

It's about time!

Aymeric Augustin - @aymericaugustin

PyConFR - 2012-09-16T14:30:00+02:00

RFC 3339

2012-09-16T14:30:00.000+02:00

- current era
- stated offset
- universal time
- instant in time

“a profile of the ISO 8601 standard for representation of dates and times using the Gregorian calendar”

“does not cater to local time zone rules”

in Python

```
>>> from datetime import datetime
>>> datetime(
...     year=2012, month=9, day=16,
...     hour=14, minute=30, second=0,
...     microsecond=0,
...     tzinfo=
```

`pip install pytz`

“No concrete tzinfo classes are supplied by the datetime module.”

in Python with pytz

```
>>> from pytz import FixedOffset
>>> from datetime import datetime
>>> datetime(
...     year=2012, month=9, day=16,
...     hour=14, minute=30, second=0,
...     microsecond=0,
...     tzinfo=FixedOffset(120))
datetime.datetime(2012, 9, 16, 14, 30,
tzinfo=pytz.FixedOffset(120))
>>> _.isoformat()
'2012-09-16T14:30:00+02:00'
```

human-friendly zones

```
>>> from pytz import timezone
>>> from datetime import datetime
>>> datetime(
...     year=2012, month=9, day=16,
...     hour=14, minute=30, second=0,
...     microsecond=0,
...     tzinfo=timezone("Europe/Paris"))
datetime.datetime(2012, 9, 16, 14, 30,
tzinfo=<DstTzInfo 'Europe/Paris'
PMT+0:09:00 STD>)      not CEST+2:00:00?
```

“To create local wallclock times, use the localize() method.”

human-friendly zones

```
>>> from pytz import timezone
>>> from datetime import datetime
>>> tz = timezone("Europe/Paris")
>>> tz.localize(datetime(
...     year=2012, month=9, day=16,
...     hour=14, minute=30, second=0,
...     microsecond=0))
datetime.datetime(2012, 9, 16, 14, 30,
tzinfo=<DstTzInfo 'Europe/Paris'
CEST+2:00:00 DST>)
>>> _.isoformat()
'2012-09-16T14:30:00+02:00'
```

it's complicated!
I don't need that!

it's complex
we shall see

aware vs. naive datetimes

```
>>> naive = datetime(2012, 9, 16, 14, 35)
>>> tz = timezone("Europe/Paris")
>>> aware = tz.localize(naive)
```

```
>>> naive == aware
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: can't compare offset-naive and  
offset-aware datetimes
```

aware vs. naive datetimes

```
>>> naive = datetime(2012, 9, 16, 14, 35)
>>> tz = timezone("Europe/Paris")
>>> aware = tz.localize(naive)
```

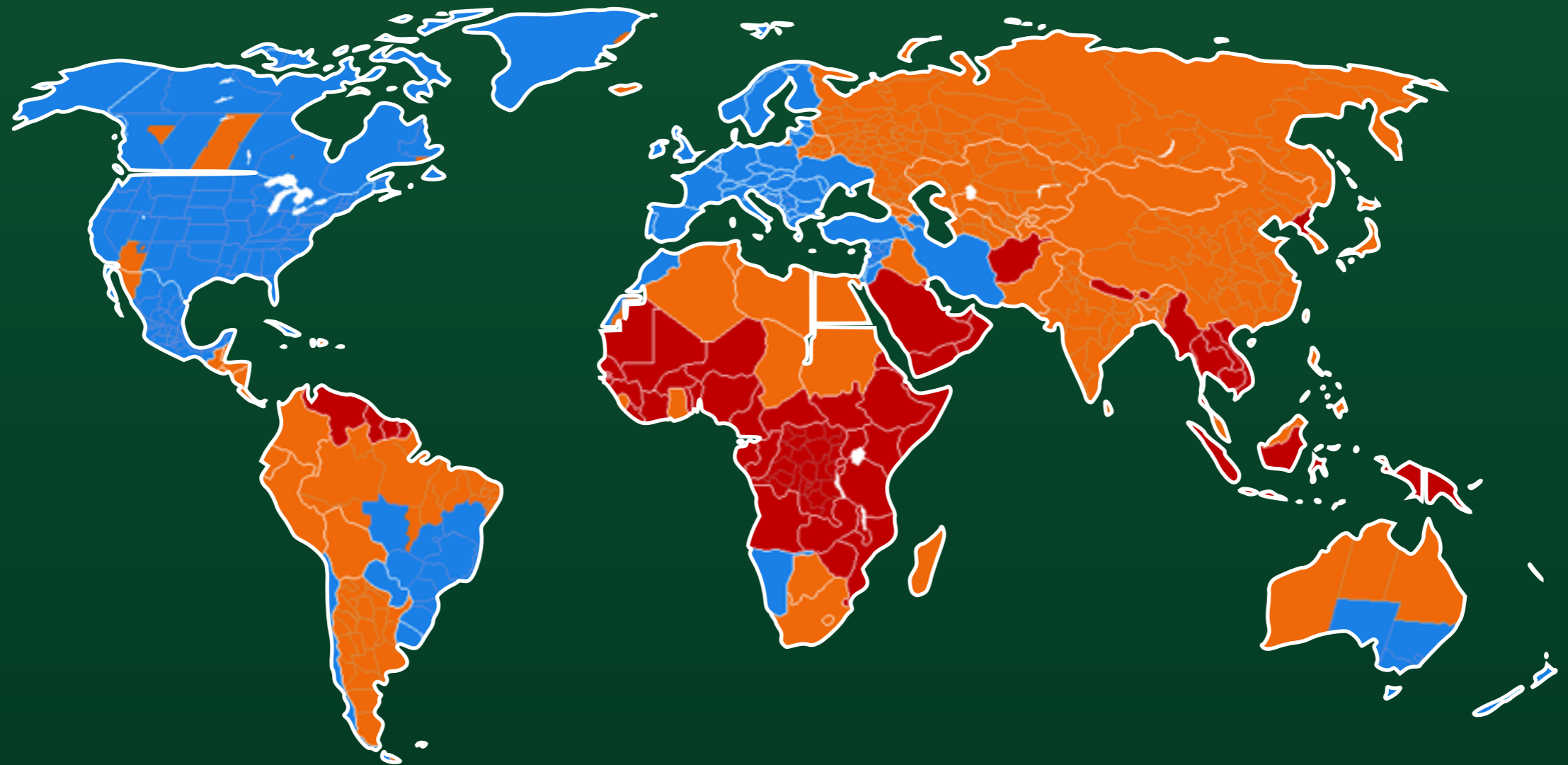
```
>>> naive - aware
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't subtract offset-naive
and offset-aware datetimes
```

aware vs. naive datetimes

“Whether a naive datetime object represents Coordinated Universal Time (UTC), local time, or time in some other timezone is purely up to the program.”

“Naive datetime objects are easy to understand and to work with, at the cost of ignoring some aspects of reality.”



countries with DST

DST transitions

spring

01:xx CET

03:xx CEST

autumn

01:xx CEST

02:xx CEST

02:xx CET

03:xx CET

DST

```
>>> from datetime import datetime
```

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

```
>>> hour = timedelta(hours=1)
```

```
>>> from pytz import timezone
```

```
>>> tz = timezone("Europe/Paris")
```

DST

```
>>> dt1 = datetime(2012, 3, 25, 1, 30)
```

```
>>> dt1
```

```
datetime.datetime(2012, 3, 25, 1, 30)
```

```
>>> _ + hour
```

```
datetime.datetime(2012, 3, 25, 2, 30)
```

DST

```
>>> tz.localize(dt1)
datetime.datetime(2012, 3, 25, 1, 30,
tzinfo=<DstTzInfo 'Europe/Paris' CET
+1:00:00 STD>)
```

```
>>> _ + hour
```

```
datetime.datetime(2012, 3, 25, 2, 30,
tzinfo=<DstTzInfo 'Europe/Paris' CET
+1:00:00 STD>)
```

this doesn't actually exist!

"A normalize() method is provided to correct this."

DST

```
>>> tz.localize(dt1)
datetime.datetime(2012, 3, 25, 1, 30,
tzinfo=<DstTzInfo 'Europe/Paris' CET
+1:00:00 STD>)
```

```
>>> tz.normalize(_ + hour)
datetime.datetime(2012, 3, 25, 3, 30,
tzinfo=<DstTzInfo 'Europe/Paris' CEST
+2:00:00 DST>)
```

DST

```
>>> dt2 = datetime(2012, 10, 28, 2, 30)
```

```
>>> dt2  
datetime.datetime(2012, 10, 28, 2, 30)
```

```
>>> _ - hour  
datetime.datetime(2012, 10, 28, 1, 30)
```

DST

```
>>> tz.localize(dt2)
datetime.datetime(2012, 10, 28, 2, 30,
tzinfo=<DstTzInfo 'Europe/Paris' CET
+1:00:00 STD>)
```

```
>>> tz.normalize(_ - hour)
datetime.datetime(2012, 10, 28, 2, 30,
tzinfo=<DstTzInfo 'Europe/Paris' CEST
+2:00:00 DST>)
```

DST

```
>>> tz.localize(dt2, is_dst=False)
datetime.datetime(2012, 10, 28, 2, 30,
tzinfo=<DstTzInfo 'Europe/Paris' CET
+1:00:00 STD>)
```

```
>>> tz.localize(dt2, is_dst=True)
datetime.datetime(2012, 10, 28, 2, 30,
tzinfo=<DstTzInfo 'Europe/Paris' CEST
+2:00:00 DST>)
```

ambiguous time

```
>>> dt2  
datetime.datetime(2012, 10, 28, 2, 30)
```

```
>>> tz.localize(_, is_dst=None)  
Traceback (most recent call last):  
...  
pytz.exceptions.AmbiguousTimeError:  
2012-10-28 02:30:00
```

non existent time

```
>>> dt1 + hour  
datetime.datetime(2012, 3, 25, 2, 30)
```

```
>>> tz.localize(_, is_dst=None)  
Traceback (most recent call last):  
...  
pytz.exceptions.NonExistentTimeError:  
2012-03-25 02:30:00
```

leap years

```
>>> from datetime import date  
>>> date(2012, 2, 29)  
datetime.date(2012, 2, 29)
```

leap seconds

```
>>> from datetime import datetime
>>> datetime(2012, 6, 30, 23, 59, 60)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: second must be in 0..59
```


“I’ll use naive datetimes in UTC”

```
>>> import datetime
```

```
>>> datetime.datetime.utcnow()  
datetime.datetime(2012, 9, 16, 12, 52,  
25, 521458)
```

Will one of your libraries ever call `datetime.datetime.now()`?

“I’ll use naive datetimes in UTC”

```
>>> import datetime, os, time
>>> os.environ['TZ'] = 'UTC'
>>> time.tzset()

>>> datetime.datetime.now()
datetime.datetime(2012, 9, 16, 12, 52,
25, 521458)
```

Are you ever going to display or process a date and time in local time?

“I’ll use naive
datetimes in UTC”

“I’ll use bytestrings
encoded in UTF-8”

unicode vs. datetime

- “there ain’t no such thing as plain text”

Joel Spolsky

- “software should only work with unicode strings internally, converting to a particular encoding on output”

Python’s Unicode HOTW

- “there ain’t no such thing as a naive datetime”

yours truly

- “always work in UTC, converting to localtime only when generating output to be read by humans”

pytz’ documentation

take advantage
of the API safety

what about dates?

dates and datetimes

```
>>> from datetime import datetime
>>> from datetime import date, time

>>> issubclass(datetime, date)
True

>>> datetime.combine(
...     date(2012, 9, 16), time(14, 55))
datetime.datetime(2012, 9, 16, 14, 55)
```


dates and datetimes

```
>>> ny = timezone("America/New_York")
>>> in_ny = ny.localize(
...     datetime(2012, 9, 16, 1, 30))
```

```
>>> la = timezone("America/Los_Angeles")
>>> in_la = la.localize(
...     datetime(2012, 9, 15, 22, 30))
```

```
>>> in_ny == in_la
```

```
True
```

```
>>> in_ny.date() == in_la.date()
```

```
False
```

dates and datetimes

- dates are always naive
- depending on how you use them, this might be a problem
- using an aware datetime as a date is an accident waiting to happen
- Django supports mixing naive datetimes and dates (historically)

new in Django 1.4
time zone support

Django < 1.4
or USE_TZ = False

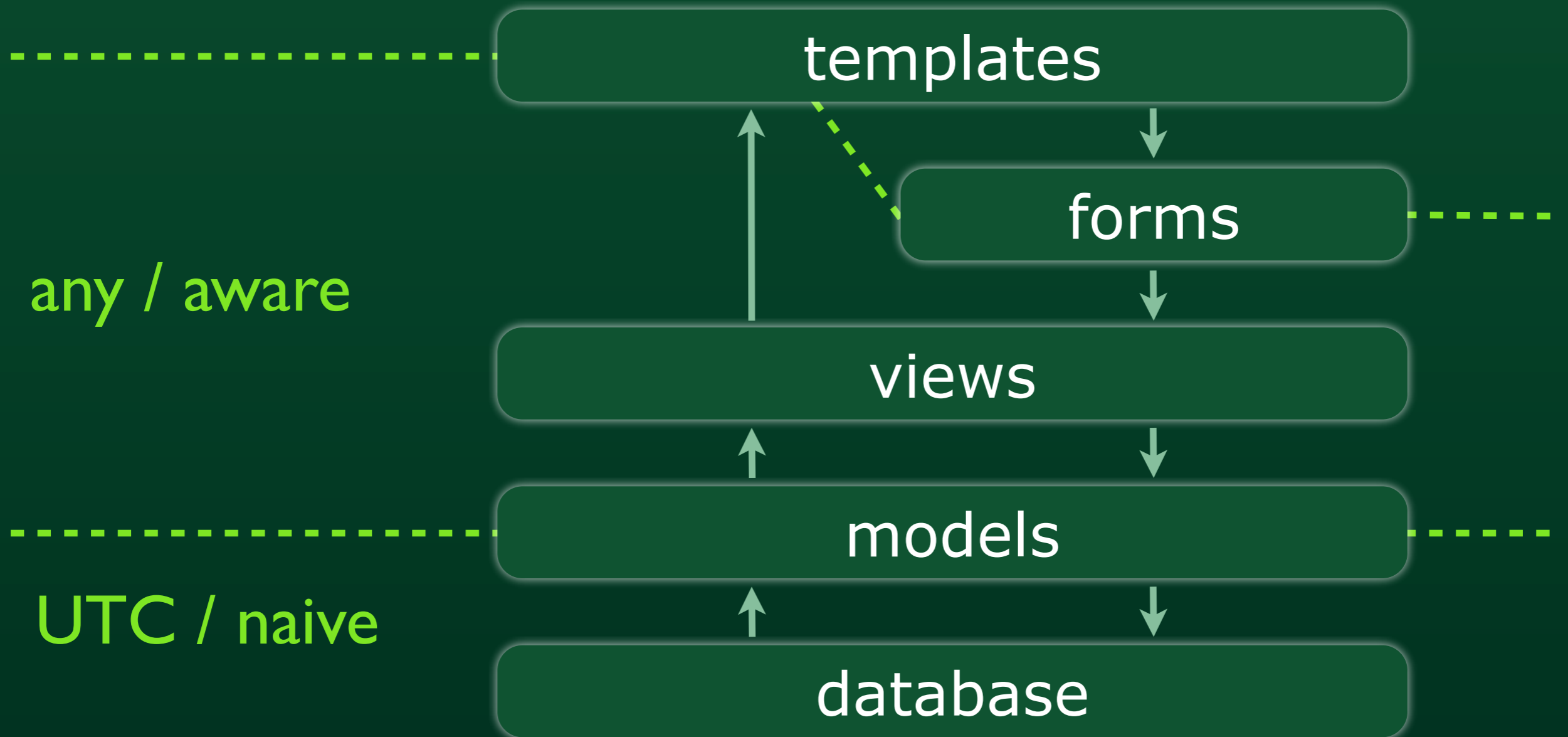
- uses naive datetimes in local time everywhere

Django \geq 1.4 and USE_TZ = True

- uses **aware datetimes in UTC** internally
- stores **naive datetimes in UTC** in the database (except PostgreSQL)
- converts to **aware datetimes in local time** in forms and templates
- supports **multiple time zones!**

conversions at the borders

local / aware



Time zone support in Django

localhost:8000/admin/tz_app/name/1/

Welcome, admin.

Home > Tz_app > Names > new name

Set your time zone and locale

Default time zone: America/Chicago

Current time zone: Europe/Paris Set

Alternative time zone: America/Chicago Set

Locale: English (en) Set

Change name History

Please correct the error below.

Name: new name

2012-10-28 02:30:00 couldn't be interpreted in time zone Europe/Paris; it may be ambiguous or it may not exist.

Dated: Date: 2012-10-28 Today |

Time: 02:30:00 Now |

Delete Save and add another Save and continue editing Save

Delete Save and add another Save and continue editing Save

default and current time zones

- `default = settings.TIME_ZONE`
 - used in models for conversions between naive and aware objects
- `current = end user's time zone`
 - used in templates and forms
 - for multiple time zones support

auto-conversions

- ensure backwards compatibility
- avoid surprises for single time zone sites
- but support sloppy constructs, e.g.
 - filter a DateTimeField with a date
 - save a datetime in a DateField

utilities

```
>>> from django.conf import settings
>>> from django.utils import timezone

>>> settings.USE_TZ = True
>>> timezone.now()
datetime.datetime(2012, 9, 16, 12, 58,
39, 438148, tzinfo=<UTC>)
>>> settings.USE_TZ = False
>>> timezone.now()
datetime.datetime(2012, 9, 16, 14, 58,
40, 746120)
```

limitations

- the database works in UTC
<https://code.djangoproject.com/ticket/17260>
- `QuerySet.dates()`
- `__year/month/day/week_day`
- authors of pluggable apps may have to handle two cases

key learnings

1. a **datetime** is a point in time
a **date** is a calendaring concept
2. use **aware datetimes in UTC** and
convert to **local time for humans**
3. learn how to use **pytz properly**
especially **localize** and **normalize**

Time isn't as simple as it seems.
Learn and practice!

Questions

Further reading

<http://docs.python.org/library/datetime>

<http://pytz.sourceforge.net/>

<https://docs.djangoproject.com/en/dev/topics/i18n/timezones/>

<https://bitbucket.org/augustin/django-tz-demo>

http://groups.google.com/group/django-developers/browse_thread/thread/cf0423bbb85b1bbf

<https://code.djangoproject.com/ticket/2626>