(locale.setlocale(locale.LC_TIME, 'zh_HK'))  # Chinese - Hong Kong

print(strftime('%c', localtime()))
```

```
Thu Sep 19 18:35:09 2019
```

```
---------------------------------------------------------------------------

Error                                     Traceback (most recent call last)

<ipython-input-51-a724f9521658> in <module>()
      4
      5 import locale
----> 6 print(locale.setlocale(locale.LC_TIME, 'zh_HK'))  # Chinese - Hong Kong
      7
      8 print(strftime('%c', localtime()))
```

```
d:\Anaconda3\lib\locale.py in setlocale(category, locale)
    596         # convert to string
    597         locale = normalize(_build_localename(locale))
--> 598     return _setlocale(category, locale)
    599
    600 def resetlocale(category=LC_ALL):
```

```
Error: unsupported locale setting
```

在这里，你已经成功地利用了地区信息，因为您使用了strftime()。

```
注意:%c是表示适合于地区的日期和时间的指令。
```

如果时间元组没有传递给参数t，那么默认情况下strftime()将使用localtime()的结果。因此，可以通过删除可选的第二个参数来简化上面的示例:

```
from time import strftime
print(strftime('The current local datetime is: %c'))
```

```
The current local datetime is: Thu Sep 19 18:36:46 2019
```

在这里，你使用了默认时间，而不是将自己的时间作为参数传递。另外，请注意format参数可以由格式化指令以外的文本组成。

进一步阅读:查看strftime()可用的指令的完整列表。
Python时间模块还包括将时间戳转换回struct_time对象的逆操作。

Python time comparison

判断当前时间比指定时间早还是晚

你无法将特定的时间点(如“现在”)与不固定的重复事件(每天早上8点发生)进行比较。

你可以看看现在是在今天早上8点之前还是之后:

```
import datetime
now = datetime.datetime.now()
print("now:", now)
today8am = now.replace(hour=8, minute=0, second=0, microsecond=0)  # 替换当前时间到早上8点

print("today8am:", today8am)
print(now < today8am)  # 判断当前时间是否比早上8点要早
print(now == today8am)
print(now > today8am)
```

```
now: 2019-09-19 16:05:04.318000
today8am: 2019-09-19 08:00:00
False
False
True
```

使用datetime

使用datetime对象的time()方法来获取一天的时间，可以使用它进行比较，而不需要考虑日期:

```
this_morning = datetime.datetime(2009, 12, 2, 9, 30)
last_night = datetime.datetime(2009, 12, 1, 20, 0)

print("this_morning:", this_morning)
print("last_night:", last_night)
print(this_morning.time() < last_night.time())
```

```
this_morning: 2009-12-02 09:30:00
last_night: 2009-12-01 20:00:00
True
```

```
a = datetime.datetime(2009, 12, 2, 10, 24, 34, 198130)
b = datetime.datetime(2009, 12, 2, 10, 24, 36, 910128)
print(a < b)
```

```
True
```

另一种不添加依赖项或使用datetime的方法是对time对象的属性进行一些计算。它有小时、分钟、秒、毫秒和时区。对于非常简单的比较，小时和分钟应该足够了。

```
d = datetime.datetime.utcnow()  # 格林尼治时间，中国+8，东八区时间
t = d.time()
print(t)
print(t.hour,t.minute,t.second)
```

```
08:39:49.792000
8 39 49
```

格式化UTC时间

Format datetime.utcnow() time

```
import datetime

presentTime = datetime.datetime.utcnow()
print(presentTime.strftime('%B %d %Y - %H:%M:%S'))
```

```
September 19 2019 - 08:37:13
```

How to use Date and Time in Python

```
import time
import datetime

print("Time in seconds since the epoch: %s" %time.time())
print("Current date and time: " , datetime.datetime.now())
print("Or like this: " ,datetime.datetime.now().strftime("%y-%m-%d-%H-%M"))

print("Current year: ", datetime.date.today().strftime("%Y"))
print("Month of year: ", datetime.date.today().strftime("%B"))
print("Week number of the year: ", datetime.date.today().strftime("%W"))
print("Weekday of the week: ", datetime.date.today().strftime("%w"))
print("Day of year: ", datetime.date.today().strftime("%j"))
print("Day of the month : ", datetime.date.today().strftime("%d"))
print("Day of week: ", datetime.date.today().strftime("%A"))
```

```
Time in seconds since the epoch: 1568882330.776
Current date and time:  2019-09-19 16:38:50.776000
Or like this:  19-09-19-16-38
Current year:  2019
Month of year:  September
Week number of the year:  37
Weekday of the week:  4
Day of year:  262
Day of the month :  19
Day of week:  Thursday
```

比较日期

Comparing dates in Python

```
# importing datetime module
import datetime

# date in yyyy/mm/dd format
d1 = datetime.datetime(2018, 5, 3)
d2 = datetime.datetime(2018, 6, 1)

# Comparing the dates will return
# either True or False
print("d1 is greater than d2 : ", d1 > d2)
print("d1 is less than d2 : ", d1 < d2)
print("d1 is not equal to d2 : ", d1 != d2)
```

```
d1 is greater than d2 :  False
d1 is less than d2 :  True
d1 is not equal to d2 :  True
```

日期排序

```
# importing datetime module
from datetime import *

# create empty list
group = []

# add today's date
group.append(date.today())

# create some more dates
d = date(2015, 6, 29)
group.append(d)

d = date(2011, 4, 7)
group.append(d)

# add 25 days to the date
# and add to the list
group.append(d + timedelta(days = 25))

# sort the list
group.sort()

# print the dates
for d in group:
    print(d)
```

```
2011-04-07
2011-05-02
2015-06-29
2019-09-19
```

比较日期大小

```
# importing datetime module
from datetime import *

# Enter birth dates and store
# into date class objects
d1, m1, y1 = [int(x) for x in input("Enter first"
        " person's date(DD/MM/YYYY) : ").split('/')]

b1 = date(y1, m1, d1)

# Input for second date
d2, m2, y2 = [int(x) for x in input("Enter second"
        " person's date(DD/MM/YYYY) : ").split('/')]

b2 = date(y2, m2, d2)

# Check the dates
```

```
if b1 == b2:
    print("Both persons are of equal age")

elif b1 > b2:
    print("The second person is older")

else:
    print("The first person is older")
```

```
Enter first person's date(DD/MM/YYYY) : 12/05/2017
Enter second person's date(DD/MM/YYYY) : 10/11/2015
The second person is older
```

日期列表排序

```
# Import the datetime module
from datetime import datetime


# Function to print the data stored in the list
def printDates(dates):

    for i in range(len(dates)):
        print(dates[i])


if __name__ == "__main__":

    dates = ["23 Jun 2018", "2 Dec 2017", "11 Jun 2018", "01 Jan 2019",
             "10 Jul 2016", "01 Jan 2007"]


    # Sort the list in ascending order of dates
    dates.sort(key=lambda date: datetime.strptime(date, '%d %b %Y'))

    # Print the dates in a sorted order
    printDates(dates)
```

```
01 Jan 2007
10 Jul 2016
2 Dec 2017
11 Jun 2018
23 Jun 2018
01 Jan 2019
```

A Beginner’ s Guide to the Python time Module

将纪元定义为1970年1月1日的子夜(在Windows和大多数UNIX系统中定义的纪元)，那么可以将1970年1月2日的子夜表示为自纪元起的86400秒。 这是因为一分钟有60秒，一小时有60分钟，还有24小时

```
print(60 * 60 * 24, "s")
```

```
86400 s
```

时间元组

9个参数

```
from time import struct_time
time_tuple = (2019, 2, 26, 7, 6, 55, 1, 57, 0)
time_obj = struct_time(time_tuple)
print(time_obj)
```

```
time.struct_time(tm_year=2019, tm_mon=2, tm_mday=26, tm_hour=7, tm_min=6, tm_sec=55, tm_wday=1,
tm_yday=57, tm_isdst=0)
```

```
time_tuple = (2019, 7, 23, 17, 8, 58)
time_obj = struct_time(time_tuple)
print(time_obj)
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-35-24776091c6b4> in <module>()
      1 time_tuple = (2019, 7, 23, 17, 8, 58)
----> 2 time_obj = struct_time(time_tuple)
      3 print(time_obj)
```

```
TypeError: time.struct_time() takes an at least 9-sequence (6-sequence given)
```

可以使用属性名而不是索引访问time_obj的特定元素:

```

```

```
day_of_year = time_obj.tm_yday
print(day_of_year)  # 一年的第几天

day_of_month = time_obj.tm_mday
print(day_of_month)  # 一个月的第几天
```

```
57
26
```

将Python时间(以秒为单位)转换为对象

```
import time
print(time.gmtime(1.99))
```

```
time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0, tm_sec=1, tm_wday=3,
tm_yday=1, tm_isdst=0)
```

注意，即使您通过的秒数非常接近2，.99小数秒也会被忽略，如tm_sec=1所示。

gmtime()的secs参数是可选的，这意味着您可以不带参数地调用gmtime()。这样做将提供当前UTC时间:

```
import time
print(time.gmtime())
```

```
time.struct_time(tm_year=2019, tm_mon=9, tm_mday=19, tm_hour=9, tm_min=2, tm_sec=58, tm_wday=3,
tm_yday=262, tm_isdst=0)
```

这个函数在时间范围内没有逆函数。相反，必须在Python的calendar模块中查找一个名为timegm()的函数:

```
import calendar
import time

print(time.gmtime())
print(calendar.timegm(time.gmtime()))
```

```
time.struct_time(tm_year=2019, tm_mon=9, tm_mday=19, tm_hour=9, tm_min=7, tm_sec=6, tm_wday=3,
tm_yday=262, tm_isdst=0)
1568884026
```

timegm()接收一个元组(或struct_time，因为它是元组的子类)，并返回自历元以来的相应秒数。

tm_zone存储本地时区

```
import time
current_local = time.localtime()
print(current_local.tm_zone)
```

```
?D1¨²¡À¨º¡Á?¨º¡À??
```

在这里，可以看到localtime()返回一个struct_time，时区设置为CST(中央标准时间)。

正如你之前看到的，你也可以根据UTC偏移量和DST(如果适用)这两条信息来判断时区:

```
import time
current_local = time.localtime()

print(current_local.tm_gmtoff)
print(current_local.tm_isdst)
```

```
28800
0
```

在本例中，可以看到current_local比格林尼治时间(代表格林尼治平均时间)晚21600秒。GMT是没有UTC偏移的时区:UTC±00:00。

21600秒除以秒每小时(3600)意味着当前本地时间是GMT-06:00(或UTC-06:00)。

可以使用GMT偏移量加上DST状态来推断current_local在标准时间是UTC-06:00，它对应于中央标准时区。

与gmtime()类似，可以在调用localtime()时忽略secs参数，它将在struct_time中返回当前的本地时间:

```
import time
print(time.localtime())
```

```
time.struct_time(tm_year=2019, tm_mon=9, tm_mday=19, tm_hour=18, tm_min=16, tm_sec=30,
tm_wday=3, tm_yday=262, tm_isdst=0)
```

将本地时间对象转换为秒

```
import time

time_tuple = (2019, 3, 10, 8, 50, 6, 6, 69, 1)
print(time.mktime(time_tuple))


time_struct = time.struct_time(time_tuple)
print(time.mktime(time_struct))
```

```
1552179006.0
1552179006.0
```

记住t必须是表示本地时间的元组，而不是UTC，这一点很重要:

```
from time import gmtime, mktime

# 1
current_utc = time.gmtime()
print(current_utc)

# 2
current_utc_secs = mktime(current_utc)
print(current_utc_secs)

# 3
print(time.gmtime(current_utc_secs))
```

```
time.struct_time(tm_year=2019, tm_mon=9, tm_mday=19, tm_hour=10, tm_min=19, tm_sec=19,
tm_wday=3, tm_yday=262, tm_isdst=0)
1568859559.0
time.struct_time(tm_year=2019, tm_mon=9, tm_mday=19, tm_hour=2, tm_min=19, tm_sec=19, tm_wday=3,
tm_yday=262, tm_isdst=0)
```

这个例子说明了为什么在本地时间使用mktime()很重要，而不是UTC:
没有参数的gmtime()使用UTC返回struct_time。current_utc显示2019年9月19日10:19:19 UTC。这是准确的，因为当前时区CST是UTC-08:00，所以UTC应该比当地时间早8个小时。 mktime()试图返回秒数，期望本地时间，但是您却传递了current_utc。因此，它没有理解current_utc是UTC时间，而是假设您的意思是2019年9月19日2:19:19 CST。 然后使用gmtime()将这些秒转换回UTC，这会导致不一致

将Python时间字符串转换为对象

当处理与日期和时间相关的字符串时，将时间戳转换为时间对象是非常有价值的。

要将时间字符串转换为struct_time，可以使用strptime()，它代表“字符串解析时间”：

```
from time import strptime

print(strptime('2019-03-01', '%Y-%m-%d'))
```

```
time.struct_time(tm_year=2019, tm_mon=3, tm_mday=1, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=4,
tm_yday=60, tm_isdst=-1)
```

strptime()的第一个参数必须是希望转换的时间戳。第二个参数是时间戳的格式。

格式参数是可选的，默认为'%a %b %d %H:%M:%S %Y'。因此，如果你有该格式的时间戳，不需要将其作为参数传递:

```
print(strptime('Fri Mar 01 23:38:40 2019'))
```

```
time.struct_time(tm_year=2019, tm_mon=3, tm_mday=1, tm_hour=23, tm_min=38, tm_sec=40, tm_wday=4,
tm_yday=60, tm_isdst=-1)
```

因为struct_time有9个关键日期和时间参数，所以strptime()必须为那些不能从字符串解析的组件提供合理的默认值。

在前面的示例中，tm_isdst=-1。这意味着strptime()不能通过时间戳确定它是否代表夏令时。

现在，知道了如何以各种方式使用时间模块来处理Python时间和日期。然而，除了简单地创建时间对象、获取Python时间字符串和使用历元之后的秒以外，还有其他一些时间用途。

延迟

另一个真正有用的Python时间函数是sleep()，它将线程的执行挂起一段指定的时间。

例如，你可以这样暂停程序的执行10秒:

```
from time import sleep, strftime
print(strftime('%c'))

sleep(10)
print(strftime('%c'))
```

```
Thu Sep 19 18:42:44 2019
Thu Sep 19 18:42:54 2019
```

程序将打印第一个格式化的datetime字符串，然后暂停10秒，最后打印第二个格式化的datetime字符串。

也可以通过分数秒睡眠():

```
from time import sleep

print(strftime('%c'))

sleep(0.5)
print(strftime('%c'))
```

```
Thu Sep 19 18:44:00 2019
Thu Sep 19 18:44:00 2019
```

sleep()对于测试或使程序等待任何原因都是有用的，但是必须小心不要停止生产代码，除非有充分的理由这样做。

在Python 3.5之前，发送给进程的信号可能会中断sleep()。但是，在3.5或更高版本中，sleep()将始终挂起执行至少一段指定的时间，即使进程接收到一个信号。

sleep()只是一个Python时间函数，它可以帮助你测试程序并使它们更健壮。

性能度量

可以使用时间来度量程序的性能。

实现此目的的方法是使用perf_counter()，顾名思义，它提供了一个高分辨率的性能计数器来测量短时间间隔。

要使用perf_counter()，需要在代码开始执行之前以及代码执行完成之后放置一个计数器:

```
from time import perf_counter
def longrunning_function():
    for i in range(1, 11):
        time.sleep(i / i ** 2)

start = perf_counter()
longrunning_function()
end = perf_counter()
execution_time = (end - start)
print(execution_time)
```

```
2.9305334309985227
```

如何在Python中获取N天前的日期

How to get the date N days ago in Python

```
from datetime import datetime, timedelta

N = 2

date_N_days_ago = datetime.now() - timedelta(days=N)

print("当前时间：", datetime.now())
print("两天前的时间：", date_N_days_ago)
```

```
当前时间： 2019-09-19 18:54:25.476000
两天前的时间： 2019-09-17 18:54:25.476000
```

timedelta Objects

Dealing with datetimes like a pro in Python

```
import datetime as dt

log_line = "/logger/ || 70.123.102.76 || - || 31/Aug/2015:23:49:01 +0000  || GET /logger/?
action-view&site_id=123 HTTP/1.1 || 200 || 236 || https://foo.com/some/url || Mozilla/5.0
(Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36 || -
|| - || - || 0.000"
parts = log_line.split(' || ')
date, request, referrer, user_agent = parts[3], parts[4], parts[7], parts[8]
date = dt.datetime.strptime(date[:-7], '%d/%b/%Y:%H:%M:%S')

print(date)
```

```
2015-08-31 23:49:01
```

显示带有时区的日期时间

首先，这种通用的日期格式(RFC 3339)看起来不太吸引人。美国人希望看到他们熟悉的“M/D/Y”和12小时的大钟(上午/下午)。欧洲人正在寻找“D/M/Y”，其中一些人正在寻找“D.M.”大部分时间是24小时。

然后，所有日期和时间都在UTC中(这是服务器记录它们的方式)。所以顾客不得不眯着眼睛思考“嘿，这到底是什么时候发生的?”——在他们的脑子里做时区偏移(甚至不要让我开始做DST)——这是一项艰苦的工作!

因此，您决定在显示信息之前将日期和时间转换为本地时区。

```
import datetime as dt
import pytz

event_date = dt.datetime(2017, 8, 7, 15, 10, 1).replace(tzinfo=pytz.UTC)
user_timezone = pytz.timezone('America/Los_Angeles')
local_date = event_date.astimezone(user_timezone)
print(local_date.strftime('%m/%d/%Y %H:%M:%S %Z'))
```

```
08/07/2017 08:10:01 PDT
```

舍入(删除)日期时间

接下来，你希望以聚合的形式显示数据，以计算每个URL收集了多少视图。这将使数据更加有用，使客户更加高兴。

为了将时间累计起来，你写了一小段代码，将日期时间四舍五入到一个小时:

```
rounded = date.replace(minute=0, second=0, microsecond=0)
```

求区间的边

你想要添加另一个特性:比较数周的数据。与上周相比，本周的活动情况如何?页面的访问量是增加了，还是减少了?要做到这一点，你需要确定一周的开始时间:

```
import datetime as dt

today = dt.datetime.now()
monday = (today - dt.timedelta(days=today.weekday())).replace(hour=0, minute=0, second=0,
microsecond=0)

print(monday)
```

```
2019-09-16 00:00:00
```

上一周时间

```
previous_monday = monday - dt.timedelta(days=7)

print(previous_monday)
```

```
2019-09-09 00:00:00
```

创建时间范围

接下来，你希望让客户能够选择自己的开始和结束时间间隔。这意味着生成一系列日期。为此，你需要导入另一个名为dateutil的模块:

```
from dateutil import rrule

print(list(rrule.rrule(rrule.WEEKLY, dtstart=dt.datetime(2017, 7, 17), count=4)))  # 间隔一周的时间列表
```

```
[datetime.datetime(2017, 7, 17, 0, 0), datetime.datetime(2017, 7, 24, 0, 0),
datetime.datetime(2017, 7, 31, 0, 0), datetime.datetime(2017, 8, 7, 0, 0)]
```

Pendulum(钟摆)

Pendulum是一个开源的Python库，创建它是为了提供一个更好、更智能的下拉式datetime替代品。源代码:https://github.com/sdispater/pendulum。

我还应该提到arrow (https://github.com/crsmithdev/arrow)。

Arrow是一个更老、更知名的库，它解决了与摆锤相同的问题:为datetime提供了更丰富、更方便的API，填补了标准库的空白。

不过，我将主要关注Pendulum，因为它修复了arrow的一些bug和问题。它更加一致和健壮(以牺牲一些性能为代价)。

```
import pendulum

now_in_shanghai = pendulum.now('Asia/Shanghai')  # 显示上海时间
print(now_in_shanghai)
```

```
2019-09-19T19:17:46.731000+08:00
```

```
# 时区转换
print(now_in_shanghai.in_timezone('UTC'))
```

```
2019-09-19T11:17:46.731000+00:00
```

```
tomorrow = pendulum.now().add(days=1)
last_week = pendulum.now().subtract(weeks=1)

print("明天:", tomorrow)
print("上一周:", last_week)
```

```
明天: 2019-09-20T19:20:59.772000+08:00
上一周: 2019-09-12T19:20:59.772000+08:00
```

```
past = pendulum.now().subtract(minutes=2)
print("两分钟前:", past.diff_for_humans())
```

```
两分钟前: 2 minutes ago
```

```
delta = past - last_week  # 时间差
print("时间差:", delta)
print(delta.hours)
```

```
时间差: <Period [2019-09-12T19:20:59.772000+08:00 -> 2019-09-19T19:19:40+08:00]>
23
```

时间翻译

```
print(delta.in_words(locale='en'))
```

```
6 days 23 hours 58 minutes 40 seconds
```

```
# Proper handling of datetime normalization  时间格式化
print(pendulum.datetime(2013, 3, 31, 2, 30, tz='Asia/Shanghai'))
```

```
2013-03-31T02:30:00+08:00
```

```
# Proper handling of dst transitions  dst时间转换
just_before = pendulum.datetime(2013, 3, 31, 1, 59, 59, 999999, tz='Europe/Paris')
print(just_before)
```

```
2013-03-31T01:59:59.999999+01:00
```

```
print(just_before.add(microseconds=1))
```

```
2013-03-31T03:00:00+02:00
```

How to Work with Python Date and Time Objects

Python有5个类用于处理日期和时间对象:

- date - 只包含日期信息(年、月、日)。
- time - 指与一天无关的时间(小时、分、秒、微秒)。
- datetime - 组合日期和时间信息。
- timedelta - 表示两个日期或时间之间的差值。
- tzinfo - 一个操作时区信息的抽象类。

这些类是在datetime模块中提供的。所以你需要通过导入所有需要的类，从日期和时间开始你的工作，就像这样:

```
# Starting with the required imports
from datetime import datetime, date, time, timedelta
```

获取当前日期和时间

首先，Python允许你获取当前日期和时间，这一点应该不足为奇。你还可以访问这些对象的独立组件。

要获取当前日期，可以使用 `date` 类的 `today()` 方法，即写入 `date.today()` 。

```
# Getting current date
today = date.today()
days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
print ("Today's date is ", today)
print ("\n", "Year:", today.year,
       "\n", "Month:", today.month,
       "\n","Day:", today.day,
       "\n","Weekday:", today.weekday(),
       "\n","Weekday (name):", days[today.weekday()])
```

```
Today's date is  2019-09-19

 Year: 2019
 Month: 9
 Day: 19
 Weekday: 3
 Weekday (name): Thursday
```

这将只返回今天的日期，没有确切的时间信息。但是，date有一组组件，你可以使用date对象的相应属性分别访问这些组件。

正如你在上面的示例中所看到的，可以访问年、月、日，甚至工作日。

注意，工作日作为索引提供，其中0对应于周一、1到周二、2到周三，依此类推。所以如果你想打印工作日的名字，你有两个选择:

```
创建表示工作日名称的字符串列表;使用date.weekday()方法的输出作为索引访问它的各个元素。
创建索引到对应字符串名称的映射，然后使用date.weekday()返回的索引作为字典的键。
```

Python time Module

```
import time
```

time.time ()

函数的作用是:返回从历元开始经过的秒数。
对于Unix系统，1970年1月1日00:00:00在UTC是epoch(时间开始的点)。

```
import time
seconds = time.time()
print("Seconds since epoch =", seconds)
```

```
Seconds since epoch = 1568896724.376
```

time.ctime ()

函数的作用是:以历元以来的秒为参数，返回一个表示本地时间的字符串。

```
import time
# seconds passed since epoch
seconds = 1545925769.9618232
local_time = time.ctime(seconds)
print("Local time:", local_time)
```

```
Local time: Thu Dec 27 23:49:29 2018
```

time . sleep ()

函数的作用是:在给定的秒数内暂停(延迟)当前线程的执行。

```
import time
print("This is printed immediately.")
time.sleep(2.4)
print("This is printed after 2.4 seconds.")
```

```
This is printed immediately.
This is printed after 2.4 seconds.
```

time.struct_time Class

时间模块中的几个函数，如gmtime()、asctime()等，都需要时间。struct_time对象作为参数或返回它。

Index	Attribute	Values
0	tm_year	0000, ..., 2018, ..., 9999
1	tm_mon	1, 2, ..., 12
2	tm_mday	1, 2, ..., 31
3	tm_hour	0, 1, ..., 23
4	tm_min	0, 1, ..., 59
5	tm_sec	0, 1, ..., 61
6	tm_wday	0, 1, ..., 6; Monday is 0
7	tm_yday	1, 2, ..., 366
8	tm_isdst	0, 1 or -1

time.struct_time 期望它的第一个参数是一个包含9个元素的序列:

```
    temp_time = time.struct_time((2000,11,30,0,0,0,3,335,-1))
print(temp_time)
```

```
time.struct_time(tm_year=2000, tm_mon=11, tm_mday=30, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3,
tm_yday=335, tm_isdst=-1)
```

time.localtime ()

函数的作用是:将历元之后经过的秒数作为参数，并在本地时间内返回struct_time。

```
import time
result = time.localtime(1545925769)
print("result:", result)
print("\nyear:", result.tm_year)
print("tm_hour:", result.tm_hour)
```

```
result: time.struct_time(tm_year=2018, tm_mon=12, tm_mday=27, tm_hour=23, tm_min=49, tm_sec=29,
tm_wday=3, tm_yday=361, tm_isdst=0)

year: 2018
tm_hour: 23
```

如果没有参数或没有传递给localtime()，则使用time()返回的值。

time.gmtime ()

函数的作用是:将历元之后经过的秒数作为参数，并在UTC中返回struct_time。

```
import time
result = time.gmtime(1545925769)
print("result:", result)
print("\nyear:", result.tm_year)
print("tm_hour:", result.tm_hour)
```

```
result: time.struct_time(tm_year=2018, tm_mon=12, tm_mday=27, tm_hour=15, tm_min=49, tm_sec=29,
tm_wday=3, tm_yday=361, tm_isdst=0)

year: 2018
tm_hour: 15
```

如果没有参数或没有传递给gmtime()，则使用time()返回的值。

time.mktime ()

函数的作用是:将struct_time(或包含与struct_time对应的9个元素的元组)作为参数，并在本地时间中返回历元以来的秒数。基本上，它是localtime()的逆函数。

```
import time
t = (2018, 12, 28, 8, 44, 4, 4, 362, 0)
local_time = time.mktime(t)
print("Local time:", local_time)
```

```
Local time: 1545957844.0
```

下面的示例显示了mktime()和localtime()之间的关系。

```
import time
seconds = 1545925769
# returns struct_time
t = time.localtime(seconds)
print("t1: ", t)
# returns seconds from struct_time
s = time.mktime(t)
print("\s:", seconds)
```

```
t1:  time.struct_time(tm_year=2018, tm_mon=12, tm_mday=27, tm_hour=23, tm_min=49, tm_sec=29,
tm_wday=3, tm_yday=361, tm_isdst=0)
\s: 1545925769
```

time.asctime ()

asctime()函数接受struct_time(或包含与struct_time对应的9个元素的元组)作为参数，并返回表示它的字符串。这里有一个例子:

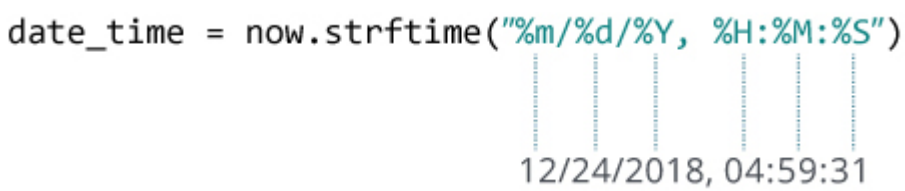
```
import time
t = (2018, 12, 28, 8, 44, 4, 4, 362, 0)
result = time.asctime(t)
print("Result:", result)
```

```
Result: Fri Dec 28 08:44:04 2018
```

time.strftime ()

函数的作用是:将struct_time(或与之对应的元组)作为参数，并根据使用的格式代码返回表示它的字符串。例如,

Python strftime()




```
import time
named_tuple = time.localtime() # get struct_time
time_string = time.strftime("%m/%d/%Y, %H:%M:%S", named_tuple)
print(time_string)
```

```
09/19/2019, 20:53:30
```

%Y, %m, %d, %H etc. 格式化标志.

- %Y - year [0001,..., 2018, 2019,..., 9999]
- %m - month [01, 02, ..., 11, 12]
- %d - day [01, 02, ..., 30, 31]
- %H - hour [00, 01, ..., 22, 23
- %M - month [00, 01, ..., 58, 59]
- %S - second [00, 01, ..., 58, 61]

time.strftime()

time.strptime ()

函数的作用是:解析一个表示时间的字符串并返回struct_time。

```
import time
time_string = "21 June, 2018"
result = time.strptime(time_string, "%d %B, %Y")
print(result)
```

```
time.struct_time(tm_year=2018, tm_mon=6, tm_mday=21, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3,
tm_yday=172, tm_isdst=-1)
```

Python datetime

获取当前日期和时间

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

```
2019-09-19 20:58:02.198000
```

在这里，我们使用import datetime语句导入了datetime模块。

datetime模块中定义的一个类是datetime类。然后，我们使用now()方法创建一个包含当前本地日期和时间的datetime对象。

获得当前日期

```
import datetime
date_object = datetime.date.today()
print(date_object)
```

```
2019-09-19
```

使用dir()函数获得一个包含模块所有属性的列表

```
import datetime
print(dir(datetime))
```

```
['MAXYEAR', 'MINYEAR', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', '_divide_and_round', 'date', 'datetime', 'datetime_CAPI',
'time', 'timedelta', 'timezone', 'tzinfo']
```

datetime模块中常用的类有:

- date Class
- time Class
- datetime Class
- timedelta Class

时间对象来表示日期

```
import datetime
d = datetime.date(2019, 4, 13)
print(d)
```

```
2019-04-13
```

date()是date类的构造函数。构造函数接受三个参数:年、月和日。

```
from datetime import date
a = date(2019, 4, 13)
print(a)
```

```
2019-04-13
```

获得当前日期

可以使用一个名为today()的类方法创建一个包含当前日期的date对象。方法如下:

```
from datetime import date
today = date.today()
print("Current date =", today)
```

```
Current date = 2019-09-19
```

从时间戳获取日期

我们还可以从时间戳创建日期对象。Unix时间戳是一个特定日期到UTC的1970年1月1日之间的秒数。可以使用fromtimestamp()方法将时间戳转换为日期。

```
from datetime import date
timestamp = date.fromtimestamp(1326244364)
print("Date =", timestamp)
```

```
Date = 2012-01-11
```

打印今天的年、月和日

我们可以很容易地从date对象中得到年、月、日、星期等。方法如下:

```
from datetime import date
# date object of today's date
today = date.today()
print("Current year:", today.year)
print("Current month:", today.month)
print("Current day:", today.day)
```

```
Current year: 2019
Current month: 9
Current day: 19
```

datetime.time

从time类实例化的时间对象表示本地时间。

表示时间的时间对象

```
from datetime import time
# time(hour = 0, minute = 0, second = 0)
a = time()
print("a =", a)
# time(hour, minute and second)
b = time(11, 34, 56)
print("b =", b)
# time(hour, minute and second)
c = time(hour = 11, minute = 34, second = 56)
print("c =", c)
# time(hour, minute, second, microsecond)
d = time(11, 34, 56, 234566)
print("d =", d)
```

```
a = 00:00:00
b = 11:34:56
c = 11:34:56
d = 11:34:56.234566
```

打印小时、分钟、秒和微秒

一旦创建了时间对象，就可以轻松地打印它的属性，如hour、minute等。

```
from datetime import time
a = time(11, 34, 56)
print("hour =", a.hour)
print("minute =", a.minute)
print("second =", a.second)
print("microsecond =", a.microsecond)
```

```
hour = 11
minute = 34
second = 56
microsecond = 0
```

注意，我们没有传递微秒参数。因此，它的默认值为0。

datetime.datetime

datetime模块有一个名为dateclass的类，它可以包含date和time对象的信息。

Python datetime对象

```
from datetime import datetime
#datetime(year, month, day)
a = datetime(2018, 11, 28)
print(a)
# datetime(year, month, day, hour, minute, second, microsecond)
b = datetime(2017, 11, 28, 23, 55, 59, 342380)
print(b)
```

```
2018-11-28 00:00:00
2017-11-28 23:55:59.342380
```

datetime()构造函数中的前三个参数year、month和day是必需的。

打印年份、月份、时间、分钟和时间戳

```
from datetime import datetime
a = datetime(2019, 11, 28, 23, 55, 59, 342380)
print("year =", a.year)
print("month =", a.month)
print("hour =", a.hour)
print("minute =", a.minute)
print("timestamp =", a.timestamp())
print('tzinfo:', a.tzinfo)
```

```
year = 2019
month = 11
hour = 23
minute = 55
timestamp = 1574956559.34238
tzinfo: None
```

```
# 最大最小时间范围

import datetime

print('Earliest  :', datetime.time.min)
print('Latest    :', datetime.time.max)
print('Resolution:', datetime.time.resolution)
```

```
Earliest  : 00:00:00
Latest    : 23:59:59.999999
Resolution: 0:00:00.000001
```

```
import datetime
now = datetime.datetime.now()
print (now)
print ("Current date and time : ", now.strftime("%Y-%m-%d %H:%M:%S"))

now_time = now.time().strftime("%H:%M:%S")
print("当前时间不包含日期的:", now_time)
```

```
2019-09-20 09:59:31.496000
Current date and time :  2019-09-20 09:59:31
当前时间不包含日期的: 09:59:31
```

```

```

datetime.timedelta

timedelta对象表示两个日期或时间之间的差。

两个日期和时间的区别

```
from datetime import datetime, date
t1 = date(year = 2018, month = 7, day = 12)
t2 = date(year = 2017, month = 12, day = 23)
t3 = t1 - t2
print("t3 =", t3)

t4 = datetime(year = 2018, month = 7, day = 12, hour = 7, minute = 9, second = 33)
t5 = datetime(year = 2019, month = 6, day = 10, hour = 5, minute = 55, second = 13)
t6 = t4 - t5

print("t6 =", t6)
print("type of t3 =", type(t3))
print("type of t6 =", type(t6))
```

```
t3 = 201 days, 0:00:00
t6 = -333 days, 1:14:20
type of t3 = <class 'datetime.timedelta'>
type of t6 = <class 'datetime.timedelta'>
```

timedelta 两个时间增量对象之间的差异

```
from datetime import timedelta
t1 = timedelta(weeks = 2, days = 5, hours = 1, seconds = 33)
t2 = timedelta(days = 4, hours = 11, minutes = 4, seconds = 54)
t3 = t1 - t2
print("t3 =", t3)
```

```
t3 = 14 days, 13:55:39
```

```
import time
import datetime

d1 = datetime.datetime(2018, 10, 18)
d2 = datetime.datetime(2017, 12, 31)

print ( (d1 - d2).days)
print('\n')

## 当前日期
curdate = datetime.date.today()
print ("curdate = ", curdate)
```

```
291
```

```
curdate =  2019-09-20
```

打印负时间增量对象

```
from datetime import timedelta
t1 = timedelta(seconds = 33)
t2 = timedelta(seconds = 54)
t3 = t1 - t2

print("t3 =", t3)
print("t3 =", abs(t3))
```

```
t3 = -1 day, 23:59:39
t3 = 0:00:21
```

持续时间(秒)

你可以使用total_seconds()方法获得timedelta对象中的总秒数。

```
from datetime import timedelta
t = timedelta(days = 5, hours = 1, seconds = 33, microseconds = 233423)
print("total seconds =", t.total_seconds())
```

```
total seconds = 435633.233423
```

你还可以使用+运算符找到两个日期和时间的和。此外，还可以用整数和浮点数乘和除时间增量对象。

Python datetime格式

日期和时间的表示方式可能在不同的地方、组织等不同。在美国使用mm/dd/yyyy更为常见，而在英国使用dd/mm/yyyy更为常见。

Python有strftime()和strptime()方法来处理这个问题。

** Python strftime() - datetime对象到字符串**

strftime()方法是在date、datetime和time类下面定义的。该方法从给定的日期、日期时间或时间对象创建格式化字符串。

使用strftime()格式化日期

```
from datetime import datetime
# current date and time
now = datetime.now()
t = now.strftime("%H:%M:%S")
print("time:", t)
s1 = now.strftime("%m/%d/%Y, %H:%M:%S")
# mm/dd/YY H:M:S format
print("s1:", s1)
s2 = now.strftime("%d/%m/%Y, %H:%M:%S")
# dd/mm/YY H:M:S format
print("s2:", s2)
```

```
time: 21:12:55
s1: 09/19/2019, 21:12:55
s2: 19/09/2019, 21:12:55
```

这里，%Y、%m、%d、%H等是格式标志。strftime()方法接受一个或多个格式代码，并根据它返回格式化的字符串。

```
%Y - year [0001,..., 2018, 2019,..., 9999]
%m - month [01, 02, ..., 11, 12]
%d - day [01, 02, ..., 30, 31]
%H - hour [00, 01, ..., 22, 23
%M - minute [00, 01, ..., 58, 59]
%S - second [00, 01, ..., 58, 59]
```

更多使用见Python strftime().

Python strptime()——字符串到datetime

方法的作用是:根据给定的字符串(表示日期和时间)创建一个datetime对象。

strptime()

```
from datetime import datetime
date_string = "21 June, 2018"
print("date_string =", date_string)
date_object = datetime.strptime(date_string, "%d %B, %Y")
print("date_object =", date_object)
```

```
date_string = 21 June, 2018
date_object = 2018-06-21 00:00:00
```

strptime()方法有两个参数:
表示与第一个参数等价的日期和时间格式代码的字符串

另外，%d、%B和%Y格式代码分别用于日、月(全称)和年。

访问Python strptime()了解更多信息。

假设你正在处理一个项目，需要根据其时区显示日期和时间。我们建议你使用第三方pytZ module，而不是自己处理时区。

```
from datetime import datetime
import pytz
local = datetime.now()
print("Local:", local.strftime("%m/%d/%Y, %H:%M:%S"))
tz_NY = pytz.timezone('America/New_York')
datetime_NY = datetime.now(tz_NY)
print("NY:", datetime_NY.strftime("%m/%d/%Y, %H:%M:%S"))
tz_London = pytz.timezone('Europe/London')
datetime_London = datetime.now(tz_London)
print("London:", datetime_London.strftime("%m/%d/%Y, %H:%M:%S"))
```

```
Local: 09/19/2019, 21:18:02
NY: 09/19/2019, 09:18:02
London: 09/19/2019, 14:18:02
```

Python Datetime

当前时间

```
import datetime

x = datetime.datetime.now()
print(x)
```

```
2019-09-19 21:18:55.756000
```

指定格式

```
import datetime

x = datetime.datetime.now()

print(x.year)
print(x.strftime("%A"))
```

```
2019
Thursday
```

创建时间对象

```
import datetime

x = datetime.datetime(2020, 5, 17)

print(x)
```

```
2020-05-17 00:00:00
```

datetime()类还接受时间和时区(小时、分钟、秒、微秒、tzone)的参数，但它们是可选的，默认值为0(时区没有参数)。

strftime()方法

datetime对象具有将日期对象格式化为可读字符串的方法。

该方法被调用strftime()，并接受一个参数format来指定返回字符串的格式:

```
import datetime

x = datetime.datetime(2018, 6, 1)

print(x.strftime("%B"))
```

```
June
```

所有规范格式化的参考资料:

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%

Mastering Python Datetime (With Examples)

获得当前日期时间

```
# curr_date_time.py

from datetime import datetime

curr_datetime = datetime.now()
curr_date = curr_datetime.date()
curr_time = curr_datetime.time()

print(f"The current date time is {curr_datetime}")
print(f"Today's date is {curr_date}")
print(f"Current time is {curr_time}")
```

```
The current date time is 2019-09-19 21:27:59.565000
Today's date is 2019-09-19
Current time is 21:27:59.565000
```

datetime.now()提供当前的本地日期和时间。当我们同时使用date()和time()方法时，它将分别返回当前日期和时间。

将日期和时间分割为多个组件

```
from datetime import datetime

curr_datetime = datetime.now()

curr_year = curr_datetime.year
curr_month = curr_datetime.month
curr_day = curr_datetime.day

curr_hour = curr_datetime.hour
curr_min = curr_datetime.minute
curr_sec = curr_datetime.second

print(f"The current date time is {curr_datetime}")

print(f"The year is {curr_year}")
print(f"The month is {curr_month}")
print(f"The day is {curr_day}")

print(f"Current hour of the day is {curr_hour}")
print(f"Current minute of the day is {curr_min}")
print(f"Current seconds are {curr_sec}")
```

```
The current date time is 2019-09-19 21:29:48.821000
The year is 2019
The month is 9
The day is 19
Current hour of the day is 21
Current minute of the day is 29
Current seconds are 48
```

创建手动日期和时间对象

我们可以使用python datetime模块在python应用程序中创建手工的日期和时间对象。假设我们想为2017年1月1日创建一个date对象。为此，我们可以使用datetime模块中的date类。通过年、月、日的参数。

```
from datetime import date

req_date = date(year=2019,
                month=10,
                day=1)

print(req_date)
```

```
2019-10-01
```

我们还可以使用datetime模块中的datetime类来创建上面的date对象。但是，datetime类包含date和time对象的信息。这就是datetime类的样子。

```
class datetime.datetime(year, month, day, hour=0, minute=0, second=0, microsecond=0,
tzinfo=None, *, fold=0)
```

```
from datetime import datetime

req_date = datetime(year=2017,
                month=1,
                day=1)

print(req_date)
```

```
2017-01-01 00:00:00
```

如果我们这样创建一个date对象，我们还将获得默认时间信息，该信息将被设置为0000小时或午夜12点。

假设我们希望一天的时间是上午9:30:36。为了补充这些信息，我们还必须提供小时、分钟和秒的信息。

```
from datetime import datetime

req_date = datetime(year=2017,
                month=1,
                day=1,
                hour=9,
                minute=30,
                second=36)

print(req_date)
```

```
2017-01-01 09:30:36
```

使用上面的逻辑，我们还可以通过使用python datetime模块中的time类来创建时间对象。在这种情况下，我们只需要提供小时、分钟和秒的详细信息。注意，所有这些都是可选的。如果您不提供任何参数，您将收到一个指向0000 hours或12am的time对象。

```
from datetime import time


req_time = time(
                hour=9,
                minute=30,
                second=36)

null_time = time()

print(req_time)
print(null_time)
```

```
09:30:36
00:00:00
```

Timedelta

timedelta对象提供两个日期或时间之间的差异。一般语法是这样的。

```
class datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0,
weeks=0)
```

所有参数都是可选的，默认值为0，你可以传入正值和负值。

```
from datetime import datetime,timedelta

# Get the current date and time
curr_date_time = datetime.now()
print(curr_date_time)


time_diff = timedelta(days=2,
                      hours=3,
                      minutes=15)

# Add time diff to the current time
req_date_time = curr_date_time + time_diff

print(req_date_time)
```

```
2019-09-19 21:35:21.238000
2019-09-22 00:50:21.238000
```

2019年6月1日、9日之前的第1周和第3天是什么时间?

```
from datetime import datetime,timedelta

# Get the base date and time
base_date_time = datetime(year=2019,
                          month=6,
                          day=1,
                          hour=9)
print(base_date_time)
```

```
time_diff = timedelta(weeks=-1,
                      days=-3)

# Subtract time diff to the current time
req_date_time = base_date_time + time_diff

print(req_date_time)
```

```
2019-06-01 09:00:00
2019-05-22 09:00:00
```

datetime.strftime ()

必须将日期和时间复制到JSON中吗?或者可以使用日期和时间作为后缀来命名日志文件?为了执行这些任务，你需要将python datetime对象转换为其相关且可用的字符串表示形式。datetime类中的strftime方法允许您创建datetime对象的字符串表示形式。为了做到这一点，你需要提供两个参数。

```
一个datetime对象
显式格式化字符串
```

```
from datetime import datetime

# Get the base date and time
base_date_time = datetime.now()
print(base_date_time)

# Format it to January 1, 1970
print(datetime.strftime(base_date_time, "%B %d, %Y"))

# Print the day of the week with time in AM/PM
print(datetime.strftime(base_date_time, "%A, %I:%M:%S %p"))
```

```
2019-09-19 21:37:39.754000
September 19, 2019
Thursday, 09:37:39 PM
```

一个额外的小技巧

07年6月的领先0会影响到你吗?可以在格式化字符串中使用-删除前导0。

```
from datetime import datetime

# Get the base date and time
base_date_time = datetime.now()

# Format it to January 1, 1970
print(datetime.strftime(base_date_time, "%B %d, %Y"))
```

```
September 19, 2019
```

datetime.strptime()

```
from datetime import datetime

dt_string = "January 1, 2019"

dt_object = datetime.strptime(dt_string, "%B %d, %Y")
print(dt_object)
```

```
2019-01-01 00:00:00
```

Datetime到Unix时间戳

Unix时间戳是从纪元到任何日期和时间的秒数。Epoch是1970年1月1日的通用术语。我们可以使用python datetime模块将任何datetime对象转换为对应的Unix时间戳并返回。

```
from datetime import datetime

# convert current datetime object to timestamp
tstamp = datetime.now().timestamp()
print(tstamp)

# convert timestamp back to datetime object
dt_object = datetime.fromtimestamp(tstamp)
print(dt_object)
```

```
1568900466.084
2019-09-19 21:41:06.084000
```

时区处理

到目前为止，我们一直在处理简单的datetime对象。

什么是简单的datetime对象?它没有任何关于时区的信息。你可以很容易地检查datetime对象是否简单或不使用tzinfo。如果它是一个简单的datetime对象，下面的代码应该不会返回任何值。

```
from datetime import datetime

dt_object = datetime(year=2019,
                     month=1,
                     day=1)

print(dt_object.tzinfo)
```

```
None
```

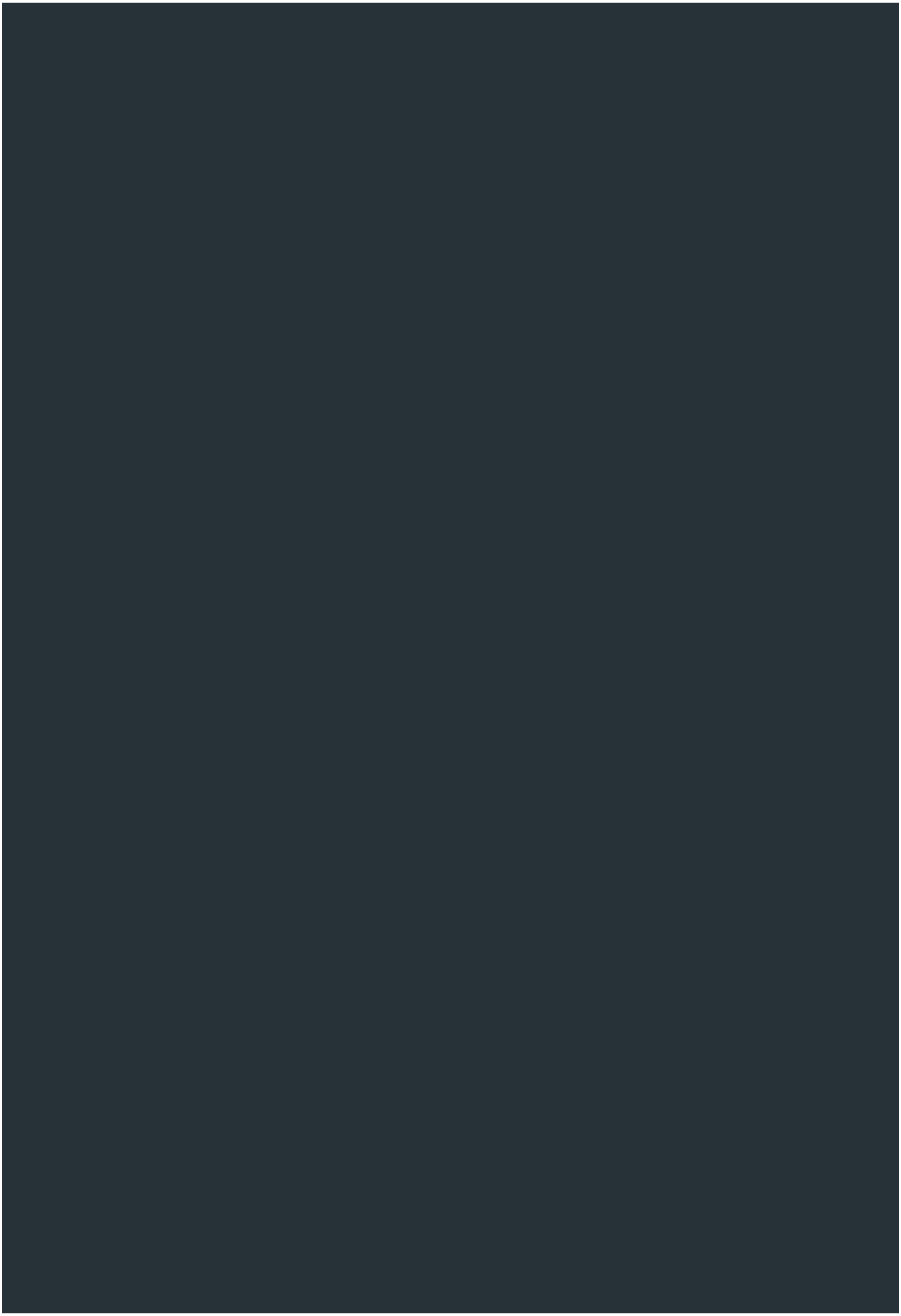
这是python datetime模块的缺点之一。无法使用python datetime模块中的任何类或方法使datetime对象知道时区。这并不意味着你不能实现它。可以使用pytz库向应用程序添加时区信息。 那么，如何找出正确的时区名称呢?

你可以在这里参考所有时区数据库名称的列表。

你还可以使用pytz库生成所有时区的列表。

```
import pytz

print(pytz.all_timezones)
```



```
['Africa/Abidjan', 'Africa/Accra', 'Africa/Addis_Ababa', 'Africa/Algiers', 'Africa/Asmara',
'Africa/Asmera', 'Africa/Bamako', 'Africa/Bangui', 'Africa/Banjul', 'Africa/Bissau',
'Africa/Blantyre', 'Africa/Brazzaville', 'Africa/Bujumbura', 'Africa/Cairo',
'Africa/Casablanca', 'Africa/Ceuta', 'Africa/Conakry', 'Africa/Dakar', 'Africa/Dar_es_Salaam',
'Africa/Djibouti', 'Africa/Douala', 'Africa/El_Aaiun', 'Africa/Freetown', 'Africa/Gaborone',
'Africa/Harare', 'Africa/Johannesburg', 'Africa/Juba', 'Africa/Kampala', 'Africa/Khartoum',
'Africa/Kigali', 'Africa/Kinshasa', 'Africa/Lagos', 'Africa/Libreville', 'Africa/Lome',
'Africa/Luanda', 'Africa/Lubumbashi', 'Africa/Lusaka', 'Africa/Malabo', 'Africa/Maputo',
'Africa/Maseru', 'Africa/Mbabane', 'Africa/Mogadishu', 'Africa/Monrovia', 'Africa/Nairobi',
'Africa/Ndjamena', 'Africa/Niamey', 'Africa/Nouakchott', 'Africa/Ouagadougou', 'Africa/Porto-
Novo', 'Africa/Sao_Tome', 'Africa/Timbuktu', 'Africa/Tripoli', 'Africa/Tunis',
'Africa/Windhoek', 'America/Adak', 'America/Anchorage', 'America/Anguilla', 'America/Antigua',
'America/Araguaina', 'America/Argentina/Buenos_Aires', 'America/Argentina/Catamarca',
'America/Argentina/ComodRivadavia', 'America/Argentina/Cordoba', 'America/Argentina/Jujuy',
'America/Argentina/La_Rioja', 'America/Argentina/Mendoza', 'America/Argentina/Rio_Gallegos',
'America/Argentina/Salta', 'America/Argentina/San_Juan', 'America/Argentina/San_Luis',
'America/Argentina/Tucuman', 'America/Argentina/Ushuaia', 'America/Aruba', 'America/Asuncion',
'America/Atikokan', 'America/Atka', 'America/Bahia', 'America/Bahia_Banderas',
'America/Barbados', 'America/Belem', 'America/Belize', 'America/Blanc-Sablon',
'America/Boa_Vista', 'America/Bogota', 'America/Boise', 'America/Buenos_Aires',
'America/Cambridge_Bay', 'America/Campo_Grande', 'America/Cancun', 'America/Caracas',
'America/Catamarca', 'America/Cayenne', 'America/Cayman', 'America/Chicago',
'America/Chihuahua', 'America/Coral_Harbour', 'America/Cordoba', 'America/Costa_Rica',
'America/Creston', 'America/Cuiaba', 'America/Curacao', 'America/Danmarkshavn',
'America/Dawson', 'America/Dawson_Creek', 'America/Denver', 'America/Detroit',
'America/Dominica', 'America/Edmonton', 'America/Eirunepe', 'America/El_Salvador',
'America/Ensenada', 'America/Fort_Nelson', 'America/Fort_Wayne', 'America/Fortaleza',
'America/Glace_Bay', 'America/Godthab', 'America/Goose_Bay', 'America/Grand_Turk',
'America/Grenada', 'America/Guadeloupe', 'America/Guatemala', 'America/Guayaquil',
'America/Guyana', 'America/Halifax', 'America/Havana', 'America/Hermosillo',
'America/Indiana/Indianapolis', 'America/Indiana/Knox', 'America/Indiana/Marengo',
'America/Indiana/Petersburg', 'America/Indiana/Tell_City', 'America/Indiana/Vevay',
'America/Indiana/Vincennes', 'America/Indiana/Winamac', 'America/Indianapolis',
'America/Inuvik', 'America/Iqaluit', 'America/Jamaica', 'America/Jujuy', 'America/Juneau',
'America/Kentucky/Louisville', 'America/Kentucky/Monticello', 'America/Knox_IN',
'America/Kralendijk', 'America/La_Paz', 'America/Lima', 'America/Los_Angeles',
'America/Louisville', 'America/Lower_Princes', 'America/Maceio', 'America/Managua',
'America/Manaus', 'America/Marigot', 'America/Martinique', 'America/Matamoros',
'America/Mazatlan', 'America/Mendoza', 'America/Menominee', 'America/Merida',
'America/Metlakatla', 'America/Mexico_City', 'America/Miquelon', 'America/Moncton',
'America/Monterrey', 'America/Montevideo', 'America/Montreal', 'America/Montserrat',
'America/Nassau', 'America/New_York', 'America/Nipigon', 'America/Nome', 'America/Noronha',
'America/North_Dakota/Beulah', 'America/North_Dakota/Center', 'America/North_Dakota/New_Salem',
'America/Ojinaga', 'America/Panama', 'America/Pangnirtung', 'America/Paramaribo',
'America/Phoenix', 'America/Port-au-Prince', 'America/Port_of_Spain', 'America/Porto_Acre',
'America/Porto_Velho', 'America/Puerto_Rico', 'America/Punta_Arenas', 'America/Rainy_River',
'America/Rankin_Inlet', 'America/Recife', 'America/Regina', 'America/Resolute',
'America/Rio_Branco', 'America/Rosario', 'America/Santa_Isabel', 'America/Santarem',
'America/Santiago', 'America/Santo_Domingo', 'America/Sao_Paulo', 'America/Scoresbysund',
'America/Shiprock', 'America/Sitka', 'America/St_Barthelemy', 'America/St_Johns',
'America/St_Kitts', 'America/St_Lucia', 'America/St_Thomas', 'America/St_Vincent',
'America/Swift_Current', 'America/Tegucigalpa', 'America/Thule', 'America/Thunder_Bay',
'America/Tijuana', 'America/Toronto', 'America/Tortola', 'America/Vancouver', 'America/Virgin',
'America/Whitehorse', 'America/Winnipeg', 'America/Yakutat', 'America/Yellowknife',
'Antarctica/Casey', 'Antarctica/Davis', 'Antarctica/DumontDUrville', 'Antarctica/Macquarie',
'Antarctica/Mawson', 'Antarctica/McMurdo', 'Antarctica/Palmer', 'Antarctica/Rothera',
'Antarctica/South_Pole', 'Antarctica/Syowa', 'Antarctica/Troll', 'Antarctica/Vostok',
'Arctic/Longyearbyen', 'Asia/Aden', 'Asia/Almaty', 'Asia/Amman', 'Asia/Anadyr', 'Asia/Aqtau',
'Asia/Aqtobe', 'Asia/Ashgabat', 'Asia/Ashkhabad', 'Asia/Atyrau', 'Asia/Baghdad', 'Asia/Bahrain',
'Asia/Baku', 'Asia/Bangkok', 'Asia/Barnaul', 'Asia/Beirut', 'Asia/Bishkek', 'Asia/Brunei',
```

```
'Asia/Calcutta', 'Asia/Chita', 'Asia/Choibalsan', 'Asia/Chongqing', 'Asia/Chungking',
'Asia/Colombo', 'Asia/Dacca', 'Asia/Damascus', 'Asia/Dhaka', 'Asia/Dili', 'Asia/Dubai',
'Asia/Dushanbe', 'Asia/Famagusta', 'Asia/Gaza', 'Asia/Harbin', 'Asia/Hebron',
'Asia/Ho_Chi_Minh', 'Asia/Hong_Kong', 'Asia/Hovd', 'Asia/Irkutsk', 'Asia/Istanbul',
'Asia/Jakarta', 'Asia/Jayapura', 'Asia/Jerusalem', 'Asia/Kabul', 'Asia/Kamchatka',
'Asia/Karachi', 'Asia/Kashgar', 'Asia/Kathmandu', 'Asia/Katmandu', 'Asia/Khandyga',
'Asia/Kolkata', 'Asia/Krasnoyarsk', 'Asia/Kuala_Lumpur', 'Asia/Kuching', 'Asia/Kuwait',
'Asia/Macao', 'Asia/Macau', 'Asia/Magadan', 'Asia/Makassar', 'Asia/Manila', 'Asia/Muscat',
'Asia/Nicosia', 'Asia/Novokuznetsk', 'Asia/Novosibirsk', 'Asia/Omsk', 'Asia/Oral',
'Asia/Phnom_Penh', 'Asia/Pontianak', 'Asia/Pyongyang', 'Asia/Qatar', 'Asia/Qyzylorda',
'Asia/Rangoon', 'Asia/Riyadh', 'Asia/Saigon', 'Asia/Sakhalin', 'Asia/Samarkand', 'Asia/Seoul',
'Asia/Shanghai', 'Asia/Singapore', 'Asia/Srednekolymsk', 'Asia/Taipei', 'Asia/Tashkent',
'Asia/Tbilisi', 'Asia/Tehran', 'Asia/Tel_Aviv', 'Asia/Thimbu', 'Asia/Thimphu', 'Asia/Tokyo',
'Asia/Tomsk', 'Asia/Ujung_Pandang', 'Asia/Ulaanbaatar', 'Asia/Ulan_Bator', 'Asia/Urumqi',
'Asia/Ust-Nera', 'Asia/Vientiane', 'Asia/Vladivostok', 'Asia/Yakutsk', 'Asia/Yangon',
'Asia/Yekaterinburg', 'Asia/Yerevan', 'Atlantic/Azores', 'Atlantic/Bermuda', 'Atlantic/Canary',
'Atlantic/Cape_Verde', 'Atlantic/Faeroe', 'Atlantic/Faroe', 'Atlantic/Jan_Mayen',
'Atlantic/Madeira', 'Atlantic/Reykjavik', 'Atlantic/South_Georgia', 'Atlantic/St_Helena',
'Atlantic/Stanley', 'Australia/ACT', 'Australia/Adelaide', 'Australia/Brisbane',
'Australia/Broken_Hill', 'Australia/Canberra', 'Australia/Currie', 'Australia/Darwin',
'Australia/Eucla', 'Australia/Hobart', 'Australia/LHI', 'Australia/Lindeman',
'Australia/Lord_Howe', 'Australia/Melbourne', 'Australia/NSW', 'Australia/North',
'Australia/Perth', 'Australia/Queensland', 'Australia/South', 'Australia/Sydney',
'Australia/Tasmania', 'Australia/Victoria', 'Australia/West', 'Australia/Yancowinna',
'Brazil/Acre', 'Brazil/DeNoronha', 'Brazil/East', 'Brazil/West', 'CET', 'CST6CDT',
'Canada/Atlantic', 'Canada/Central', 'Canada/Eastern', 'Canada/Mountain', 'Canada/Newfoundland',
'Canada/Pacific', 'Canada/Saskatchewan', 'Canada/Yukon', 'Chile/Continental',
'Chile/EasterIsland', 'Cuba', 'EET', 'EST', 'EST5EDT', 'Egypt', 'Eire', 'Etc/GMT', 'Etc/GMT+0',
'Etc/GMT+1', 'Etc/GMT+10', 'Etc/GMT+11', 'Etc/GMT+12', 'Etc/GMT+2', 'Etc/GMT+3', 'Etc/GMT+4',
'Etc/GMT+5', 'Etc/GMT+6', 'Etc/GMT+7', 'Etc/GMT+8', 'Etc/GMT+9', 'Etc/GMT-0', 'Etc/GMT-1',
'Etc/GMT-10', 'Etc/GMT-11', 'Etc/GMT-12', 'Etc/GMT-13', 'Etc/GMT-14', 'Etc/GMT-2', 'Etc/GMT-3',
'Etc/GMT-4', 'Etc/GMT-5', 'Etc/GMT-6', 'Etc/GMT-7', 'Etc/GMT-8', 'Etc/GMT-9', 'Etc/GMT0',
'Etc/Greenwich', 'Etc/UCT', 'Etc/UTC', 'Etc/Universal', 'Etc/Zulu', 'Europe/Amsterdam',
'Europe/Andorra', 'Europe/Astrakhan', 'Europe/Athens', 'Europe/Belfast', 'Europe/Belgrade',
'Europe/Berlin', 'Europe/Bratislava', 'Europe/Brussels', 'Europe/Bucharest', 'Europe/Budapest',
'Europe/Busingen', 'Europe/Chisinau', 'Europe/Copenhagen', 'Europe/Dublin', 'Europe/Gibraltar',
'Europe/Guernsey', 'Europe/Helsinki', 'Europe/Isle_of_Man', 'Europe/Istanbul', 'Europe/Jersey',
'Europe/Kaliningrad', 'Europe/Kiev', 'Europe/Kirov', 'Europe/Lisbon', 'Europe/Ljubljana',
'Europe/London', 'Europe/Luxembourg', 'Europe/Madrid', 'Europe/Malta', 'Europe/Mariehamn',
'Europe/Minsk', 'Europe/Monaco', 'Europe/Moscow', 'Europe/Nicosia', 'Europe/Oslo',
'Europe/Paris', 'Europe/Podgorica', 'Europe/Prague', 'Europe/Riga', 'Europe/Rome',
'Europe/Samara', 'Europe/San_Marino', 'Europe/Sarajevo', 'Europe/Saratov', 'Europe/Simferopol',
'Europe/Skopje', 'Europe/Sofia', 'Europe/Stockholm', 'Europe/Tallinn', 'Europe/Tirane',
'Europe/Tiraspol', 'Europe/Ulyanovsk', 'Europe/Uzhgorod', 'Europe/Vaduz', 'Europe/Vatican',
'Europe/Vienna', 'Europe/Vilnius', 'Europe/Volgograd', 'Europe/Warsaw', 'Europe/Zagreb',
'Europe/Zaporozhye', 'Europe/Zurich', 'GB', 'GB-Eire', 'GMT', 'GMT+0', 'GMT-0', 'GMT0',
'Greenwich', 'HST', 'Hongkong', 'Iceland', 'Indian/Antananarivo', 'Indian/Chagos',
'Indian/Christmas', 'Indian/Cocos', 'Indian/Comoro', 'Indian/Kerguelen', 'Indian/Mahe',
'Indian/Maldives', 'Indian/Mauritius', 'Indian/Mayotte', 'Indian/Reunion', 'Iran', 'Israel',
'Jamaica', 'Japan', 'Kwajalein', 'Libya', 'MET', 'MST', 'MST7MDT', 'Mexico/BajaNorte',
'Mexico/BajaSur', 'Mexico/General', 'NZ', 'NZ-CHAT', 'Navajo', 'PRC', 'PST8PDT', 'Pacific/Apia',
'Pacific/Auckland', 'Pacific/Bougainville', 'Pacific/Chatham', 'Pacific/Chuuk',
'Pacific/Easter', 'Pacific/Efate', 'Pacific/Enderbury', 'Pacific/Fakaofo', 'Pacific/Fiji',
'Pacific/Funafuti', 'Pacific/Galapagos', 'Pacific/Gambier', 'Pacific/Guadalcanal',
'Pacific/Guam', 'Pacific/Honolulu', 'Pacific/Johnston', 'Pacific/Kiritimati', 'Pacific/Kosrae',
'Pacific/Kwajalein', 'Pacific/Majuro', 'Pacific/Marquesas', 'Pacific/Midway', 'Pacific/Nauru',
'Pacific/Niue', 'Pacific/Norfolk', 'Pacific/Noumea', 'Pacific/Pago_Pago', 'Pacific/Palau',
'Pacific/Pitcairn', 'Pacific/Pohnpei', 'Pacific/Ponape', 'Pacific/Port_Moresby',
'Pacific/Rarotonga', 'Pacific/Saipan', 'Pacific/Samoa', 'Pacific/Tahiti', 'Pacific/Tarawa',
'Pacific/Tongatapu', 'Pacific/Truk', 'Pacific/Wake', 'Pacific/Wallis', 'Pacific/Yap', 'Poland',
```

```
'Portugal', 'ROC', 'ROK', 'Singapore', 'Turkey', 'UCT', 'US/Alaska', 'US/Aleutian',
'US/Arizona', 'US/Central', 'US/East-Indiana', 'US/Eastern', 'US/Hawaii', 'US/Indiana-Starke',
'US/Michigan', 'US/Mountain', 'US/Pacific', 'US/Samoa', 'UTC', 'Universal', 'W-SU', 'WET',
'Zulu']
```

以上任何一种方法都应该帮助您为你的需求选择正确的时区。一旦您有了时区的名称，你就可以将它包含在您的应用程序中。

```
from datetime import datetime
import pytz

dt = datetime.now()
readable_dt = datetime.strftime(dt, "%d/%m/%Y, %H:%M:%S")
print(readable_dt)

tz_India = pytz.timezone("Asia/Calcutta")
dt_India = datetime.now(tz_India)
readable_dt_India = datetime.strftime(dt_India, "%d/%m/%Y, %H:%M:%S")
print(readable_dt_India)
```

```
19/09/2019, 21:44:01
19/09/2019, 19:14:01
```

How to Format Dates in Python

```
import datetime

t = datetime.time(1, 10, 20, 13)
print(t)
```

```
01:10:20.000013
```

```
print('hour:', t.hour)
print('Minutes:', t.minute)
print('Seconds:', t.second)
print('Microsecond:', t.microsecond)
```

```
hour: 1
Minutes: 10
Seconds: 20
Microsecond: 13
```

```
import datetime

today = datetime.date.today()
print(today)
```

```
2019-09-19
```

```
print('ctime:', today.ctime())
```

```
ctime: Thu Sep 19 00:00:00 2019
```

```
print('Year:', today.year)
print('Month:', today.month)
print('Day :', today.day)
```

```
Year: 2019
Month: 9
Day : 19
```

```
import datetime

x = datetime.datetime(2018, 9, 15)

print(x.strftime("%b %d %Y %H:%M:%S"))
```

```
Sep 15 2018 00:00:00
```

```
import datetime

x = datetime.datetime(2018, 9, 15, 12, 45, 35)

print(x.strftime("%b %d %Y %H:%M:%S"))
```

```
Sep 15 2018 12:45:35
```

```
from datetime import datetime

str = '9-15-18'
date_object = datetime.strptime(str, '%m-%d-%y')

print(date_object)
```

```
2018-09-15 00:00:00
```

```
from time import gmtime, strftime

# using simple format of showing time
s = strftime("%a, %d %b %Y %H:%M:%S + 1010", gmtime())
print("Example 1:", s)

print()

# only chnge in this is the full names
# and the representation
s = strftime("%A, %D %B %Y %H:%M:%S + 0000", gmtime())
print("Example 2:", s)

print()

# this will show you the preferd date time format
s = strftime("%c")
print("Example 3:", s)

print()

# this will tell about the centuries
s = strftime("%C")
print("Example 4:", s)

print()

# MOTY: month of the year
# DOTY: Day of the year
# Simple representation
# % n - new line
s = strftime("%A, %D %B %Y, %r")
print("Example 5:", s)

print()

# % R - time in 24 hour notation
s = strftime(" %R ")
print("Example 6:", s)

print()

# % H - hour, using a 24-hour clock (00 to 23) in Example 1, 2, 3
# % I - hour, using a 12-hour clock (01 to 12)
s = strftime("%a, %d %b %Y %I:%M:%S + 0000", gmtime())
print("Example 7:", s)

print()

# % T - current time, equal to % H:% M:% S
s = strftime("%r, %T ", gmtime())
print("Example 8:", s)

print()
```

```
# % u an % U use (see difference)
s = strftime("%r, %u, %U")
print("Example 9:", s)

print()

# use of % V, % W, % w
s = strftime("%r, %V, %W, %w")
print("Example 10:", s)

print()

# use of % x, % X, % y, % Y
s = strftime("%x, %X, %y, %Y")
print("Example 11:", s)

print()

# # use of % Z, % z
# s = strftime("%r, %z, %Z")
# print("Example 12:", s)
```

```
Example 1: Thu, 19 Sep 2019 13:57:16 + 1010

Example 2: Thursday, 09/19/19 September 2019 13:57:16 + 0000

Example 3: Thu Sep 19 21:57:16 2019

Example 4: 20

Example 5: Thursday, 09/19/19 September 2019, 09:57:16 PM

Example 6:  21:57

Example 7: Thu, 19 Sep 2019 01:57:16 + 0000

Example 8: 01:57:16 PM, 13:57:16

Example 9: 09:57:16 PM, 4, 37

Example 10: 09:57:16 PM, 38, 37, 4

Example 11: 09/19/19, 21:57:16, 19, 2019
```

Python, Datum und Zeit

```
from datetime import date
x = date(2019, 12, 19)
print(x)
```

```
2019-12-19
```

最大最小时间

```
from datetime import date
print(date.min)
print(date.max)
```

```
0001-01-01
9999-12-31
```

```
x = date(1, 1, 1) # 1. Januar 1
print(x.toordinal())
x = date(1, 1, 2) # 2. Januar 1
print(x.toordinal())
print(x.today())
print(x.today().toordinal())
```

```
1
2
2019-09-19
737321
```

```
print(date.fromordinal(726952))
```

```
1991-04-30
```

```
print(x.weekday())
print(date.today())

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

```
1
2019-09-19
2
1
1
```

```
from datetime import time
t = time(15, 6, 23)
print(t)
```

```
15:06:23
```

```
print(time.min)
print(time.max)
```

```
00:00:00
23:59:59.999999
```

```
print(t.hour, t.minute, t.second)
```

```
15 6 23
```

```
t = t.replace(hour=11, minute=59)
print(t)
```

```
11:59:23
```

```
print(x.ctime())
```

```
Tue Jan  2 00:00:00 0001
```

```
from datetime import datetime
t = datetime(2017, 4, 19, 16, 31, 0)
print(t)
```

```
2017-04-19 16:31:00
```

```
print(t.tzinfo == None)
```

```
True
```

```
from datetime import datetime
import pytz
t = datetime.now(pytz.utc)
print(t)
```

```
2019-09-19 14:09:18.751000+00:00
```

```
print(t.tzinfo, t.tzinfo.utcoffset(t))
```

```
UTC 0:00:00
```

```
from datetime import datetime, timedelta as delta
ndays = 15
start = datetime(1991, 4, 30)
dates = [start - delta(days=x) for x in range(0, ndays)]
dates
```

```
[datetime.datetime(1991, 4, 30, 0, 0),
 datetime.datetime(1991, 4, 29, 0, 0),
 datetime.datetime(1991, 4, 28, 0, 0),
 datetime.datetime(1991, 4, 27, 0, 0),
 datetime.datetime(1991, 4, 26, 0, 0),
 datetime.datetime(1991, 4, 25, 0, 0),
 datetime.datetime(1991, 4, 24, 0, 0),
 datetime.datetime(1991, 4, 23, 0, 0),
 datetime.datetime(1991, 4, 22, 0, 0),
 datetime.datetime(1991, 4, 21, 0, 0),
 datetime.datetime(1991, 4, 20, 0, 0),
 datetime.datetime(1991, 4, 19, 0, 0),
 datetime.datetime(1991, 4, 18, 0, 0),
 datetime.datetime(1991, 4, 17, 0, 0),
 datetime.datetime(1991, 4, 16, 0, 0)]
```

```
from datetime import datetime
delta = datetime(1993, 12, 14) - datetime(1991, 4, 30)
print(delta, type(delta))
```

```
959 days, 0:00:00 <class 'datetime.timedelta'>
```

```
print(delta.days)
```

```
959
```

```
t1 = datetime(2017, 1, 31, 14, 17)
t2 = datetime(2015, 12, 15, 16, 59)
delta = t1 - t2
print(delta.days, delta.seconds)
```

```
412 76680
```

```
from datetime import datetime, timedelta
d1 = datetime(1991, 4, 30)
d2 = d1 + timedelta(10)
print(d2)
print(d2 - d1)
d3 = d1 - timedelta(100)
print(d3)
d4 = d1 - 2 * timedelta(50)
print(d4)
```

```
1991-05-10 00:00:00
10 days, 0:00:00
1991-01-20 00:00:00
1991-01-20 00:00:00
```

```
from datetime import datetime, timedelta
d1 = datetime(1991, 4, 30)
d2 = d1 + timedelta(10,100)
print(d2)
print(d2 - d1)
```

```
1991-05-10 00:01:40
10 days, 0:01:40
```

```
print(d1.strftime('%Y-%m-%d'))
print("Wochentag: " + d1.strftime('%a'))
print("Wochentag ausgeschrieben: " + d1.strftime('%A'))
# Weekday as a decimal number, where 0 is Sunday
# and 6 is Saturday
print("Wochentag als Dezimalzahl: " + d1.strftime('%w'))
```

```
1991-04-30
Wochentag: Tue
Wochentag ausgeschrieben: Tuesday
Wochentag als Dezimalzahl: 2
```

```
# Day of the month as a zero-padded decimal number.
```

```
# 01, 02, ..., 31
print(d1.strftime('%d'))
# Month as locale's abbreviated name.
# Jan, Feb, ..., Dec (en_US);
# Jan, Feb, ..., Dez (de_DE)
print(d1.strftime('%b'))
# Month as locale's full name.
# January, February, ..., December (en_US);
# Januar, Februar, ..., Dezember (de_DE)
print(d1.strftime('%B'))
# Month as a zero-padded decimal number.
# 01, 02, ..., 12
print(d1.strftime('%m'))
```

```
30
Apr
April
04
```

```
from datetime import datetime, timedelta
d1 = datetime(1993, 12, 14)
print(d1.strftime('%d %B %Y'))
print("Nur in Zahlen:")
print(d1.strftime('%d/%m/%Y'))
print("Auf US Art:")
print(d1.strftime('%m/%d/%Y'))
print(f"It was a {d1.strftime('%A'):s}!")
```

```
14 December 1993
Nur in Zahlen:
14/12/1993
Auf US Art:
12/14/1993
It was a Tuesday!
```

Python标准库 时间与日期的区别 (time, datetime包)

datetime包是基于time包的一个高级包， 为我们提供了多一层的便利。

datetime可以理解为date和time两个组成部分。date是指年月日构成的日期(相当于日历)，time是指时分秒微秒构成的一天24小时中的具体时间(相当于手表)。你可以将这两个分开管理(datetime.date类，datetime.time类)，也可以将两者合在一起(datetime.datetime类)。由于其构造大同小异，我们将只介绍datetime.datetime类。

```
import time
print(time.time())   # wall clock time, unit: second
print(time.clock())  # processor clock time, unit: second
```

```
1568902580.278
12613.064300088512
```

Python获取时间范围内日期列表和周列表的函数

获取日期列表

```
import datetime


def dateRange(beginDate, endDate):
    dates = []
    dt = datetime.datetime.strptime(beginDate, "%Y-%m-%d")
    date = beginDate[:]
    while date <= endDate:
        dates.append(date)
        dt = dt + datetime.timedelta(1)
        date = dt.strftime("%Y-%m-%d")
    return dates


if __name__ == '__main__':
    for date in dateRange('2019-09-01', '2019-09-11'):
        print(date)
```

```
2019-09-01
2019-09-02
2019-09-03
2019-09-04
2019-09-05
2019-09-06
2019-09-07
2019-09-08
2019-09-09
2019-09-10
2019-09-11
```

python给定起始和结束日期，如何得到中间所有日期

```
import datetime
start = '2019-09-21'
end = '2019-10-01'

datestart = datetime.datetime.strptime(start, '%Y-%m-%d')
dateend = datetime.datetime.strptime(end, '%Y-%m-%d')

while datestart < dateend:
    datestart += datetime.timedelta(days=1)
    print(datestart.strftime('%Y-%m-%d'))
```

```
2019-09-22
2019-09-23
2019-09-24
2019-09-25
2019-09-26
2019-09-27
2019-09-28
2019-09-29
2019-09-30
2019-10-01
```

Python获取两时段内日期、月份、小时的列表清单

```
import warnings
import datetime

warnings.filterwarnings("ignore")


def getNowDay():
    DayNow = datetime.datetime.today().strftime('%Y-%m-%d')
    return DayNow


def getYesterDay():
    YesterDay = (
        datetime.datetime.today() - datetime.timedelta(1)).strftime('%Y-%m-%d')
    return YesterDay


# 获取日期清单
def dateRange(beginDate, endDate):
    dates = []
    dt = datetime.datetime.strptime(beginDate, "%Y-%m-%d")
    date = beginDate[:]
    while date <= endDate:
        dates.append(date)
        dt = dt + datetime.timedelta(1)
        date = dt.strftime("%Y-%m-%d")
    return dates


# 获取月份清单
def monthRange(beginDate, endDate):
    monthSet = set()
    for date in dateRange(beginDate, endDate):
        monthSet.add(date[0:7])
    monthList = []
    for month in monthSet:
        monthList.append(month)
    return sorted(monthList)


# 获取小时清单
def dateHourRange(beginDateHour, endDateHour):
    dhours = []
    dhour = datetime.datetime.strptime(beginDateHour, "%Y-%m-%d %H")
    date = beginDateHour[:]
    while date <= endDateHour:
        dhours.append(date)
        dhour = dhour + datetime.timedelta(hours=1)
        date = dhour.strftime("%Y-%m-%d %H")
    return dhours


print (getNowDay())
print (getYesterDay())
print (dateRange(beginDate='2018-06-05', endDate='2018-07-09'))
print (monthRange(beginDate='2018-01-09', endDate='2019-09-01'))
print (dateHourRange(beginDateHour='2018-01-01 23', endDateHour='2018-01-03 00'))
```

```
2019-09-19
2019-09-18
['2018-06-05', '2018-06-06', '2018-06-07', '2018-06-08', '2018-06-09', '2018-06-10', '2018-06-
11', '2018-06-12', '2018-06-13', '2018-06-14', '2018-06-15', '2018-06-16', '2018-06-17', '2018-
06-18', '2018-06-19', '2018-06-20', '2018-06-21', '2018-06-22', '2018-06-23', '2018-06-24',
'2018-06-25', '2018-06-26', '2018-06-27', '2018-06-28', '2018-06-29', '2018-06-30', '2018-07-
01', '2018-07-02', '2018-07-03', '2018-07-04', '2018-07-05', '2018-07-06', '2018-07-07', '2018-
07-08', '2018-07-09']
['2018-01', '2018-02', '2018-03', '2018-04', '2018-05', '2018-06', '2018-07', '2018-08', '2018-
09', '2018-10', '2018-11', '2018-12', '2019-01', '2019-02', '2019-03', '2019-04', '2019-05',
'2019-06', '2019-07', '2019-08', '2019-09']
['2018-01-01 23', '2018-01-02 00', '2018-01-02 01', '2018-01-02 02', '2018-01-02 03', '2018-01-
02 04', '2018-01-02 05', '2018-01-02 06', '2018-01-02 07', '2018-01-02 08', '2018-01-02 09',
'2018-01-02 10', '2018-01-02 11', '2018-01-02 12', '2018-01-02 13', '2018-01-02 14', '2018-01-02
15', '2018-01-02 16', '2018-01-02 17', '2018-01-02 18', '2018-01-02 19', '2018-01-02 20', '2018-
01-02 21', '2018-01-02 22', '2018-01-02 23', '2018-01-03 00']
```

Python计算出给定的时间段的具体日期列表-大全

计算昨天和明天的日期

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

```
2019-09-20 08:02:41.861000
```

```
yesterday = (datetime.datetime.now() - datetime.timedelta(days=1)).strftime("%Y%m%d")
tomorrow = (datetime.datetime.now() + datetime.timedelta(days=1)).strftime("%Y%m%d")

print(yesterday)
print(tomorrow)
```

```
20190919
20190921
```

计算某一月有多少天

```
import calendar
monthRange = calendar.monthrange(2013, 6)
print(monthRange)
```

```
(5, 30)
```

输出的是一个元组，第一个元素是上一个月的最后一天为星期几(0-6),星期天为0;第二个元素是这个月的天数。

计算周的日期函数。包含某一周开始、结束日期，周的详细日期列表

```
import datetime


def allweeks(week):
```

```
    "计算一年内所有周的具体日期"
    _week = int(week) - 1
    current = datetime.datetime.now()
    start = datetime.date(current.year, 1, 1)
    last_day = datetime.date(current.year, 12, 31)
    isfirst = start.weekday()
    last_week = last_day.strftime('%W')
    weeks = {}  # 计算出一年中每个周的开始和结束日期。例如：0: [datetime.date(2015, 1, 1),
datetime.date(2015, 1, 4)]
    date_list_for_week = []  # 用于得出具体的某一个周的具体天数列表：例如: 第9周：['20150223',
'20150224', '20150225', '20150226', '20150227', '20150228', '20150301']
    if isfirst != 0:
        end = datetime.timedelta(7 - start.weekday() - 1)
        weeks[0] = [start, start + end]
    start += datetime.timedelta(7 - start.weekday())
    for i in range(0, int(last_week)):
        days = datetime.timedelta(weeks=i)
        end = start + days
        if i + 1 == int(last_week):
            weeks[i + 1] = [end, last_day]
        else:
            weeks[i + 1] = [end, end + datetime.timedelta(6)]
    starttime = ''.join(str(
        weeks[_week][0]).split('-'))  # 计算出给出周的开始日期，如20150907
    endtime = ''.join(str(weeks[_week][1]).split("-"))  # 计算出给出周的结束日期，如201509013
    delta = datetime.timedelta(days=1)
    startdate = datetime.datetime(
        int(starttime[0:4]), int(starttime[4:6]), int(starttime[6:8]))
    if int(starttime[4:6]) == int(endtime[4:6]):
        for i in range(int(endtime[6:]) - int(starttime[6:]) + 1):
            days = (startdate + delta * i).strftime('%Y%m%d')
            date_list_for_week.append(days)
    elif int(starttime[4:6]) != int(endtime[4:6]):
        for i in range(7):
            days = (startdate + delta * i).strftime('%Y%m%d')
            date_list_for_week.append(days)
    return date_list_for_week, weeks, starttime, endtime


if __name__ == "__main__":
    week = 3
    print(allweeks(week))
```

```
(['20190114', '20190115', '20190116', '20190117', '20190118', '20190119', '20190120'], {0:
[datetime.date(2019, 1, 1), datetime.date(2019, 1, 6)], 1: [datetime.date(2019, 1, 7),
datetime.date(2019, 1, 13)], 2: [datetime.date(2019, 1, 14), datetime.date(2019, 1, 20)], 3:
[datetime.date(2019, 1, 21), datetime.date(2019, 1, 27)], 4: [datetime.date(2019, 1, 28),
datetime.date(2019, 2, 3)], 5: [datetime.date(2019, 2, 4), datetime.date(2019, 2, 10)], 6:
[datetime.date(2019, 2, 11), datetime.date(2019, 2, 17)], 7: [datetime.date(2019, 2, 18),
datetime.date(2019, 2, 24)], 8: [datetime.date(2019, 2, 25), datetime.date(2019, 3, 3)], 9:
[datetime.date(2019, 3, 4), datetime.date(2019, 3, 10)], 10: [datetime.date(2019, 3, 11),
datetime.date(2019, 3, 17)], 11: [datetime.date(2019, 3, 18), datetime.date(2019, 3, 24)], 12:
[datetime.date(2019, 3, 25), datetime.date(2019, 3, 31)], 13: [datetime.date(2019, 4, 1),
datetime.date(2019, 4, 7)], 14: [datetime.date(2019, 4, 8), datetime.date(2019, 4, 14)], 15:
[datetime.date(2019, 4, 15), datetime.date(2019, 4, 21)], 16: [datetime.date(2019, 4, 22),
datetime.date(2019, 4, 28)], 17: [datetime.date(2019, 4, 29), datetime.date(2019, 5, 5)], 18:
[datetime.date(2019, 5, 6), datetime.date(2019, 5, 12)], 19: [datetime.date(2019, 5, 13),
datetime.date(2019, 5, 19)], 20: [datetime.date(2019, 5, 20), datetime.date(2019, 5, 26)], 21:
[datetime.date(2019, 5, 27), datetime.date(2019, 6, 2)], 22: [datetime.date(2019, 6, 3),
datetime.date(2019, 6, 9)], 23: [datetime.date(2019, 6, 10), datetime.date(2019, 6, 16)], 24:
[datetime.date(2019, 6, 17), datetime.date(2019, 6, 23)], 25: [datetime.date(2019, 6, 24),
datetime.date(2019, 6, 30)], 26: [datetime.date(2019, 7, 1), datetime.date(2019, 7, 7)], 27:
[datetime.date(2019, 7, 8), datetime.date(2019, 7, 14)], 28: [datetime.date(2019, 7, 15),
datetime.date(2019, 7, 21)], 29: [datetime.date(2019, 7, 22), datetime.date(2019, 7, 28)], 30:
[datetime.date(2019, 7, 29), datetime.date(2019, 8, 4)], 31: [datetime.date(2019, 8, 5),
datetime.date(2019, 8, 11)], 32: [datetime.date(2019, 8, 12), datetime.date(2019, 8, 18)], 33:
[datetime.date(2019, 8, 19), datetime.date(2019, 8, 25)], 34: [datetime.date(2019, 8, 26),
datetime.date(2019, 9, 1)], 35: [datetime.date(2019, 9, 2), datetime.date(2019, 9, 8)], 36:
[datetime.date(2019, 9, 9), datetime.date(2019, 9, 15)], 37: [datetime.date(2019, 9, 16),
datetime.date(2019, 9, 22)], 38: [datetime.date(2019, 9, 23), datetime.date(2019, 9, 29)], 39:
[datetime.date(2019, 9, 30), datetime.date(2019, 10, 6)], 40: [datetime.date(2019, 10, 7),
datetime.date(2019, 10, 13)], 41: [datetime.date(2019, 10, 14), datetime.date(2019, 10, 20)],
42: [datetime.date(2019, 10, 21), datetime.date(2019, 10, 27)], 43: [datetime.date(2019, 10,
28), datetime.date(2019, 11, 3)], 44: [datetime.date(2019, 11, 4), datetime.date(2019, 11, 10)],
45: [datetime.date(2019, 11, 11), datetime.date(2019, 11, 17)], 46: [datetime.date(2019, 11,
18), datetime.date(2019, 11, 24)], 47: [datetime.date(2019, 11, 25), datetime.date(2019, 12,
1)], 48: [datetime.date(2019, 12, 2), datetime.date(2019, 12, 8)], 49: [datetime.date(2019, 12,
9), datetime.date(2019, 12, 15)], 50: [datetime.date(2019, 12, 16), datetime.date(2019, 12,
22)], 51: [datetime.date(2019, 12, 23), datetime.date(2019, 12, 29)], 52: [datetime.date(2019,
12, 30), datetime.date(2019, 12, 31)]}, '20190114', '20190120')
```

计算自定义时间的日期函数。（比如计算20150811-20150922之间的日期列表）

```
import datetime

def get_datelist(starttime,endtime):
    startdate = datetime.datetime(int(starttime[0:4]),int(starttime[4:6]),int(starttime[6:8]))
    #now = datetime.datetime.now()
    delta = datetime.timedelta(days=1)
    # my_yestoday = startdate + delta
    # my_yes_time = my_yestoday.strftime('%Y%m%d')
    n = 0
    date_list = []
    while 1:
        if starttime<=endtime:
            days = (startdate  + delta*n).strftime('%Y%m%d')
            n = n+1
            date_list.append(days)
            if days == endtime:
                break
    return date_list
print(get_datelist('20150811','20150922'))
```

```
['20150811', '20150812', '20150813', '20150814', '20150815', '20150816', '20150817', '20150818',
'20150819', '20150820', '20150821', '20150822', '20150823', '20150824', '20150825', '20150826',
'20150827', '20150828', '20150829', '20150830', '20150831', '20150901', '20150902', '20150903',
'20150904', '20150905', '20150906', '20150907', '20150908', '20150909', '20150910', '20150911',
'20150912', '20150913', '20150914', '20150915', '20150916', '20150917', '20150918', '20150919',
'20150920', '20150921', '20150922']
```

将自定义时间分割成星期来进行取值，不足一星期的按天来取值

计算某一时间在第几周(第几周包含的时间区间)

```
import collections

def startdate_enddate_weeks_list(startdate,enddate):
    '''计算startdate---enddate的周所属的周和周包含的日期列表
     week_date_start_end {1: ['20181231','20190106'],...}
     '''

    start_date=datetime.datetime.strptime(str(startdate),'%Y%m%d')
    end_date=datetime.datetime.strptime(str(enddate),'%Y%m%d')
    _u=datetime.timedelta(days=1)
    n=0
    week_date=collections.OrderedDict()
    date=[]
    while 1:
        _time=start_date+n*_u
        y,w=_time.isocalendar()[:2]
        year_week=str(y)+str(w)
        if year_week in week_date:
            date.append(_time.strftime('%Y%m%d'))
        else:
            date=[_time.strftime('%Y%m%d')]
        week_date[year_week]=date
        n=n+1
        if _time==end_date:
            break
    return week_date


a=startdate_enddate_weeks_list("20151011","20151102")
print(a)
```

```
OrderedDict([('201541', ['20151011']), ('201542', ['20151012', '20151013', '20151014',
'20151015', '20151016', '20151017', '20151018']), ('201543', ['20151019', '20151020', '20151021',
'20151022', '20151023', '20151024', '20151025']), ('201544', ['20151026', '20151027', '20151028',
'20151029', '20151030', '20151031', '20151101']), ('201545', ['20151102'])])
```

算出某一日期所属当前月的所有日期列表，dates是一个具体日期

例20160307

```
def get_one_mon_all_daylist(dates):
    '''得到一个月的具体日期列表：从1号到月末的天列表'''
    a_mon_daylist = []
    mon_days = calendar.monthrange(int(dates[:4]),int(dates[4:6]))[1]#得到当前月的天数
    for i in range(1,int(mon_days)+1):
        if len(str(i)) == 1:
            a_mon_daylist.append(dates[:6]+"0"+str(i))
        elif len(str(i)) == 2:
            a_mon_daylist.append(dates[:6]+str(i))
    return a_mon_daylist
```

```
print(get_one_mon_all_daylist('20160307'))
```

```
['20160301', '20160302', '20160303', '20160304', '20160305', '20160306', '20160307', '20160308',
'20160309', '20160310', '20160311', '20160312', '20160313', '20160314', '20160315', '20160316',
'20160317', '20160318', '20160319', '20160320', '20160321', '20160322', '20160323', '20160324',
'20160325', '20160326', '20160327', '20160328', '20160329', '20160330', '20160331']
```

python获取任意时间段的日期(年月日)

```
import datetime

#   结束时间
d1 = datetime.date(2019, 10, 7)
#   开始时间
d2 = datetime.date(2019, 9, 25)
#   间隔天数
date_range = int((d1 - d2).days)

# today = datetime.date.today()
#   只要计算出两个时间段区间的

dateList = []
while date_range != 0:
    date = d1 - datetime.timedelta(days=date_range)
    date_range = date_range - 1
    dateList.append(str(date))

print(dateList)
```

```
['2019-09-25', '2019-09-26', '2019-09-27', '2019-09-28', '2019-09-29', '2019-09-30', '2019-10-
01', '2019-10-02', '2019-10-03', '2019-10-04', '2019-10-05', '2019-10-06']
```

python判断当前时间是否在某个时间段内

python判断当前时间是否在某个时间段里

```
# https://gist.github.com/RicoNut/741bdc134436e9f5fc93539624fbbfc8
# 例如是否在9:30-11:30和13:30-15:30之间
# in_time_range("093000-113000,133000-153000")
def in_time_range(ranges):
    now = time.strptime(time.strftime("%H%M%S"),"%H%M%S")
    ranges = ranges.split(",")
    for range in ranges:
        r = range.split("-")
        if time.strptime(r[0],"%H%M%S") <= now <= time.strptime(r[1],"%H%M%S") or
time.strptime(r[0],"%H%M%S") >= now >=time.strptime(r[1],"%H%M%S"):
            return True
    return False
```

```
print(in_time_range("093000-113000,133000-153000"))
```

```
False
```

判断时间是否处于另外start和end之间

```
import datetime

dt = datetime.date(2015, 6, 1)
start = datetime.date(2015, 1, 1)
end = datetime.date(2015, 12, 1)

print(dt > start)
print(dt < end)
```

```
True
True
```

python判断时间是否落在两个时区之间（只比较时刻不比较日期）

```
import datetime
# 范围时间
d_time1 = datetime.datetime.strptime(str(datetime.datetime.now().date())+'8:30', '%Y-%m-%d%H:%M')
d_time2 =  datetime.datetime.strptime(str(datetime.datetime.now().date())+'18:33', '%Y-%m-
%d%H:%M')

# 当前时间
n_time = datetime.datetime.now()
print('当前时间：  '+str(n_time))
# 判断当前时间是否在范围时间内
if n_time > d_time1 and n_time<d_time2:
    print("在此区间中")
else:
    print("不在此区间")
```

```
当前时间： 2019-09-19 22:48:56.445000
不在此区间
```

时间字符串直接比大小

```
import datetime
t1 = '15:40'
t2 = '18:17'
now = datetime.datetime.now().strftime("%H:%M")
print("当前时间:" + now)
if t1 < now < t2:
    print("在此区间中")
else:
    print('不在此区间中')
```

```
当前时间:22:50
不在此区间中
```

直接将当前时间格式化成字符串然后转换成整数进行比较

```
import time
now = time.strftime("%H%M%S")
print("当前时间：" + now)
#时间区间[09:35:10,18:01:01]
if(180101 > int(time.strftime("%H%M%S")) > 93510):
    print('在此区间中')
```

```
当前时间：225041
```

如何使用Python的datetime模块确定当前时间是否在指定范围内?

```
from datetime import datetime, time
now = datetime.now()
now_time = now.time()
if now_time >= time(8,30) and now_time <= time(16,30):
    print("yes, within the interval")
else:
    print("no, not in the interval")
```

```
no, not in the interval
```

```

```

python 如何判断两个时间是否在同一个5分钟时段内？ - 知乎

5分钟时段是按9:55，10:00，10:05，10:10划分的，如何判断两个时间是不是在同一时间段。

比如9:56:00和9:59:59是属于一个时段的，但9:59:59和10:00:00就不是一个时段的了

用 unix 时间戳除以300的得数是否一致来判断，相同的即在同一个时间段，否则不在。

300 是 5 分钟的秒数，然后中国的时区 +8 小时刚好是 5 分钟的倍数，所以不影响。

```
time_list = ['09:56:00', '09:59:59', '10:00:00', '10:01:01', '10:06:00']

for t in time_list:
    timestamp = time.mktime(time.strptime('2016-01-01 %s' % t, "%Y-%m-%d %H:%M:%S"))
    print('%s\t%s' % (t, int(timestamp)/300))
```

```
09:56:00        4838711.2
09:59:59        4838711.996666667
10:00:00        4838712.0
10:01:01        4838712.203333333
10:06:00        4838713.2
```

python 时间戳、时间字符格式化、判断时间在某个时间段内

获得当天时间的前一天、后一天（注意时间戳是以秒s为单位的，当将时间戳再转为格式化的时间字符串时，注意不能再用%f 毫秒）

```
from datetime import timedelta, date
import time

def testfunc():
    today = date.today()
    print(today, type(today))
    yesterday = date.today() + timedelta(days=-1)
    print(yesterday, type(yesterday))
    yesterday = (date.today() + timedelta(days=-1)).strftime("%Y.%m.%d")
    print(yesterday)

    a = time.time()
    print(a)

    ltime = time.localtime(a)
    print("ltime", ltime)

    gtime = time.gmtime(a)
    print(gtime)

    time_string = time.strftime('%Y.%m.%d.%H.%M.%S', ltime)
    gtime_string = time.strftime('%Y-%m-%dT%H:%M:%SZ', time.gmtime(time.time()))
    print(time_string)
    print("gtime_string", gtime_string, type(gtime_string))

    date_str = "2018/11/13 17:32"
    print("start:", date_str)

    timeArray = time.strptime(date_str, "%Y/%m/%d %H:%M")
    timeStamp = int(time.mktime(timeArray))
    timeStampPlusOneday = timeStamp + +86400
    # print timeStampPlusOneday

    timeArray = time.localtime(timeStampPlusOneday)
    otherStyleTime = time.strftime("%Y/%m/%d %H:%M", timeArray)
    print("other:", otherStyleTime)
```

```
    print("test")
    yesterday = date.today() + timedelta(days=-1)
    yesterday = yesterday.strftime("%Y.%m.%d")
    print(yesterday)
    print(yesterday + " 00:00:00")
    print("uesd:", time.strftime('%Y-%m-%d %H:%M:%S', time.gmtime(time.time())))
```

```
testfunc()
```

```
2019-09-20 <class 'datetime.date'>
2019-09-19 <class 'datetime.date'>
2019.09.19
1568941240.077
ltime time.struct_time(tm_year=2019, tm_mon=9, tm_mday=20, tm_hour=9, tm_min=0, tm_sec=40,
tm_wday=4, tm_yday=263, tm_isdst=0)
time.struct_time(tm_year=2019, tm_mon=9, tm_mday=20, tm_hour=1, tm_min=0, tm_sec=40, tm_wday=4,
tm_yday=263, tm_isdst=0)
2019.09.20.09.00.40
gtime_string 2019-09-20T01:00:40Z <class 'str'>
start: 2018/11/13 17:32
other: 2018/11/14 17:32
test
2019.09.19
2019.09.19 00:00:00
uesd: 2019-09-20 01:00:40
```

判断某个时间点是否在两个时间点内

```
def getNonWorkTime():
    # 范围时间
    a_time = datetime.datetime.strptime(
        str(datetime.datetime.now().date()) + '2:00', '%Y-%m-%d%H:%M')
    b_time = datetime.datetime.strptime(
        str(datetime.datetime.now().date()) + '5:00', '%Y-%m-%d%H:%M')
    c_time = datetime.datetime.strptime(
        str(datetime.datetime.now().date()) + '7:00', '%Y-%m-%d%H:%M')
    d_time = datetime.datetime.strptime(
        str(datetime.datetime.now().date()) + '13:00', '%Y-%m-%d%H:%M')
    print(a_time, b_time, c_time, d_time)
    # 当前时间
    n_time = datetime.datetime.now()
    print("now:", n_time)
    # 判断当前时间是否在范围时间内
    if b_time > n_time > a_time or d_time > n_time > c_time:
        print("not work")
    else:
        print("work")
```

```
getNonWorkTime()
```

```
2019-09-20 02:00:00 2019-09-20 05:00:00 2019-09-20 07:00:00 2019-09-20 13:00:00
now: 2019-09-20 09:02:42.126000
not work
```

获得一分钟之前的时间，一小时之后的时间

```
def oneMinStr():
    start = time.strftime('%Y/%m/%d %H:%M:%S', time.gmtime(time.time()))
    print("start:", start,type(start))
    timeArray = time.strptime(start, "%Y/%m/%d %H:%M:%S")
    timeStamp = int(time.mktime(timeArray))
    timeStampPlusOneday = timeStamp + -60
    # print timeStampPlusOneday
    timeArray = time.localtime(timeStampPlusOneday)
    endtime = time.strftime("%Y/%m/%d %H:%M:%S", timeArray)
    print("other:", endtime,endtime.split()[1])
    s = "(" + endtime + "-" + start.split()[1] + ")"
    print(s)
```

```
oneMinStr()
```

```
start: 2019/09/20 01:04:04 <class 'str'>
other: 2019/09/20 01:03:04 01:03:04
(2019/09/20 01:03:04-01:04:04)
```

```
def addOneHour():
    #date_str = "2018/11/13 17:32"
    date_str = "2019-02-21T16:00:00.000000Z"
    print("start:", date_str)
    timeArray = time.strptime(date_str, "%Y-%m-%dT%H:%M:%S.%fZ")
    timeStamp = int(time.mktime(timeArray))
    timeStampPlusOneday = timeStamp + +3600
    # print timeStampPlusOneday
    #timeStamp = 1381419600
    dateArray = datetime.datetime.utcfromtimestamp(timeStampPlusOneday)
    otherStyleTime = dateArray.strftime("%Y--%m--%d %H:%M:%S")
    print(otherStyleTime)  # 2013--10--10 15:40:00


    timeArray = time.localtime(timeStampPlusOneday)
    otherStyleTime = time.strftime("%Y/%m/%d %H:%M", timeArray)
    print("other:", otherStyleTime)

    minTime = "2019-02-21T16:00:00.000000Z"
    maxTime = "2019-02-28T16:00:00.000000Z"
    gt = "2019-02-21T16:00:00.000000Z"
    lt = "2019-02-21T18:00:00.000000Z"

    print("initial time:", gt)

    gtArray = time.strptime(gt, "%Y-%m-%dT%H:%M:%S.%fZ")
    #时间戳与字符类型互转
    timeStamp = int(time.mktime(gtArray))
    timeStampPlusOneHour = timeStamp + +3600
    print(timeStampPlusOneHour)
    timeArray = time.localtime(timeStampPlusOneHour)
    otherStyleTime = time.strftime("%Y-%m-%dT%H:%M:%SZ", timeArray)
    print("addOH", otherStyleTime)
```

```
addOneHour()
```

```
start: 2019-02-21T16:00:00.000000Z
2019--02--21 09:00:00
other: 2019/02/21 17:00
initial time: 2019-02-21T16:00:00.000000Z
1550739600
addOH 2019-02-21T17:00:00Z
```

时间分段--划分为等间隔的时间间隔

How to divide python datetime-span into equally spaced time-intervals?

```
from datetime import timedelta
import datetime

date_a = datetime.datetime.now()
date_b = date_a + timedelta(minutes=27)
# date_a = datetime.datetime(2016, 8, 9, 8, 24, 30, 993352)
# date_b = datetime.datetime(2016, 8, 9, 7, 24, 30, 993352)

delta = timedelta(minutes=5)
five_min_timestamps = []
date_x = date_a
while date_x < date_b:
    date_x += timedelta(minutes=5)
    five_min_timestamps.append(date_x)

print(five_min_timestamps)
```

```
[datetime.datetime(2019, 9, 20, 9, 21, 24, 582000), datetime.datetime(2019, 9, 20, 9, 26, 24,
582000), datetime.datetime(2019, 9, 20, 9, 31, 24, 582000), datetime.datetime(2019, 9, 20, 9, 36,
24, 582000), datetime.datetime(2019, 9, 20, 9, 41, 24, 582000), datetime.datetime(2019, 9, 20, 9,
46, 24, 582000)]
```

Given a date range how can we break it up into N contiguous sub-intervals?

```
import datetime
begin = datetime.date(2019, 10, 1)
end = datetime.date(2019, 10, 21)

intervals = 6

date_list = []

delta = (end - begin)/4
for i in range(1, intervals + 1):
    date_list.append((begin+i*delta).strftime('%Y%m%d'))

print(date_list)
```

```
['20191006', '20191011', '20191016', '20191021', '20191026', '20191031']
```

```
def date_range(start, end, intv):
    from datetime import datetime
    start = datetime.strptime(start,"%Y%m%d")
    end = datetime.strptime(end,"%Y%m%d")
    diff = (end  - start ) / intv
    for i in range(intv):
        yield (start + diff * i).strftime("%Y%m%d")
    yield end.strftime("%Y%m%d")

begin = '20191121'
end = '20191208'
print(list(date_range(begin, end, 3)))
```

```
['20191121', '20191126', '20191202', '20191208']
```

DateTimeRange 时间库

DateTimeRange 0.6.1

```
pip install DateTimeRange
```

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-22T10:10:00+0900")
print(str(time_range))
```

```
2015-03-22T10:00:00+0900 - 2015-03-22T10:10:00+0900
```

按天间隔

```
import datetime
from datetimerange import DateTimeRange

time_range = DateTimeRange("2015-01-01T00:00:00+0900", "2015-01-04T00:00:00+0900")
for value in time_range.range(datetime.timedelta(days=1)):
    print(value)
```

```
2015-01-01 00:00:00+09:00
2015-01-02 00:00:00+09:00
2015-01-03 00:00:00+09:00
2015-01-04 00:00:00+09:00
```

按月间隔

```
from datetimerange import DateTimeRange
from dateutil.relativedelta import relativedelta

time_range = DateTimeRange("2015-01-01T00:00:00+0900", "2016-01-01T00:00:00+0900")
for value in time_range.range(relativedelta(months=+4)):
    print(value)
```

```
2015-01-01 00:00:00+09:00
2015-05-01 00:00:00+09:00
2015-09-01 00:00:00+09:00
2016-01-01 00:00:00+09:00
```

测试当前时间是否在区间内

Test whether a value within the time range

```python
from datetimerange import DateTimeRange

time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-22T10:10:00+0900")
print("2015-03-22T10:05:00+0900" in time_range)
print("2015-03-22T10:15:00+0900" in time_range)

time_range_smaller = DateTimeRange("2015-03-22T10:03:00+0900", "2015-03-22T10:07:00+0900")
print(time_range_smaller in time_range)
```

```
True
False
True
```

判断两个时间段是否重叠

Test whether a value intersect the time range

```python
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900",
                           "2015-03-22T10:10:00+0900")
x = DateTimeRange("2015-03-22T10:05:00+0900", "2015-03-22T10:15:00+0900")
print(time_range.is_intersection(x))
```

```
True
```

确定交叉的时间范围

Make an intersected time range

```python
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-22T10:10:00+0900")
x = DateTimeRange("2015-03-22T10:05:00+0900", "2015-03-22T10:15:00+0900")
print(time_range.intersection(x))
```

```
2015-03-22T10:05:00+0900 - 2015-03-22T10:10:00+0900
```

确定两个时间区间的最大时间范围

Make an encompassed time range

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-22T10:10:00+0900")
x = DateTimeRange("2015-03-22T10:05:00+0900", "2015-03-22T10:15:00+0900")
print(time_range.encompass(x))
```

```
2015-03-22T10:00:00+0900 - 2015-03-22T10:15:00+0900
```

截断时间范围

Truncate time range

```
from datetimerange import DateTimeRange

time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-22T10:10:00+0900")
time_range.is_output_elapse = True
print("before truncate: ", time_range)

time_range.truncate(10)
print("after truncate:  ", time_range)
```

```
before truncate:  2015-03-22T10:00:00+0900 - 2015-03-22T10:10:00+0900 (0:10:00)
after truncate:   2015-03-22T10:00:30+0900 - 2015-03-22T10:09:30+0900 (0:09:00)
```

Python 随机取一个时间段里面的时间

https://segmentfault.com/q/1010000006617581

如规定时间10:00:00~18:00:00 我想随机取里面的时间里面

把这个时间段转换成总共的秒数。

定一个用秒数表示的start_time和end_time；

random（start_time，end_time）；

然后转换成时间

```
base_time + timedelta(seconds=randint(start, end))
```

```
import datetime
import random

def random_datetime(start_datetime, end_datetime):
    delta = end_datetime - start_datetime
    inc = random.randrange(delta.total_seconds())
    return start_datetime + datetime.timedelta(seconds=inc)

if __name__=='__main__':
    start_datetime = datetime.datetime(2019, 8, 17, 10, 0, 0)
    end_datetime = datetime.datetime(2019, 8, 17, 18, 0, 0)
    dt = random_datetime(start_datetime, end_datetime)
    print(dt)
```

```
2019-08-17 10:13:14
```

python 判断当前时间是否为零点

```
import time
time_now = int(time.time()) #unix时间
time_local = time.localtime(time_now) #转换为win_time
dt = time.strftime("%H:%M:%S", time_local)# 转换成新的时间格式(18:59:20)
print(dt)

a=dt.split(':')
b=[]
for a in dt.split(':'):
    b.append(a)
c=''.join(b)
if c=='000000':
    print('是零点')
else:
    print('不是零点')
```

```
09:54:17
不是零点
```

知道2个时间点 2016-02-23 和 2016-05-23 ， 如何获取这2个时间之间对应的周数的列表

```
from datetime import datetime,timedelta

d1 = "2019-08-24"
d2 = "2019-09-28"

date1=datetime.strptime(d1,"%Y-%m-%d")
date2=datetime.strptime(d2,"%Y-%m-%d")
delta = date2 - date1

per_day_seconds = 24*60*60
per_week_seconds = 7*per_day_seconds
total_seconds = delta.total_seconds()
weeks = total_seconds//per_week_seconds
days = total_seconds//per_day_seconds
if weeks*per_week_seconds < total_seconds:    weeks += 1

print("Delata Time:",delta,"Weeks:",weeks)

oneday = timedelta(days=1)
WEEK=["M","T","W","R","F","S","U"]

d0=date1
weekdays = [d0,]
for i in range(int(days)):
    d0 += oneday
    if d0.weekday() == 6 :     # For Sunday
        print(",".join([str((WEEK[d.weekday()],d.strftime("%Y-%m-%d"))) for d in weekdays]))
        weekdays = []
    weekdays.append(d0)

print(",".join([str((WEEK[d.weekday()],d.strftime("%Y-%m-%d"))) for d in weekdays]))
```

```
Delata Time: 35 days, 0:00:00 Weeks: 5.0
('S', '2019-08-24')
('U', '2019-08-25'),('M', '2019-08-26'),('T', '2019-08-27'),('W', '2019-08-28'),('R', '2019-08-
29'),('F', '2019-08-30'),('S', '2019-08-31')
('U', '2019-09-01'),('M', '2019-09-02'),('T', '2019-09-03'),('W', '2019-09-04'),('R', '2019-09-
05'),('F', '2019-09-06'),('S', '2019-09-07')
('U', '2019-09-08'),('M', '2019-09-09'),('T', '2019-09-10'),('W', '2019-09-11'),('R', '2019-09-
12'),('F', '2019-09-13'),('S', '2019-09-14')
('U', '2019-09-15'),('M', '2019-09-16'),('T', '2019-09-17'),('W', '2019-09-18'),('R', '2019-09-
19'),('F', '2019-09-20'),('S', '2019-09-21')
('U', '2019-09-22'),('M', '2019-09-23'),('T', '2019-09-24'),('W', '2019-09-25'),('R', '2019-09-
26'),('F', '2019-09-27'),('S', '2019-09-28')
```

以分钟划分一天24获得时间列表

直接列写字符串的方式

```
hourList = ['00', '01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13',
'14', '15',  '16', '17', '18', '19', '20', '21', '22', '23']
print(hourList)

# for hour in hourList:
#      print(hour)
```

```
['00', '01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15',
'16', '17', '18', '19', '20', '21', '22', '23']
```

通过字符格式化及range的方式5

```
hour_list = ['{num:02d}'.format(num=i) for i in range(24)]
print(hour_list)

# for hour in hour_list:
#      print(hour)
```

```
['00', '01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15',
'16', '17', '18', '19', '20', '21', '22', '23']
```

获取分钟间隔列表

```
# hour_list = ['{h:02d}:{m:02d}'.format(h=h, m=m) for h in range(24) for m in range(60)]
hour_list = ['{h:02d}:{m:02d}'.format(h=h, m=m) for h in range(2) for m in range(60)]
print(hour_list)

# for hour in hour_list:
#      print(hour)
```

```
['00:00', '00:01', '00:02', '00:03', '00:04', '00:05', '00:06', '00:07', '00:08', '00:09',
'00:10', '00:11', '00:12', '00:13', '00:14', '00:15', '00:16', '00:17', '00:18', '00:19',
'00:20', '00:21', '00:22', '00:23', '00:24', '00:25', '00:26', '00:27', '00:28', '00:29',
'00:30', '00:31', '00:32', '00:33', '00:34', '00:35', '00:36', '00:37', '00:38', '00:39',
'00:40', '00:41', '00:42', '00:43', '00:44', '00:45', '00:46', '00:47', '00:48', '00:49',
'00:50', '00:51', '00:52', '00:53', '00:54', '00:55', '00:56', '00:57', '00:58', '00:59',
'01:00', '01:01', '01:02', '01:03', '01:04', '01:05', '01:06', '01:07', '01:08', '01:09',
'01:10', '01:11', '01:12', '01:13', '01:14', '01:15', '01:16', '01:17', '01:18', '01:19',
'01:20', '01:21', '01:22', '01:23', '01:24', '01:25', '01:26', '01:27', '01:28', '01:29',
'01:30', '01:31', '01:32', '01:33', '01:34', '01:35', '01:36', '01:37', '01:38', '01:39',
'01:40', '01:41', '01:42', '01:43', '01:44', '01:45', '01:46', '01:47', '01:48', '01:49',
'01:50', '01:51', '01:52', '01:53', '01:54', '01:55', '01:56', '01:57', '01:58', '01:59']
```

```
time_str = '13:55'
time_object = datetime.strptime(time_str, '%H:%M').time()
print(time_object)
print(type(time_object))
```

```
13:55:00
<class 'datetime.time'>
```

字符串转为时间格式

Converting Strings using datetime

```
import datetime

date_time_str = '2018-06-29 08:15:27.243860'
date_time_obj = datetime.datetime.strptime(date_time_str, '%Y-%m-%d %H:%M:%S.%f')

print('Date:', date_time_obj.date())
print('Time:', date_time_obj.time())
print('Date-time:', date_time_obj)
```

```
Date: 2018-06-29
Time: 08:15:27.243860
Date-time: 2018-06-29 08:15:27.243860
```

```
import datetime

date_time_str = 'Jun 28 2018  7:40AM'
date_time_obj = datetime.datetime.strptime(date_time_str, '%b %d %Y %I:%M%p')

print('Date:', date_time_obj.date())
print('Time:', date_time_obj.time())
print('Date-time:', date_time_obj)
```

```
Date: 2018-06-28
Time: 07:40:00
Date-time: 2018-06-28 07:40:00
```

Python string to datetime – strptime()

```
datetime.strptime(date_string, format)

time.strptime(time_string[, format])
```

字符串转时间

```
from datetime import datetime

datetime_str = '09/19/18 13:55:26'

datetime_object = datetime.strptime(datetime_str, '%m/%d/%y %H:%M:%S')

print(type(datetime_object))
print(datetime_object)  # printed in default format
```

```
<class 'datetime.datetime'>
2018-09-19 13:55:26
```

字符串转日期对象

```
date_str = '09-19-2018'
date_object = datetime.strptime(date_str, '%m-%d-%Y').date()
print(type(date_object))
print(date_object)  # printed in default formatting
```

```
<class 'datetime.date'>
2018-09-19
```

```
date_str = '09/19/2018'
date_object = datetime.strptime(date_str, '%m/%d/%Y').date()
print(type(date_object))
print(date_object)  # printed in default formatting
```

```
<class 'datetime.date'>
2018-09-19
```

字符串转时间对象

```
time_str = '13::55::26'
time_object = datetime.strptime(time_str, '%H::%M::%S').time()
print(type(time_object))
print(time_object)
```

```
<class 'datetime.time'>
13:55:26
```

```
time_str = '13:55'
time_object = datetime.strptime(time_str, '%H:%M').time()
print(time_object)
```

```
13:55:00
```

```
import time

time_obj = time.strptime(time_str, '%H::%M::%S')
print(type(time_obj))
print(time_obj)

# default formatting - "%a %b %d %H:%M:%S %Y"
print(time.strptime('Wed Sep 19 14:55:02 2018'))
```

```
<class 'time.struct_time'>
time.struct_time(tm_year=1900, tm_mon=1, tm_mday=1, tm_hour=13, tm_min=55, tm_sec=26, tm_wday=0,
tm_yday=1, tm_isdst=-1)
time.struct_time(tm_year=2018, tm_mon=9, tm_mday=19, tm_hour=14, tm_min=55, tm_sec=2, tm_wday=2,
tm_yday=262, tm_isdst=-1)
```

错误用法示例

```
datetime_str = '09/19/18 13:55:26'

try:
    datetime_object = datetime.strptime(datetime_str, '%m/%d/%y')
except ValueError as ve:
    print('ValueError Raised:', ve)

time_str = '99::55::26'

try:
    time_object = time.strptime(time_str, '%H::%M::%S')
except ValueError as e:
    print('ValueError:', e)
```

```
ValueError Raised: unconverted data remains:  13:55:26
ValueError: time data '99::55::26' does not match format '%H::%M::%S'
```