time

在Python中，与时间处理有关的模块就包括：time，datetime 以及 calendar

在开始之前，首先要说明这几点：

在Python中，通常有这几种方式来表示时间：

- 时间戳 timestamp；
- 格式化的时间字符串（str）；
- 元组（struct_time，共九个元素）。

由于 Python 的 time 模块实现主要调用C库，所以各个平台可能有所不同。

1. UTC（Coordinated Universal Time，世界协调时）亦即格林威治天文时间，世界标准时间。在中国为UTC+8。DST（Daylight Saving Time）即夏令时。
2. 时间戳（timestamp）的方式：通常来说，时间戳表示的是从1970年1月1日00:00:00开始按秒计算的偏移量。我们运行"type(time.time())"，返回的是float类型。返回时间戳方式的函数主要有time()，clock()等。
3. 元组（struct_time）方式：struct_time元组共有9个元素，返回struct_time的函数主要有全球统一时间gmtime()，localtime()，strptime()。

在 Python 中三种时间表达方式

1. 时间戳（`timestamp`）：也就是 1970 年 1 月 1 日之后的秒，例如 1506388236.216345，可以通过 `time.time()` 获得。时间戳是一个浮点数，可以进行加减运算，但请注意不要让结果超出取值范围。
2. 格式化的时间字符串（`string_time`）：也就是年月日时分秒这样的我们常见的时间字符串，例如 `2017-09-26 09:12:48`，可以通过 `time.strftime('%Y-%m-%d')` 获得;
3. 结构化时间（`struct_time`）：一个包含了年月日时分秒的多元元组，例如 `time.struct_time(tm_year=2017, tm_mon=9, tm_mday=26, tm_hour=9, tm_min=14, tm_sec=50, tm_wday=1, tm_yday=269, tm_isdst=0)`，可以通过 `time.localtime()` 获得。

时间戳

时间戳（timestamp），一个能表示一份数据在某个特定时间之前已经存在的、 完整的、 可验证的数据,通常是一个字符序列，唯一地标识某一刻的时间。

.time()

返回当前系统时间戳。时间戳是一个数字可以做算术运算。

```
>>> time.time()
1506391907.020303
```

该方法经常用于计算程序运行时间：

```python
import time

# 打印时间戳
print(time.time())
time.sleep(3)
print(time.time())
```

.sleep(t)

time 模块最常用的方法之一，用来睡眠或者暂停程序 t 秒，t 可以是浮点数或整数。

struct_time

> 结构化时间

.localtime([secs])

使用 `time.localtime()` 等方法可以获得一个结构化时间元组。secs参数未提供时，则以当前时间为准。

```
>>> time.localtime()
time.struct_time(tm_year=2011, tm_mon=5, tm_mday=5, tm_hour=14, tm_min=14, tm_sec=50, tm_wday=3, tm_yday=125, tm_isdst=0)
>>> time.localtime(1304575584.1361799)
time.struct_time(tm_year=2011, tm_mon=5, tm_mday=5, tm_hour=14, tm_min=6, tm_sec=24, tm_wday=3, tm_yday=125, tm_isdst=0)

```

结构化时间元组共有 9 个元素，按顺序排列如下表：

索引（Index）	属性（Attribute）	值（Values）
0	tm_year（年）	比如2011
1	tm_mon（月）	1 - 12
2	tm_mday（日）	1 - 31
3	tm_hour（时）	0 - 23
4	tm_min（分）	0 - 59
5	tm_sec（秒）	0 - 61（包含闰秒和双闰秒）
6	tm_wday（weekday）	0 - 6（0表示周一）
7	tm_yday（一年中的第几天）	1 - 366
8	tm_isdst（是否是夏令时）	默认为-1,0,1

既然结构化时间是一个元组，那么就可以通过索引进行取值，也可以进行分片，或者通过属性名获取对应的值。

```
>>>import time
>>> lt = time.localtime()
>>> lt
time.struct_time(tm_year=2017, tm_mon=9, tm_mday=26, tm_hour=9, tm_min=27,
tm_sec=29, tm_wday=1, tm_yday=269, tm_isdst=0)
>>> lt[3]
9
>>> lt[2:5]
(26, 9, 27)
>>> lt.tm_wday
1

```

但是要记住，**Python 的 time 类型是不可变类型，所有的时间值都只读，不能改**！！

```
>>> lt.tm_wday = 2
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    lt.tm_wday = 2
AttributeError: readonly attribute

```

.gmtime([secs])

将一个时间戳转换为 UTC 时区的结构化时间。可选参数 secs 的默认值为`time.time()`。

```
>>> time.gmtime()
time.struct_time(tm_year=2017, tm_mon=9, tm_mday=26, tm_hour=2, tm_min=14,
tm_sec=17, tm_wday=1, tm_yday=269, tm_isdst=0)

>>> t = time.time() - 10000
>>> time.gmtime(t)
time.struct_time(tm_year=2017, tm_mon=9, tm_mday=25, tm_hour=23, tm_min=31,
tm_sec=3, tm_wday=0, tm_yday=268, tm_isdst=0)
```

.localtime([secs])

将一个时间戳转换为当前时区的结构化时间。如果 secs 参数未提供，则以当前时间为准，即`time.time()`。

```
>>> time.localtime()
time.struct_time(tm_year=2017, tm_mon=9, tm_mday=26, tm_hour=10, tm_min=20,
tm_sec=42, tm_wday=1, tm_yday=269, tm_isdst=0)

>>> time.localtime(1406391907)
time.struct_time(tm_year=2014, tm_mon=7, tm_mday=27, tm_hour=0, tm_min=25,
tm_sec=7, tm_wday=6, tm_yday=208, tm_isdst=0)

>>> time.localtime(time.time() + 10000)
time.struct_time(tm_year=2017, tm_mon=9, tm_mday=26, tm_hour=13, tm_min=7,
tm_sec=54, tm_wday=1, tm_yday=269, tm_isdst=0)

```

.mktime(t)

将一个结构化时间转化为时间戳。 `time.mktime()` 执行与 `gmtime()`，`localtime()` 相反的操作，它接收 `struct_time` 对象作为参数,返回用秒数表示时间的浮点数。如果输入的值不是一个合法的时间，将触发 `OverflowError` 或 `ValueError`。

```
>>> time.mktime(1406391907)
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    time.mktime(1406391907)
TypeError: Tuple or struct_time argument required

>>> time.mktime(time.localtime())
1506393039.0
```

字符串

.ctime([secs])

把一个时间戳转化为本地时间的格式化字符串。默认使用 `time.time()` 作为参数。

```
>>> time.ctime()
'Tue Sep 26 10:22:31 2017'
>>> time.ctime(time.time())
'Tue Sep 26 10:23:51 2017'
>>> time.ctime(1406391907)
'Sun Jul 27 00:25:07 2014'
>>> time.ctime(time.time() + 10000)
'Tue Sep 26 13:11:55 2017'
```

.strftime(format [, t])

返回格式化字符串表示的当地时间。把一个 `struct_time`（如 `time.localtime()` 和 `time.gmtime()` 的返回值）转化为格式化的时间字符串，显示的格式由参数 `format` 决定。如果未指定 t，默认传入 `time.localtime()`。如果元组中任何一个元素越界，就会抛出 `ValueError` 的异常。

```
>>> time.strftime("%Y-%m-%d %H:%M:%S")
'2017-09-26 10:34:50'
>>> time.strftime("%Y-%m-%d %H:%M:%S",time.gmtime())
'2017-09-26 02:34:53'

```

.strptime(string[,format])

将格式化时间字符串转化成结构化时间。该方法是 `time.strftime()` 方法的逆操作。
`time.strptime()` 方法根据指定的格式把一个时间字符串解析为时间元组。要注意的是，你提供的字符串要和 format 参数的格式一一对应，如果 string 中日期间使用“-”分隔，format 中也必须使用“-”分隔，时间中使用冒号“:”分隔，后面也必须使用冒号分隔，否则会报格式不匹配的错误。并且值也要在合法的区间范围内，千万不要整出 14 个月来。

```
>>> import time
>>> stime = "2017-09-26 12:11:30"
>>> st = time.strptime(stime,"%Y-%m-%d %H:%M:%S")
>>> st
```

```
time.struct_time(tm_year=2017, tm_mon=9, tm_mday=26, tm_hour=12, tm_min=11,
tm_sec=30, tm_wday=1, tm_yday=269, tm_isdst=-1)
>>> for item in st:
    print(item)


2017
9
26
12
11
30
1
269
-1
>>> wrong_time = "2017-14-26 12:11:30"
>>> st  = time.strptime(wrong_time,"%Y-%m-%d %H:%M:%S")
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    st  = time.strptime(wrong_time,"%Y-%m-%d %H:%M:%S")
  File "C:\Python36\lib\_strptime.py", line 559, in _strptime_time
    tt = _strptime(data_string, format)[0]
  File "C:\Python36\lib\_strptime.py", line 362, in _strptime
    (data_string, format))
ValueError: time data '2017-14-26 12:11:30' does not match format '%Y-%m-%d
%H:%M:%S'

```

格式化时间字符串

利用 `time.strftime('%Y-%m-%d %H:%M:%S')` 等方法可以获得一个格式化时间字符串。

```
>>> time.strftime('%Y-%m-%d %H:%M:%S')
'2017-09-26 10:04:28'
```

注意其中的空格、短横线和冒号都是美观修饰符号，真正起控制作用的是百分符。对于格式化控制字符串 `"%Y-%m-%d %H:%M:%S`，其中每一个字母所代表的意思如下表所示，注意大小写的区别：

格式	含义
%a	本地星期名称的简写（如星期四为Thu）
%A	本地星期名称的全称（如星期四为Thursday）
%b	本地月份名称的简写（如八月份为agu）
%B	本地月份名称的全称（如八月份为august）
%c	本地相应的日期和时间的字符串表示（如：15/08/27 10:20:06）
%d	一个月中的第几天（01 - 31）
%f	微秒（范围0.999999）
%H	一天中的第几个小时（24小时制，00 - 23）
%I	第几个小时（12小时制，0 - 11）
%j	一年中的第几天（001 - 366）
%m	月份（01 - 12）
%M	分钟数（00 - 59）
%p	本地am或者pm的标识符
%S	秒（00 - 61）
%U	一年中的星期数。（00 - 53星期天是一个星期的开始。）第一个星期天之 前的所有天数都放在第0周。
%w	一个星期中的第几天（0 - 6，0是星期天）
%W	和%U基本相同，不同的是%W以星期一为一个星期的开始。
%x	本地相应日期字符串（如15/08/01）
%X	本地相应时间字符串（如08:08:10）
%y	去掉世纪的年份（00 - 99）两个数字表示的年份
%Y	完整的年份（4个数字表示年份）
%z	与UTC时间的间隔（如果是本地时间，返回空字符串）
%Z	时区的名字（如果是本地时间，返回空字符串）
%%	‘%’字符

时间格式之间的转换

Python 的三种类型时间格式，可以互相进行转换，如下图和下表所示：

从	到	方法
时间戳	UTC结构化时间	gmtime()
时间戳	本地结构化时间	localtime()
UTC结构化时间	时间戳	calendar.timegm()
本地结构化时间	时间戳	mktime()
结构化时间	格式化字符串	strftime()
格式化字符串	结构化时间	strptime()

datetime

`datetime` 包含用于处理日期和时间的函数和类。

datetime 类

datetime 是 date 与 time 的结合体，包括 date 与 time 的所有信息。其原型如下：

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

各参数的含义与date、time的构造函数中的一样，要注意参数值的范围。

datetime类定义的类属性与方法：

1. datetime.min、datetime.max：datetime所能表示的最小值与最大值；
2. datetime.resolution：datetime最小单位；
3. datetime.today()：返回一个表示当前本地时间的datetime对象；
4. datetime.now([tz])：返回一个表示当前本地时间的datetime对象，如果提供了参数tz，则获取tz参数所指时区的本地时间；
5. datetime.utcnow()：返回一个当前utc时间的datetime对象；
6. datetime.fromtimestamp(timestamp[, tz])：根据时间戳创建一个datetime对象，参数tz指定时区信息；
7. datetime.utcfromtimestamp(timestamp)：根据时间戳创建一个datetime对象；
8. datetime.combine(date, time)：根据date和time，创建一个datetime对象；
9. datetime.strptime(date_string, format)：将格式字符串转换为datetime对象，data 与 time 类没有提供该方法。

使用示例：

```
>>> datetime.datetime.min
datetime.datetime(1, 1, 1, 0, 0)
>>> datetime.datetime.max
datetime.datetime(9999, 12, 31, 23, 59, 59, 999999)
>>> datetime.datetime.resolution
datetime.timedelta(0, 0, 1)
```

```
>>> print datetime.datetime.resolution
0:00:00.000001
>>> today = datetime.datetime.today()
>>> today
datetime.datetime(2016, 5, 12, 12, 46, 47, 246240)
>>> datetime.datetime.now()
datetime.datetime(2016, 5, 12, 12, 47, 9, 850643)
>>> datetime.datetime.utcnow()
datetime.datetime(2016, 5, 12, 4, 47, 42, 188124)
>>> datetime.datetime.fromtimestamp(time.time())
datetime.datetime(2016, 5, 12, 12, 48, 40, 459676)
>>> datetime.datetime.combine(datetime.date(1990, 10, 05),
datetime.time(18, 18, 18))
datetime.datetime(1990, 10, 5, 18, 18, 18)
>>> datetime.datetime.strptime("2010-04-07 01:48:16.234000", "%Y-%m-%d
%H:%M:%S .%f")
datetime.datetime(2010, 4, 7, 1, 48, 16, 234000)
```

datetime 常用的实例方法与属性

datetime 类提供的实例方法与属性大部分功能与 date 和 time 类似，这里仅罗列方法名不再赘述：

1. datetime.year、month、day、hour、minute、second、microsecond、tzinfo：
2. datetime.date()：获取date对象；
3. datetime.time()：获取time对象；
4. datetime.ctime()：返回一个日期时间的C格式字符串，等效于 time.ctime(time.mktime(dt.timetuple()))；
5. datetime.strftime(format)

datetime 对象同样可以进行比较，或者相减返回一个时间间隔对象，或者日期时间加上一个间隔返回一个新的日期时间对象。

Date类

日期值用`date`类表示。实例具有属性`year`，`month`和`day`。使用`today()`类方法可以轻松创建当前日期。

date 类表示一个日期（由年、月、日组成），其原型如下：

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

参数说明：

1. year 的范围是 [MINYEAR, MAXYEAR]，即 [1, 9999]；
2. month 的范围是[1, 12]。（月份是从1开始的，不是从0开始）；
3. day 的最大值根据给定的year, month参数来决定。例如闰年2月份有29天；

date 类定义了一些常用的类方法与类属性：

1. date.max、date.min：date对象所能表示的最大、最小日期；
2. date.resolution：date对象表示日期的最小单位。这里是天。
3. date.today()：返回一个表示当前本地日期的 date 对象；
4. date.fromtimestamp(timestamp)：根据给定的时间戳，返回一个 date 对象；
5. datetime.fromordinal(ordinal)：将Gregorian日历时间转换为date对象；（Gregorian Calendar：一种日历表示方法，类似于我国的农历，西方国家使用比较多）

使用示例：

```
>>> datetime.date.max
datetime.date(9999, 12, 31)
>>> datetime.date.min
datetime.date(1, 1, 1)
>>> datetime.date.resolution
datetime.timedelta(1)
>>> datetime.date.today()
datetime.date(2016, 5, 12)
>>> datetime.date.fromtimestamp(time.time())
datetime.date(2016, 5, 12)
```

date提供的实例方法和属性：

1. date.year、date.month、date.day：年、月、日；
2. date.replace(year, month, day)：生成一个新的日期对象，用参数指定的年，月，日代替原有对象中的属性。（原有对象仍保持不变）
3. date.timetuple()：返回日期对应的time.struct_time对象；
4. date.toordinal()：返回日期对应的Gregorian Calendar日期；
5. date.weekday()：返回weekday，如果是星期一，返回0；如果是星期2，返回1，以此类推；
6. data.isoweekday()：返回weekday，如果是星期一，返回1；如果是星期2，返回2，以此类推；
7. date.isocalendar()：返回格式如(year，month，day)的元组；
8. date.isoformat()：返回格式如'YYYY-MM-DD'的字符串；
9. date.strftime(fmt)：自定义格式化字符串。

使用示例：

```
>>> today = datetime.date.today()
>>> today.year
2016
>>> today.month
5
>>> today.day
12
>>> tomorrow = today.replace(day=13)
>>> tomorrow
datetime.date(2016, 5, 13)
>>> tomorrow.timetuple()
time.struct_time(tm_year=2016, tm_mon=5, tm_mday=13, tm_hour=0, tm_min=0,
tm_sec=0, tm_wday=4, tm_yday=134, tm_isdst=-1)
>>> tomorrow.toordinal()
736097
>>> tomorrow.weekday()
4
>>> tomorrow.isoweekday()
5
>>> tomorrow.isocalendar()
(2016, 19, 5)
>>> tomorrow.isoformat()
'2016-05-13'
>>> tomorrow.strftime("%y-%m-%d")
'16-05-13'
```

date 重载了简单的运算符

```
#date 允许对日期进行加减和比较：
#日期加上一个间隔，返回一个新的日期对象
date2 = date1 + timedelta
#日期隔去间隔，返回一个新的日期对象
date2 = date1 - timedelta
#两个日期相减，返回一个时间间隔对象
timedelta = date1 - date2
#两个日期进行比较
date1 < date2
```

使用示例：

```
>>> now = datetime.date.today()
>>> now
datetime.date(2016, 5, 12)
>>> now += datetime.date.resolution
>>> now
datetime.date(2016, 5, 13)
>>> now -= datetime.date.resolution
>>> now
datetime.date(2016, 5, 12)
>>> now < datetime.date.max
True
```

Time类

time 类表示时间（由时、分、秒以及微秒组成），其原型如下：

```
class datetime.time(hour=0, minute=0, second=0, microsecond=0, tzinfo=None)
```

参数说明:

1. hour 的范围为[0, 24),
2. minute 的范围为[0, 60),
3. second 的范围为[0, 60),
4. microsecond 的范围为[0, 1000000),
5. tzinfo 表示时区信息。

time 类定义的类属性：

1. time.min、time.max：time类所能表示的最小、最大时间。其中，time.min = time(0, 0, 0, 0)，time.max = time(23, 59, 59, 999999)；
2. time.resolution：时间的最小单位，这里是1微秒；

使用示例：

```
>>> datetime.time.min
datetime.time(0, 0)
>>> datetime.time.max
datetime.time(23, 59, 59, 999999)
>>> datetime.time.resolution
datetime.timedelta(0, 0, 1)
```

time类提供的实例方法和属性：

1. time.hour、time.minute、time.second、time.microsecond：时、分、秒、微秒；
2. time.tzinfo：时区信息；

3. time.replace([hour[, minute[, second[, microsecond[, tzinfo]]]]])：创建一个新的时间对象，用参数指定的时、分、秒、微秒代替原有对象中的属性（原有对象仍保持不变）；
4. time.isoformat()：返回型如"HH:MM:SS"格式的字符串表示；
5. time.strftime(fmt)：返回自定义格式化字符串。

使用示例：

```
>>> tm = datetime.time(18, 18, 18)
>>> tm.hour
18
>>> tm.minute
18
>>> tm.second
18
>>> tm.microsecond
0
>>> tm.tzinfo
>>> tm.isoformat()
'18:18:18'
>>> tm.replace(hour=20)
datetime.time(20, 18, 18)
>>> tm.strftime("%I:%M:%S %p")
'06:18:18 PM'
```

time 类的对象只能进行比较，无法进行加减操作。

timedelta 类

datetime.timedelta 对象代表两个时间之间的的时间差，两个 date 或 datetime 对象相减时可以返回一个timedelta 对象。其原型如下：

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

所有参数可选，且默认都是0，参数的值可以是整数，浮点数，正数或负数。

内部只存储days，seconds，microseconds，其他参数的值会自动按如下规则抓转换：

1. 1 millisecond（毫秒） 转换成 1000 microseconds（微秒）
2. 1 minute 转换成 60 seconds
3. 1 hour 转换成 3600 seconds
4. 1 week转换成 7 days

```
b = a + datetime.timedelta(hours=5)
c = a + datetime.timedelta(weeks=1)
```

日期计算

日期计算使用标准算术运算符。

例:计算新中国诞生了多少天。

```
import datetime

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

```
print('Today    :', today)  # Today    : 2018-03-18

birth_day = datetime.date(1949, 10, 10)
diff = today - birth_day
print("新中国诞生了 %s 日" % diff.days)

one_day = datetime.timedelta(days=1)
print('一天时间  :', one_day)  # 一天时间  : 1 day, 0:00:00

yesterday = today - one_day
print('昨天:', yesterday)  # 昨天: 2018-03-17

tomorrow = today + one_day
print('明天 :', tomorrow)  # 明天 : 2018-03-19

print('明天 - 昨天:', tomorrow - yesterday)  # 2 days, 0:00:00
print('昨天 - 明天:', yesterday - tomorrow)  # -2 days, 0:00:00
```

示例说明了使用`timedelta`对象计算新日期，可以减去日期实例以生成 timedeltas（包括负 delta 值）。

`timedelta`对象还支持与整数，浮点数和其他`timedelta`对象进行算术运算。

日期和时间比较

可以使用标准比较运算符比较日期和时间值，以确定哪个更早或更晚。

```
import datetime
import time

print('Times:')
t1 = datetime.time(12, 55, 0)
print('  t1:', t1)
t2 = datetime.time(13, 5, 0)
print('  t2:', t2)
print('  t1 < t2:', t1 < t2)

print('Dates:')
d1 = datetime.date.today()
print('  d1:', d1)
d2 = datetime.date.today() + datetime.timedelta(days=1)
print('  d2:', d2)
print('  d1 > d2:', d1 > d2)

```

支持所有比较运算符。

合并 data 与 time

`datetime`实例具有`date`和`time`对象的所有属性。

与`date`一样，`datetime`为创建新实例提供了方便的类方法。

```python
import datetime

t = datetime.time(1, 2, 3)
print('t :', t)     # t : 01:02:03

d = datetime.date.today()
print('d :', d)     # d : 2018-03-18

dt = datetime.datetime.combine(d, t)
print('dt:', dt)    # dt: 2018-03-18 01:02:03
```

`combine()` 从一个 `date` 和一个 `time` 实例创建 `datetime` 实例。

字符串格式化和解析

datetime 对象的默认字符串表示形式使用 ISO-8601 格式（`YYYY-MM-DDTHH:MM:SS.mmmmmm`）。可以使用 `strftime()` 对其进行格式化。

```python
import datetime

ft = "%Y %m %d %H:%M:%S %Y"

today = datetime.datetime.today()
print('默认ISO     :', today)  # ISO     : 2018-03-18 16:20:34.941204

s = today.strftime(ft)
print('字符串格式化:', s)  # strftime: Sun Mar 18 16:20:34 2018
```

每个日期时间格式代码仍必须以前缀为前缀 `%`，后续冒号将作为文字字符处理，以包含在输出中。

符号	含义	例
`%a`	缩写的工作日名称	`'Wed'`
`%A`	完整的工作日名称	`'Wednesday'`
`%w`	工作日编号 - 0（星期日）至6（星期六）	`'3'`
`%d`	每月的一天（零填充）	`'13'`
`%b`	缩写的月份名称	`'Jan'`
`%B`	全月名称	`'January'`
`%m`	一年中的一个月	`'01'`
`%y`	没有世纪的一年	`'16'`
`%Y`	与世纪的一年	`'2016'`
`%H`	24小时制的小时	`'17'`
`%I`	12小时制的小时	`'05'`
`%p`	上午下午	`'PM'`
`%M`	分钟	`'00'`
`%S`	秒	`'00'`
`%f`	微秒	`'000000'`
`%z`	时区感知对象的UTC偏移量	`'-0500'`
`%Z`	时区名称	`'EST'`
`%j`	一年中的某一天	`'013'`
`%W`	一年中的一周	`'02'`
`%c`	当前区域设置的日期和时间表示形式	`'Wed Jan 13 17:00:00 2016'`
`%x`	当前区域设置的日期表示形式	`'01/13/16'`
`%X`	当前区域设置的时间表示	`'17:00:00'`
`%%`	文字`%`字符	`'%'`

时间戳互转

```
>>> import datetime
>>> today = datetime.datetime.today()
>>> today.timestamp()
1670482659.845911
>>> datetime.datetime.fromtimestamp(1670482659)
datetime.datetime(2022, 12, 8, 14, 57, 39)
```