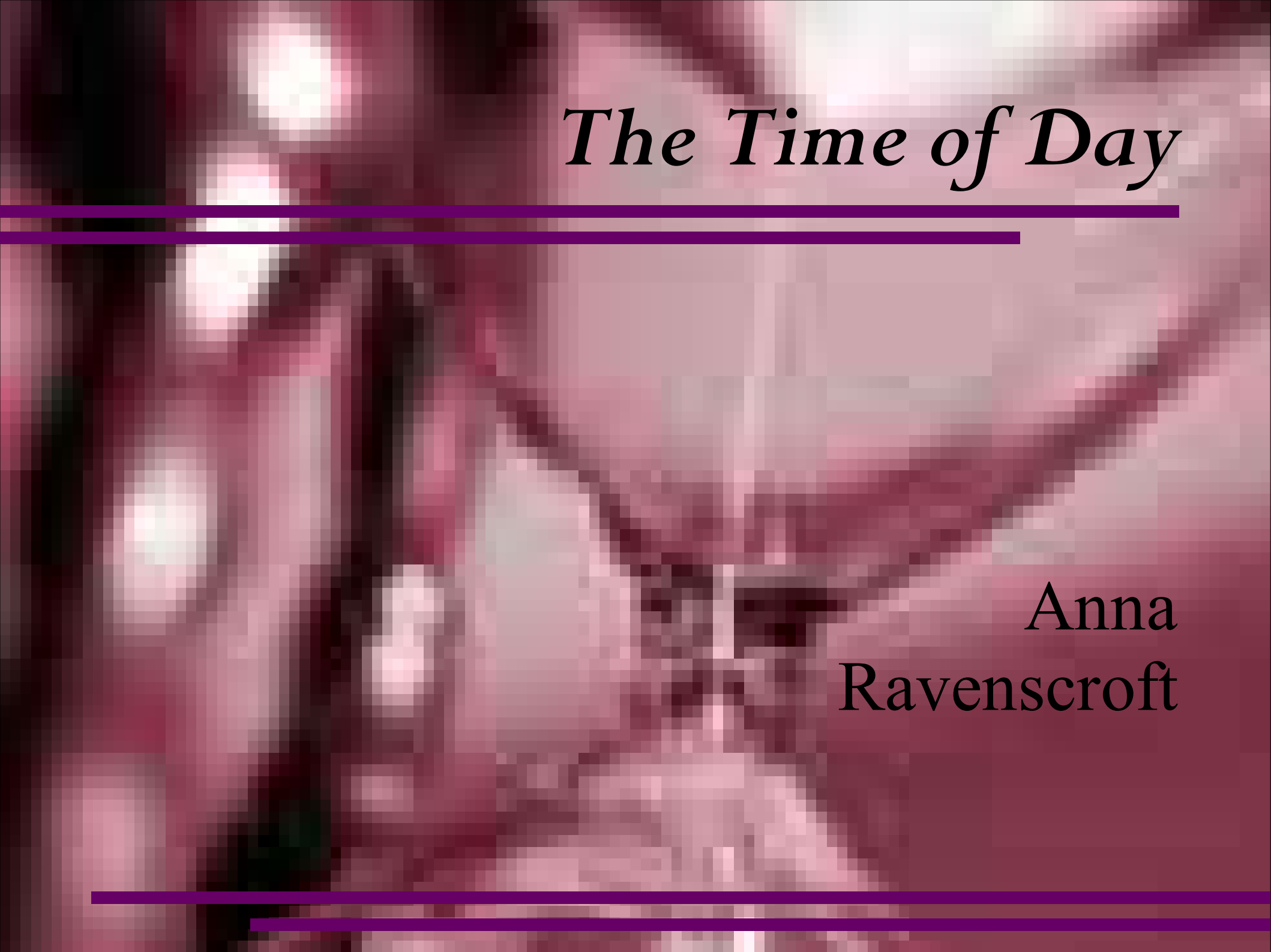
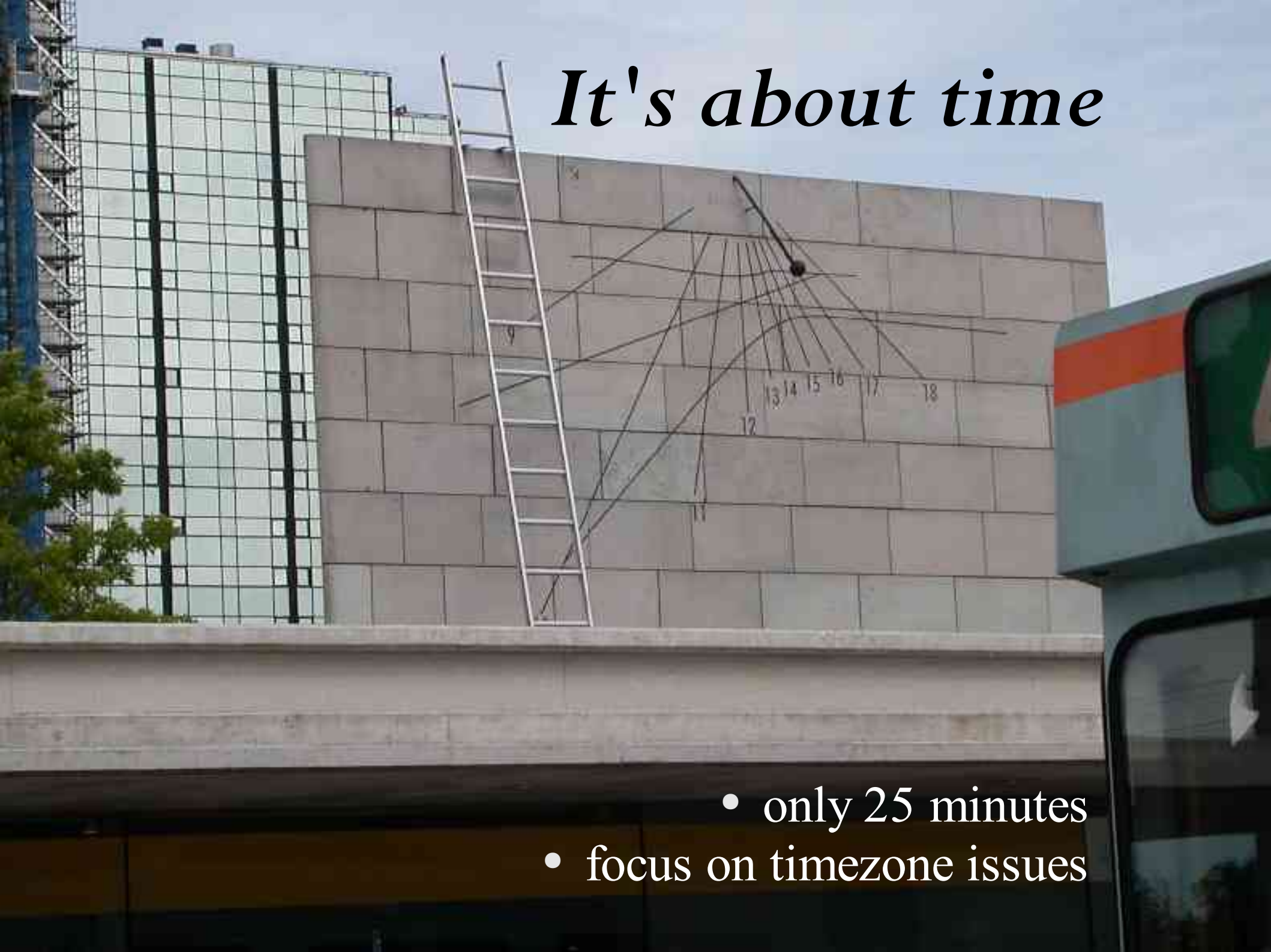

The Time of Day
Anna
Ravenscroft


It's about time
• only 25 minutes
• focus on timezone issues

Telling Time in Python

- `time`
- `datetime`
- `dateutil`
- `pytz`

Does anybody really know what time it is?

NOW according to `time.time()`

```
>>> time.time()
1086539269.711
```

- returns floating point number
- seconds from the epoch

The Epoch

- time.asctime(time.gmtime(0))
- (midnight, Jan 1, 1970)

NOW according to `time.asctime()`

- can use time.asctime() instead

```
>>> time.asctime()
'Sun Jun 06 18:25:06 2004'
```

- more readable
- pain to parse
- tends to fail when internationalized

the time Tuple

- `time.localtime()` or `time.gmttime()`
- format: year, month, day, hour, minute, second, weekday, yearday, DST flag
- sliceable

```
>>> time.localtime()
(2004, 6, 5, 18, 49, 10, 5, 157, 1)
>>> time.gmtime()
(2004, 6, 5, 16, 50, 4, 5, 157, 0)
```

DST Flag

```
>>> time.localtime()[-1]
(1)
```

- whether DST is in effect right now
  - 0 means "not in effect"
  - 1 means "in effect"
  - -1 means "heckifIknow"

- gmtime DST flag is always 0

Daylight Savings

```
>>> time.daylight
1
```

- Does my tz use DST?

```
>>> bool(time.localtime().tm_isdst)
False
>>>
```

- Is DST currently in effect?

Time operations: yesterday

- time.time()-86400

```
>>> time.time()
1086454724.4119999
>>> time.time()-86400
1086368335.6589999
>>> yesterday =
... time.localtime(time.time()-86400)
>>> print yesterday
(2004, 6, 4, 19, 19, 53, 4, 156, 1)
>>>
```

Time allows string formatting

- `time.strftime(formatstring, timetuple)`
  - takes a format string and a 9-item time tuple
- format string (kinda like string % operators)
  - `%x` gives a complete date representation
  - `%Y` (4 digits) or `%y` (2 digits)
  - `%M` (1-12) or `%b`(abbr name) or `%B`(fullname)
  - `%d`(day of month) or `%j` (day of year)
  - `%a`(abbr dayname) or `%A`(full dayname)
  - *table on page 248 of Python in a Nutshell*

Formatting Yesterday (example)

```
>>> time.strftime("%x", yesterday)
'03/22/05'     # C (US) date format: mdy
>>> time.strftime("Yesterday was %b %d,
  %y", yesterday)
'Yesterday was Mar 22, 05'
>>> time.strftime("Yesterday was %A, day
  %j of %Y")
'Yesterday was Tuesday, day 081 of 2005'
```

String parsing with `Time`

- `time.strptime(str, formatstring)`
  - takes a string, formatting
  - returns a 9-item time tuple
- same formatting rules as strftime
- works on all platforms since Python 2.3

```
>>> time.strptime("2004-06-05",
                  "%Y-%m-%d")
(2004, 6, 5, 0, 0, 0, 5, 157, -1)
>>>
```

Using the `datetime` *module*

- introduced in 2.3
- separate date, time, and datetime objects
- naïve or aware (tz, DST)
- tzinfo can be subclassed to support timezones, DST, etc.

NOW according to ***datetime*** *module*

- create a datetime object

```
>>> datetime.datetime.now()
datetime.datetime (2005, 3, 22, 18, 23,
41, 537000)
```

- format:ymdhmsu
- now is based on localtime
- naïve (doesn't care about timezone)

Using datetime

- make your life easier with import

```
>>> from datetime import datetime, timedelta
```

- result a method (or operation)

```
>>> d = datetime.now()
```

- construct your own

```
>>> y = datetime(2005, 3, 22, 10, 29)
```

datetime attributes

- datetime objects have attributes

dir(d)

```
>>> dir(d)
[ 'ctime', 'date', 'day', 'dst', 'fromordinal',
  'fromtimestamp', 'hour', 'isocalendar',
  'isoformat', 'isoweekday', 'max', 'microsecond',
  'min', 'minute', 'month', 'now', 'replace',
  'resolution', 'second', 'strftime', 'time',
  'timetuple', 'timetz', 'today', 'toordinal',
  'tzinfo', 'tzname', 'utcfromtimestamp', 'utcnow',
  'utcoffset', 'utctimetuple', 'weekday', 'year']
>>>
```

datetimeand the timetuple

```
>>> datetime.now().timetuple()
(2005, 3, 24, 4, 21, 34, 3, 83, -1)
>>>
```

datetime operations

- adding and subtracting
  - the wrong way

```
>>> d = datetime.now()
>>> x = d-1
```

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in -toplevel-
    y = d-1
TypeError: unsupported operand type(s) for -:
  'datetime.datetime' and 'int'
```

datetime operations

- adding and subtracting
  - the right way

```
>>> d
>>> x = d-timedelta(minutes=1)
>>> y = d + timedelta(days=1)
```

- also use for seconds, microseconds
- `dateutil` for fancier control:
  - weeks, months, years, etc.

Time is fleeting

- naive time
- timezone/DST aware

Datetime objects

- datetime default is naïve

```
>>> d.tzinfo
>>> print d.tzinfo
None
```

Aware datetime objects

- pass tzinfo at instantiation

```
>>> a = datetime(2005, 3, 22, 10, 29, tzinfo=eastern)
>>> a
datetime.datetime(2005, 3, 22, 10, 29,
    tzinfo=<DstTzInfo 'US/Eastern' EST-1 day, 19:00:00
    STD>)
>>>
```

aware now

- aware now is also pretty easy...

```
>>> anow=datetime.now(eastern)
>>> anow
datetime.datetime(2005, 3, 22, 4, 42, 52, 3,
   tzinfo=<DstTzInfo 'US/Eastern' EST-1 day, 19:00:00
   STD>)
>>>
```

aware is gullible

- it believes whatever tzinfo you pass it

```
>>> agul=datetime.now(kat)
>>> agul
datetime.datetime(2005, 3, 24, 4, 49, 23, 3,
   tzinfo=<DstTzInfo 'Asia/Katmandu' LMT+5:41:00
   STD>)
```

Defining timezones

- subclass `tzinfo`
- `dateutil` third-party module
- `pytz` third-party module

Let's do the time warp again!

What is the `dateutil` *module?*

- powerful third-party set of extensions to datetime
- utilities for common date operations
  - relative deltas
  - recurrence rules
  - parsing of almost any string format
  - timezone (tzinfo) implementations
  - Easter Sunday
  - 400+ test cases
- straightforward, understandable documentation

dateutil *timezone support*

- based on posix strings
- very user-configurable

```
from dateutil import tz
import datetime
posixstr =
"CET-1CEST-2,M3.5.0/02:00,M10.5.0/03:00"
spaintz = tz.tzstr(posixstr)
print datetime.datetime.now(spaintz).
  ctime()
'Thu Mar 24 04:58:08 2005'
```

`dateutil` *timezone support*

- tzlocal
- utc offset
- tzfile
- tzical
- useful, a little complex
  - timezones have to be complex, right?

Easy as **`pytz`**

- Only does timezone support

```
>>> from pytz import timezone
>>> utc=timezone('UTC')
>>> eastern=timezone('US/Eastern')
```

- based on Olson database
  - case sensitive

```
>>> kat = timezone('Asia/Katmandu')
```

If it's wrong, talk to Olson

*Stuart sez: Use **UTC***

The preferred way of dealing with times is to always work in UTC, converting to localtime only when generating output to be read by humans.

Why?

Timing and effects of DST

Daylight Saving Time begins for most of the United States at 2 am on the first Sunday of April. Time reverts to standard time at 2 a.m. on the last Sunday of October. In the U.S., each time zone switches at a different time.

April 3 will have 1:00:1:59 twice, in each TZ

In the European Union, Summer Time begins and ends at 1 am Universal Time (Greenwich Mean Time). It starts the last Sunday in March, and ends the last Sunday in October. In the EU, all time zones change at the same moment.

March 27 will have 12:00:12:59 twice, across EU

UTC Conversion with **pytz**

```
>>> utc=timezone('UTC')
>>> ut = datetime(2005, 3, 22, 11,
  10, tzinfo=utc)
>>> myut = ut.astimezone(eastern)
>>> katut = ut.astimezone(kat)
```

Conversion results

```
>>> ut
datetime.datetime(2005, 3, 22, 11, 10,
  tzinfo=<StaticTzInfo 'UTC'>)
>>> myut
datetime.datetime(2005, 3, 22, 6, 10,
  tzinfo=<DstTzInfo 'US/Eastern' EST-1
  day, 19:00:00 STD>)
>>> katut
datetime.datetime(2005, 3, 22, 16, 55,
  tzinfo=<DstTzInfo 'Asia/Katmandu'
  NPT+5:45:00 STD>)
```

Olson timezones

- maintained by Arthur David Olson
  - at least 1986, lots of volunteers
  - lots of great info on time: http://www.twinsun.com/tz/tz-link.htm including a mailing list
- coordinates with IERS
  - International Earth Rotation and Reference Systems Service
- list of names of timezones
  - http://s.keim.free.fr/tz/tznames/