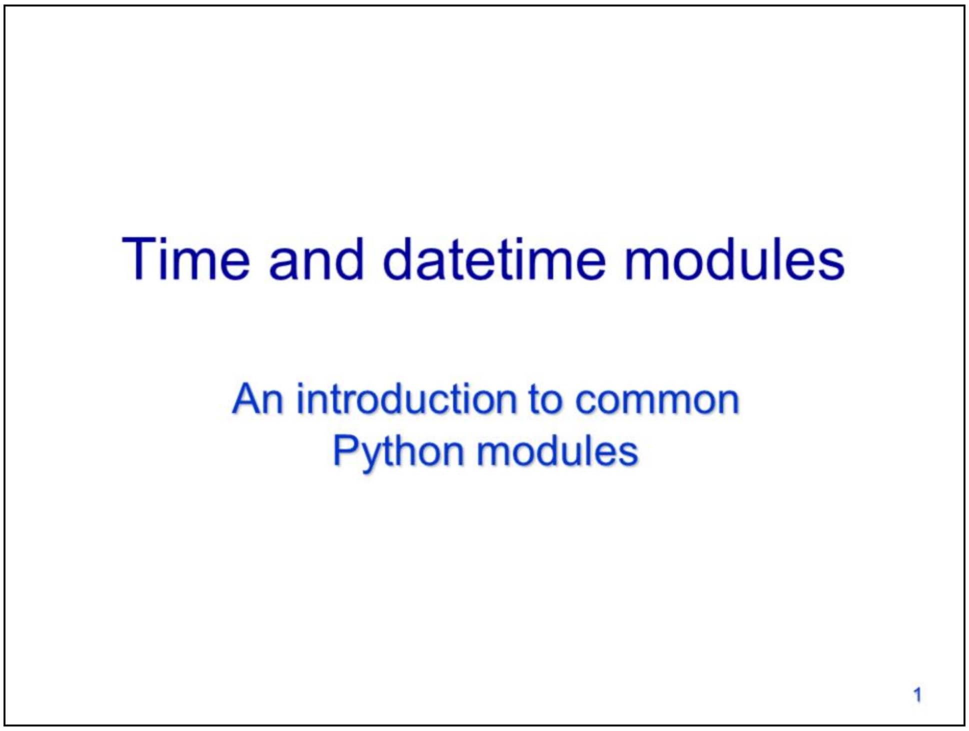
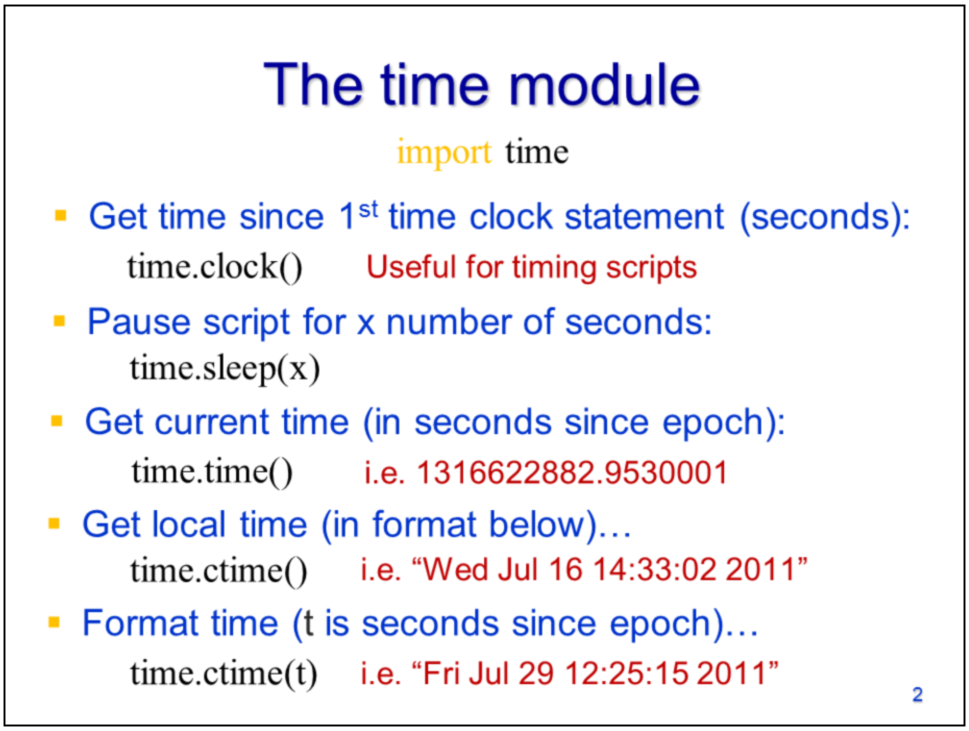


Time and datetime modules

An introduction to common Python modules

1

In this video, we will discuss the **time** and **datetime** modules.



The time module

```
import time
```

- Get time since 1st time clock statement (seconds):
  `time.clock()` Useful for timing scripts
- Pause script for x number of seconds:
  `time.sleep(x)`
- Get current time (in seconds since epoch):
  `time.time()` i.e. 1316622882.9530001
- Get local time (in format below)…
  `time.ctime()` i.e. "Wed Jul 16 14:33:02 2011"
- Format time (t is seconds since epoch)…
  `time.ctime(t)` i.e. "Fri Jul 29 12:25:15 2011"

2

The **time** module includes tools for clocking a script's processing time, getting current times, and formatting times.

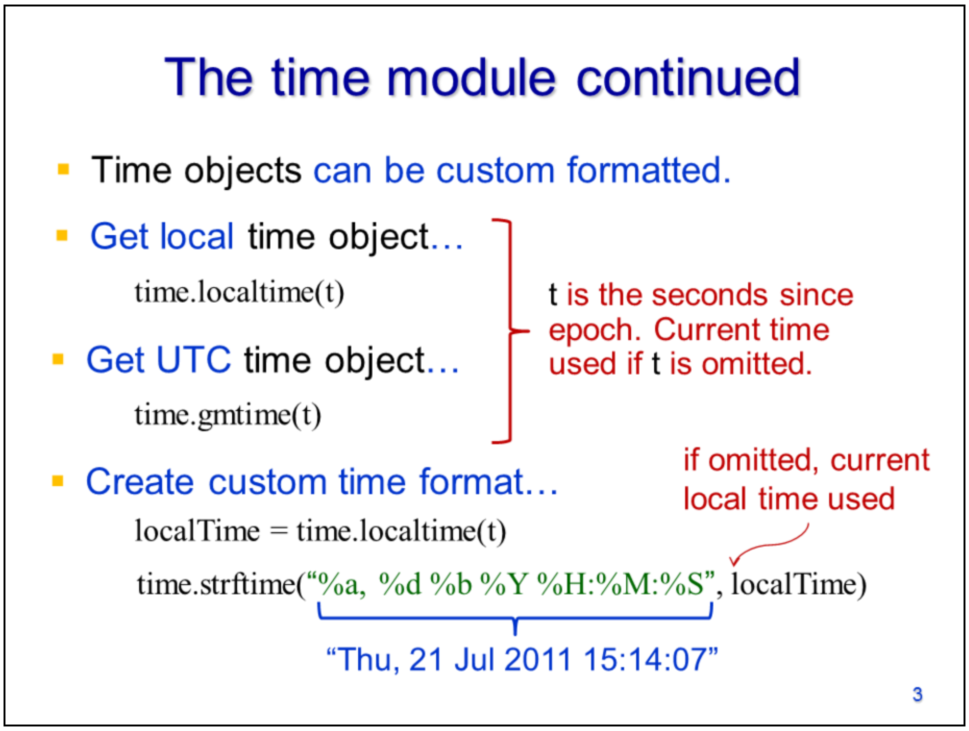
The time module's **clock** tool is used to check the time that it takes a script to complete a given task –this is a very useful tool for optimizing a script's efficiency, as we'll see in an example later.

The **sleep** tool will pause a script for a specified number of seconds. Making the script sleep can be useful when it must wait for a condition to occur before performing a certain task.

The **time** tool will get the current time in seconds since a computer epoch. This time can be converted to a more meaningful format by the **ctime** or **strftime** tools.

The **ctime** tool can get the local time in a standard time format

And it can also convert a time since epoch (e.g. from the **time** tool) into a standard time format.



The time module continued

- Time objects can be custom formatted.
- Get local time object…

```
time.localtime(t)
```

- Get UTC time object…

```
time.gmtime(t)
```

t is the seconds since epoch. Current time used if t is omitted.

- Create custom time format…

```
localTime = time.localtime(t)
time.strftime("%a,  %d %b %Y %H:%M:%S", localTime)
```

if omitted, current local time used

"Thu, 21 Jul 2011 15:14:07"

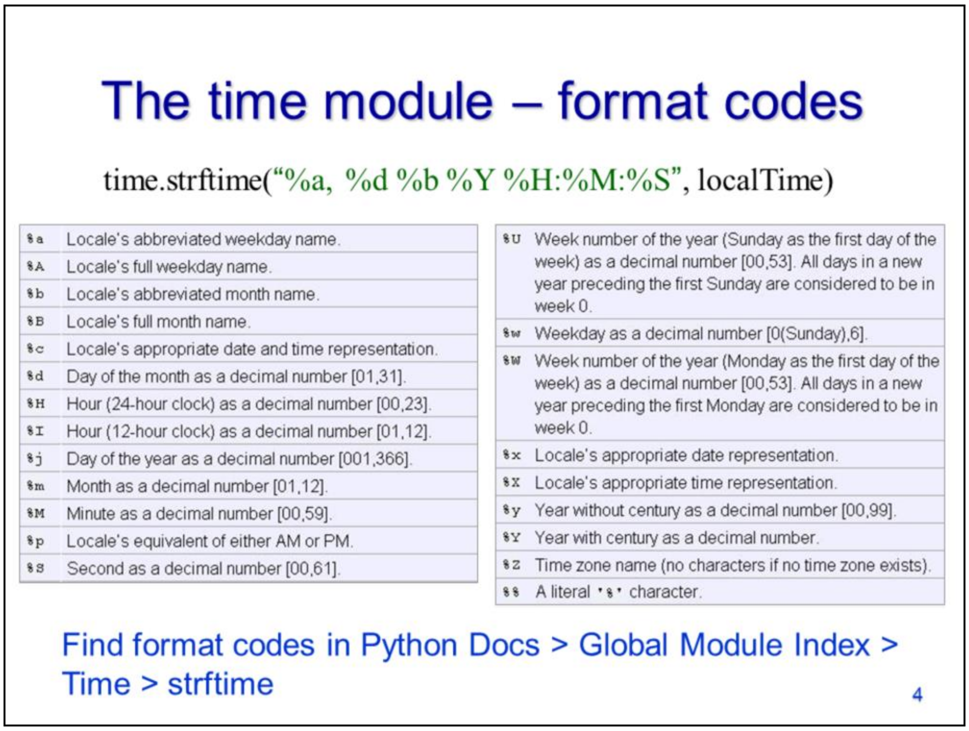
3

If a time format is required that is different from the output of the **ctime,** then custom time formats can be created.

The first step in creating a custom time format is to use the **localtime** or **gmtime** tools to convert a time since epoch to a **time object**. The gmtime tool converts the time to "Coordinated Universal Time". Both the localtime and gmtime tools will return the current time if no parameter is specified.

The time object can be input as a parameter in the **strftime** tool which allows the time format to be customized. The strftime tool allows a format string to be specified that determines the format of the output time.

An example result is shown for the format string used here.



The time module – format codes

time.strftime("%a, %d %b %Y %H:%M:%S", localTime)

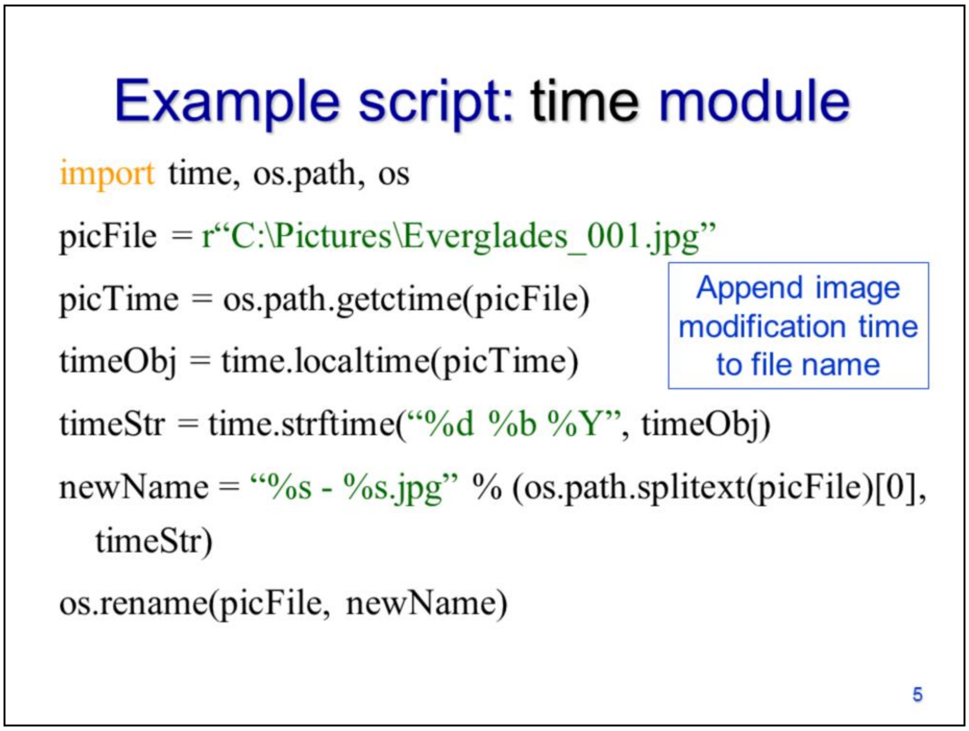
%a	Locale's abbreviated weekday name.
%A	Locale's full weekday name.
%b	Locale's abbreviated month name.
%B	Locale's full month name.
%c	Locale's appropriate date and time representation.
%d	Day of the month as a decimal number [01,31].
%H	Hour (24-hour clock) as a decimal number [00,23].
%I	Hour (12-hour clock) as a decimal number [01,12].
%j	Day of the year as a decimal number [001,366].
%m	Month as a decimal number [01,12].
%M	Minute as a decimal number [00,59].
%p	Locale's equivalent of either AM or PM.
%S	Second as a decimal number [00,61].

%U	Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0.
%w	Weekday as a decimal number [0(Sunday),6].
%W	Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0.
%x	Locale's appropriate date representation.
%X	Locale's appropriate time representation.
%y	Year without century as a decimal number [00,99].
%Y	Year with century as a decimal number.
%Z	Time zone name (no characters if no time zone exists).
%%	A literal '%' character.

Find format codes in Python Docs > Global Module Index > Time > strftime

4

This slide shows the available options for building a custom time format. Any combination of time elements may be used in the format string and spaces, commas, colons, or other punctuation may be used to separate the elements.



Example script: time module

```
import time, os.path, os
picFile = r“C:\Pictures\Everglades_001.jpg”
picTime = os.path.getctime(picFile)
timeObj = time.localtime(picTime)
timeStr = time.strftime(“%d %b %Y”, timeObj)
newName = “%s - %s.jpg” % (os.path.splitext(picFile)[0],
   timeStr)
os.rename(picFile, newName)
```

Append image modification time to file name

5

This example will show how to rename a digital photo so that the name includes the time the file was created.

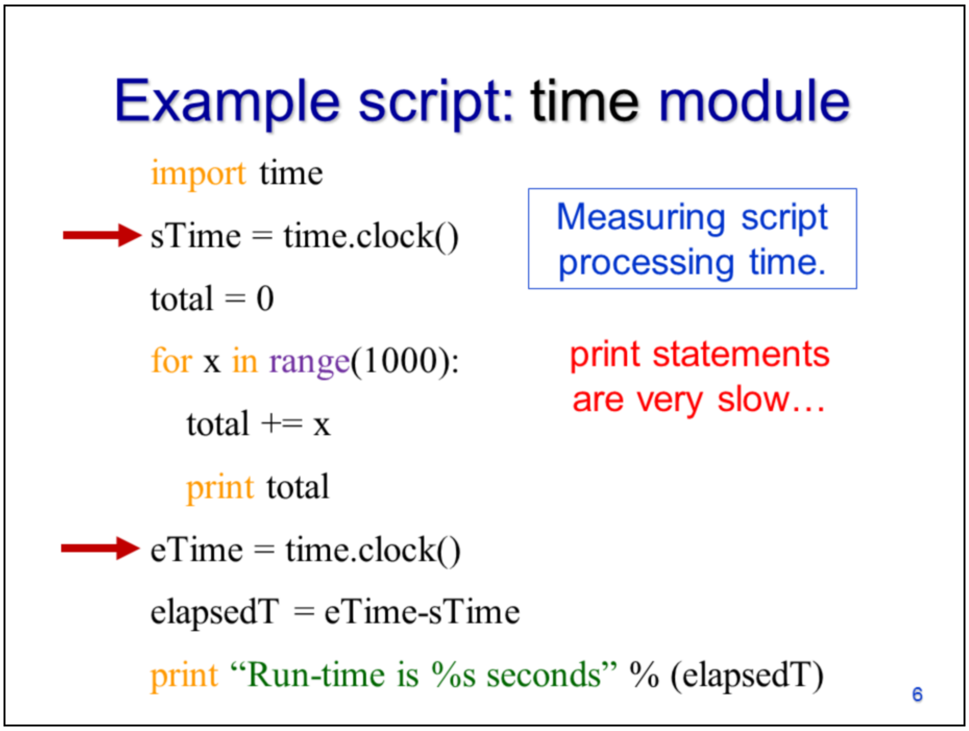
The first step is to get the file creation time using os.path’s **getctime** tool.

The creation time is then input into the time module’s **localtime** tool to create a **time object**.

The time object is then input into the **strftime** tool to create an output string with the desired time format.

The output string is then inserted into the output file name.

Finally, the os module’s **rename** tool is used to rename the file.



Example script: time module

```
import time
sTime = time.clock()
total = 0
for x in range(1000):
    total += x
    print total
eTime = time.clock()
elapsedT = eTime-sTime
print "Run-time is %s seconds" % (elapsedT)
```

Measuring script processing time.

print statements are very slow…

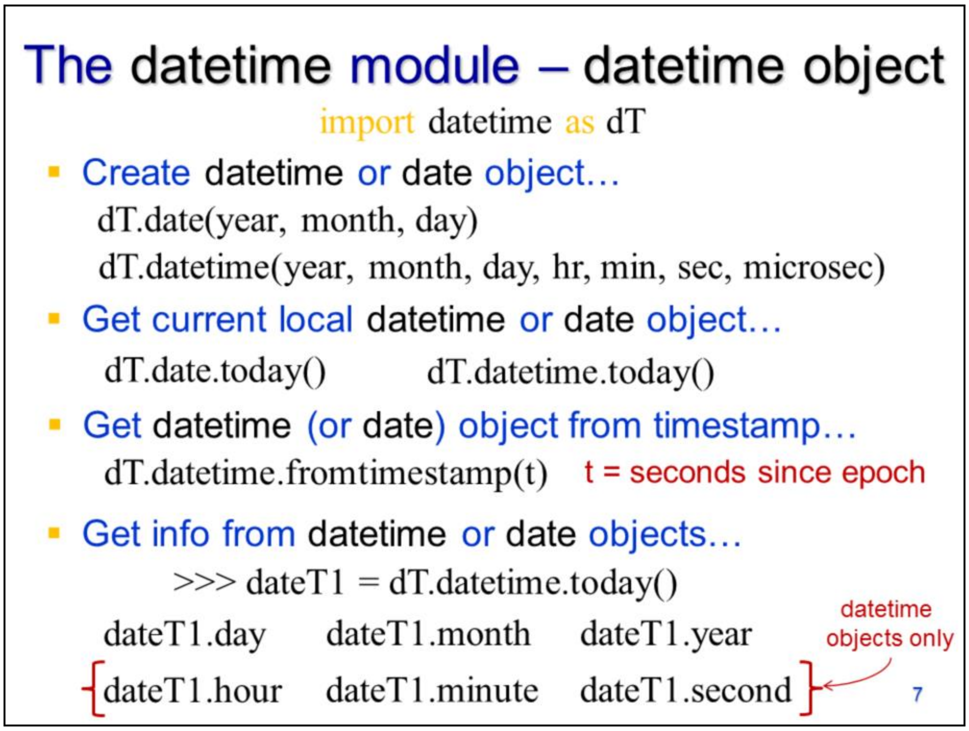
6

In this example, the time module’s clock tool is used to time how long the script takes to run.

The clock tool is used at the beginning of the script to get the start time.

It is used again at the end of the script to get the end time. The elapsed time is then calculated as the end time minus the start time.

If you run this script, you will find that it runs quite slowly. However, if you remove the print statement before running it, then it will run very fast. Print statements can slow a script significantly if printing occurs too often. If you need to include a print statement within a loop, then you should not have it execute in each iteration.


The datetime module – datetime object

```
import datetime as dT
```

- Create datetime or date object…
  ```
  dT.date(year, month, day)
  dT.datetime(year, month, day, hr, min, sec, microsec)
  ```
- Get current local datetime or date object…
  ```
  dT.date.today()        dT.datetime.today()
  ```
- Get datetime (or date) object from timestamp…
  ```
  dT.datetime.fromtimestamp(t)
  ```
  t = seconds since epoch
- Get info from datetime or date objects…
  ```
  >>> dateT1 = dT.datetime.today()
  dateT1.day     dateT1.month     dateT1.year
  dateT1.hour    dateT1.minute    dateT1.second
  ```
  datetime objects only

7

The uses for the **datetime** module include comparing dates and calculating the elapsed time between two dates. **Date** or **datetime** objects need to be created in order to compare dates or calculate elapsed time between dates. The **date** object includes only the year, month, and day.

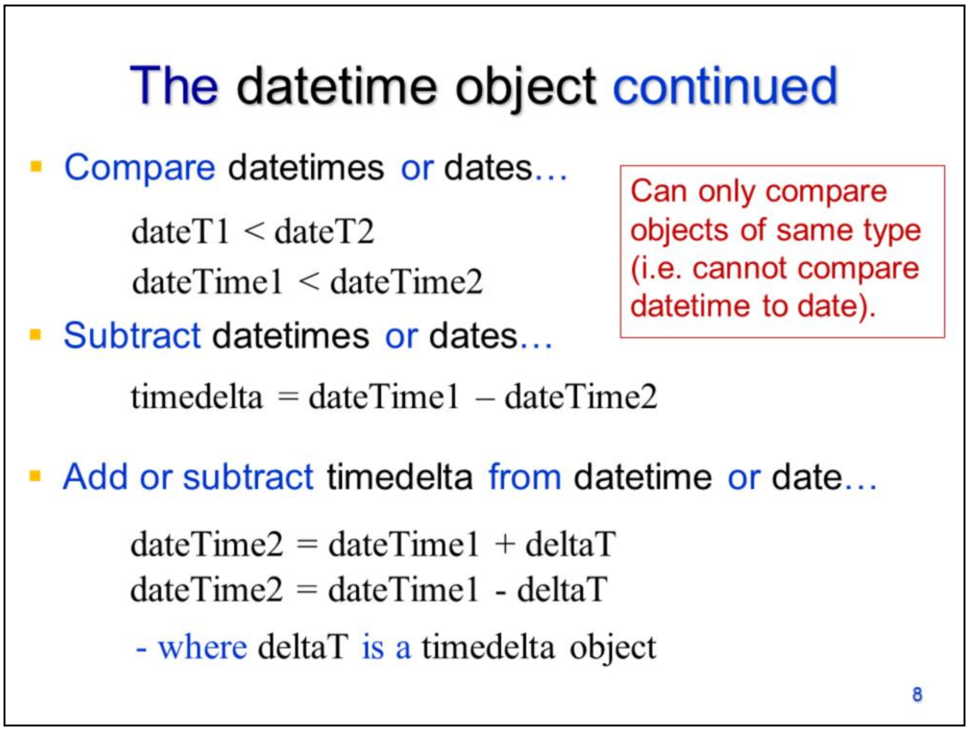
The **datetime** object includes the date as well as the time of day.

The **date.today** tool creates a date object corresponding to the current date

The **datetime.today** tool creates a datetime object corresponding to the current date and time.

The **datetime.fromtimestamp** tool can create a datetime object from a “time since epoch”.

Date objects give access to the day, month, and year to be extracted. Datetime objects give access to the day, month, year as well as the hour, minute, and second.



The datetime object continued

- Compare datetimes or dates…

  dateT1 < dateT2

  dateTime1 < dateTime2

- Subtract datetimes or dates…

  timedelta = dateTime1 – dateTime2

- Add or subtract timedelta from datetime or date…

  dateTime2 = dateTime1 + deltaT

  dateTime2 = dateTime1 - deltaT

  - where deltaT is a timedelta object

Can only compare objects of same type (i.e. cannot compare datetime to date).

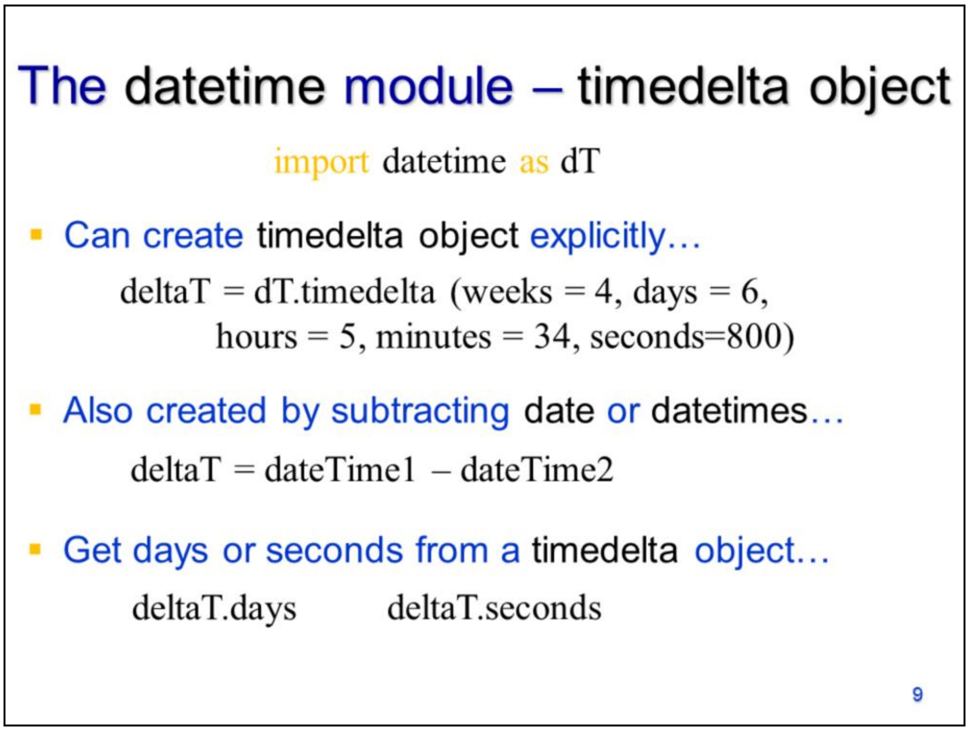
8

Date or datetime objects can only work with other objects of the same type – date objects work with date objects; datetime objects work with datetime objects.

Date or datetime objects can be compared…

Or subtracted. Subtracting the objects creates a **time delta** object.

Time delta objects can be added to or subtracted from date or datetime objects.



The datetime module – timedelta object

```
import datetime as dT
```

- Can create timedelta object explicitly…

```
deltaT = dT.timedelta (weeks = 4, days = 6,
         hours = 5, minutes = 34, seconds=800)
```

- Also created by subtracting date or datetimes…

```
deltaT = dateTime1 – dateTime2
```

- Get days or seconds from a timedelta object…

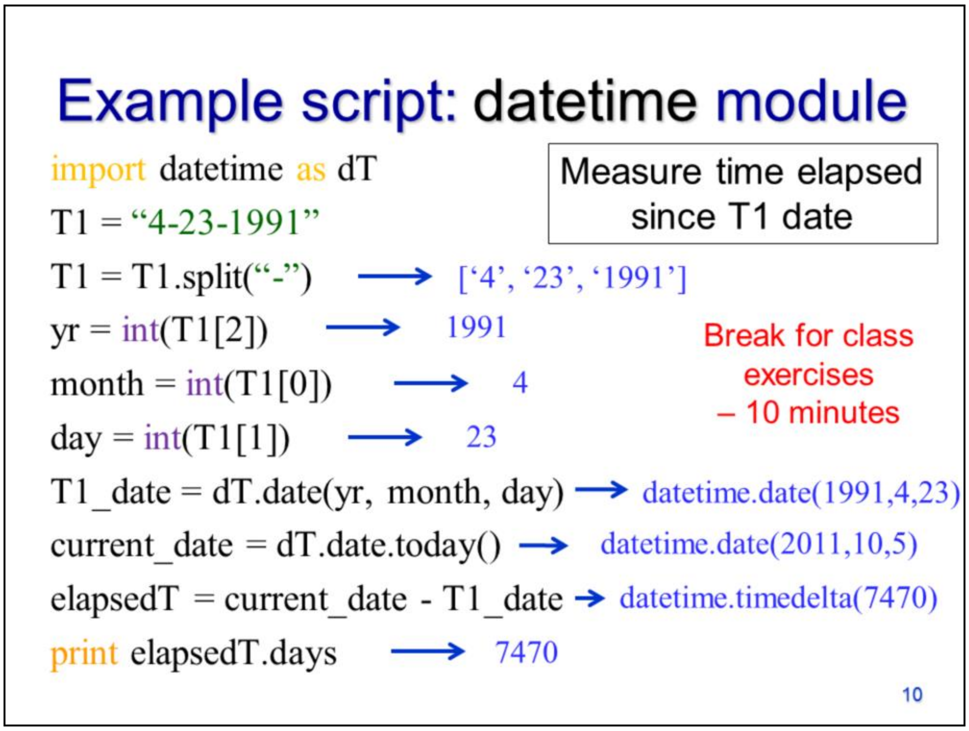
```
deltaT.days        deltaT.seconds
```

9

Time delta objects can be created explicitly using the **timedelta** tool.

They can also be created by subtracting date or datetime objects.

The days and seconds can be retrieved from a timedelta object.



Example script: datetime module

Measure time elapsed since T1 date

```
import datetime as dT
T1 = "4-23-1991"
T1 = T1.split("-")                         ⟶ ['4', '23', '1991']
yr = int(T1[2])                            ⟶ 1991
month = int(T1[0])                         ⟶ 4
day = int(T1[1])                           ⟶ 23
T1_date = dT.date(yr, month, day)          ⟶ datetime.date(1991,4,23)
current_date = dT.date.today()             ⟶ datetime.date(2011,10,5)
elapsedT = current_date - T1_date          → datetime.timedelta(7470)
print elapsedT.days                        ⟶ 7470
```

Break for class exercises – 10 minutes

10

Let's look at an example using the datetime module which measures the amount of time that has passed between a past date and today.

A date string, for the past date, is split to extract the year, month, and day.

A date object is created using the components extracted from the original date string in the **date** tool.

A date object for today's date is obtained using the **date.today** tool.

The date object for the past date is subtracted from the date object for today's date.

The days is then extracted from the **timedelta** object.