Verloren in der Zeit?

Ein Wegweiser durch Datums- und Zeitroutinen in Python

Dietmar Thaler
Grazer Linuxtage
29. April 2017

glt.foehnwall.at/glt17.html

Überblick

- Zeit und Zeitzonen
- Datum und Kalender
- Python
  - datetime
  - pytz
  - dateutil
  - arrow
- Empfehlungen

Z E I T

Zeit

- Was ist Zeit?
  - Zeit ist eine physikalische Grundgröße
  - Zeit wird mit Uhren gemessen
  - Uhren sind Zeitmesser
- Isaac Newton (1642/1643-1727): „absolute Zeit“
- 2. Hauptsatz der Thermodynamik: Zeit nimmt zu
- Albert Einstein: relativistische Zeit – Eigenzeit („proper time“)

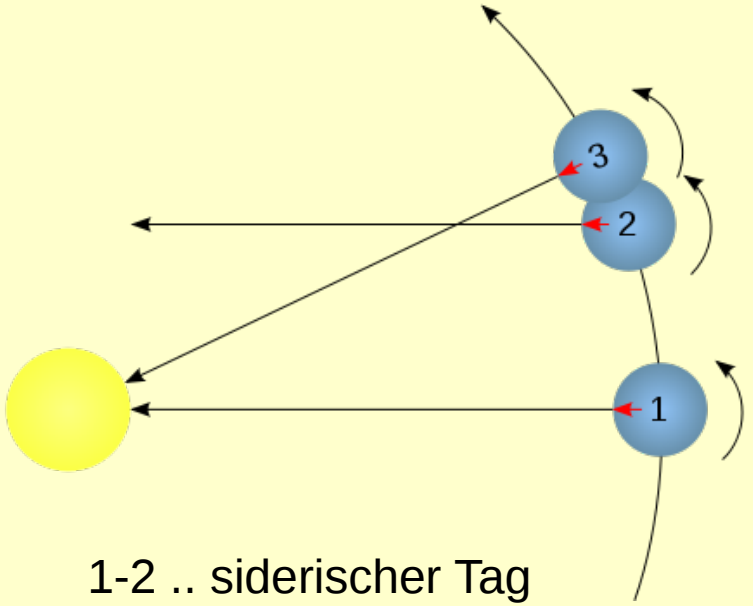
Die Atomzeit

- Ab 1967: Atomsekunde:
  Das 9.192.631.770-Fache der Periodendauer der dem Übergang zwischen den beiden Hyperfeinstrukturniveaus des Grundzustandes von Atomen des Nuklids 133Cs entsprechenden Strahlung
- Messung mit Atomuhren: Isidor Isaac Rabi (Nobelpreis für Physik 1944)
- Diese Sekunde ist die verbindliche Einheit der Zeitmessung
- Zählung seit 1. Januar 1958, 00 Uhr definiert die **TAI** (**Temp Atomique International**)

- 60 Sekunden = 1 Minute
- 1 Stunde = 60 Minuten
- 1 (bürgerlicher) Tag ~ 24 Stunden

Sonnenzeit bzw. Ortszeit

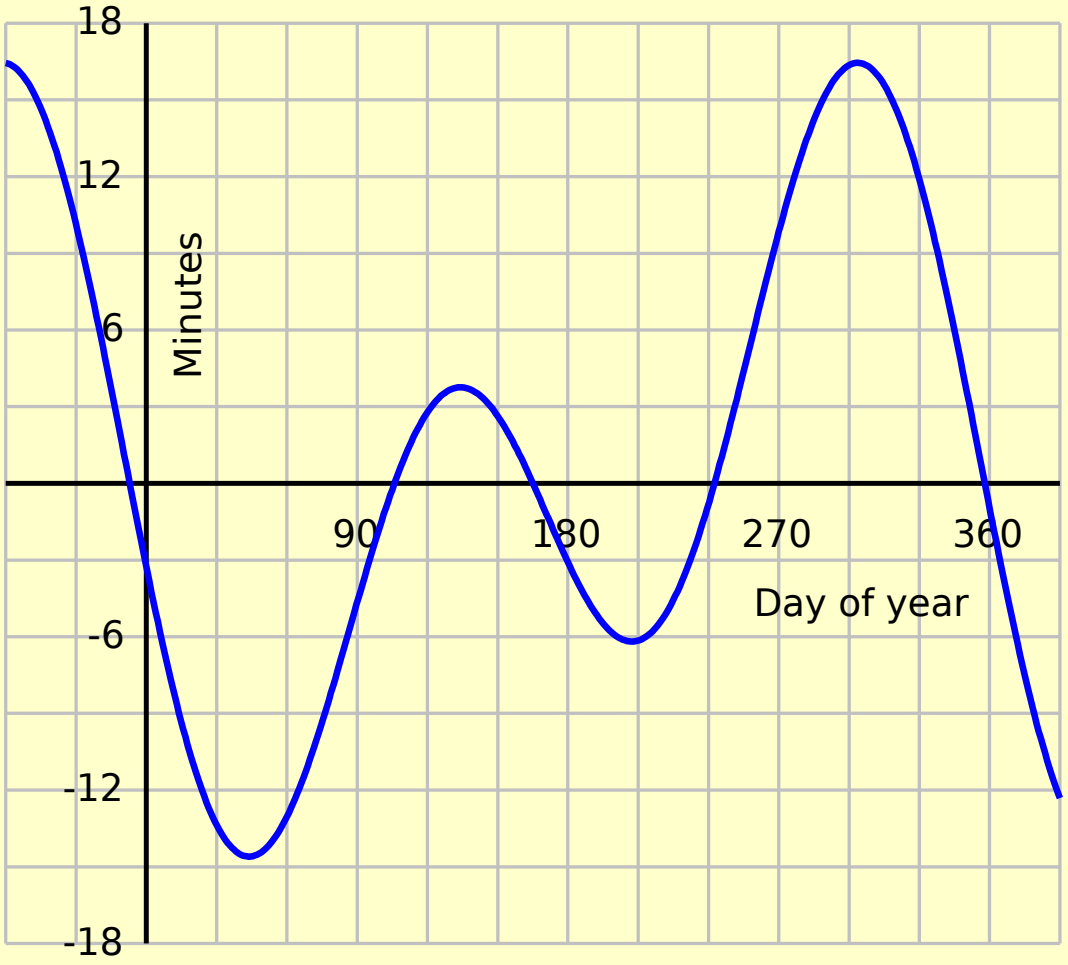
- Sonnenzeit: am (scheinbaren) Gang der Sonne orientiert
- 1 „Tag“: Zeit zwischen zwei Sonnenhöchsständen
- Unregelmäßig wegen Bahnbesonderheiten der Erde:
  Sonnentag ist übers Jahr ungleich lang



1-2 .. siderischer Tag
1-3 .. wahrer „Sonnen“-Tag

Mittlere Sonnenzeit bzw. Ortszeit

- Über das Jahr gemittelte Zeit von Sonnenhöchsstand zu Sonnenhöchststand
- **Gilt nur für einen Längenkreis**
- Unregelmäßigkeiten der Bahn ausgeglichen



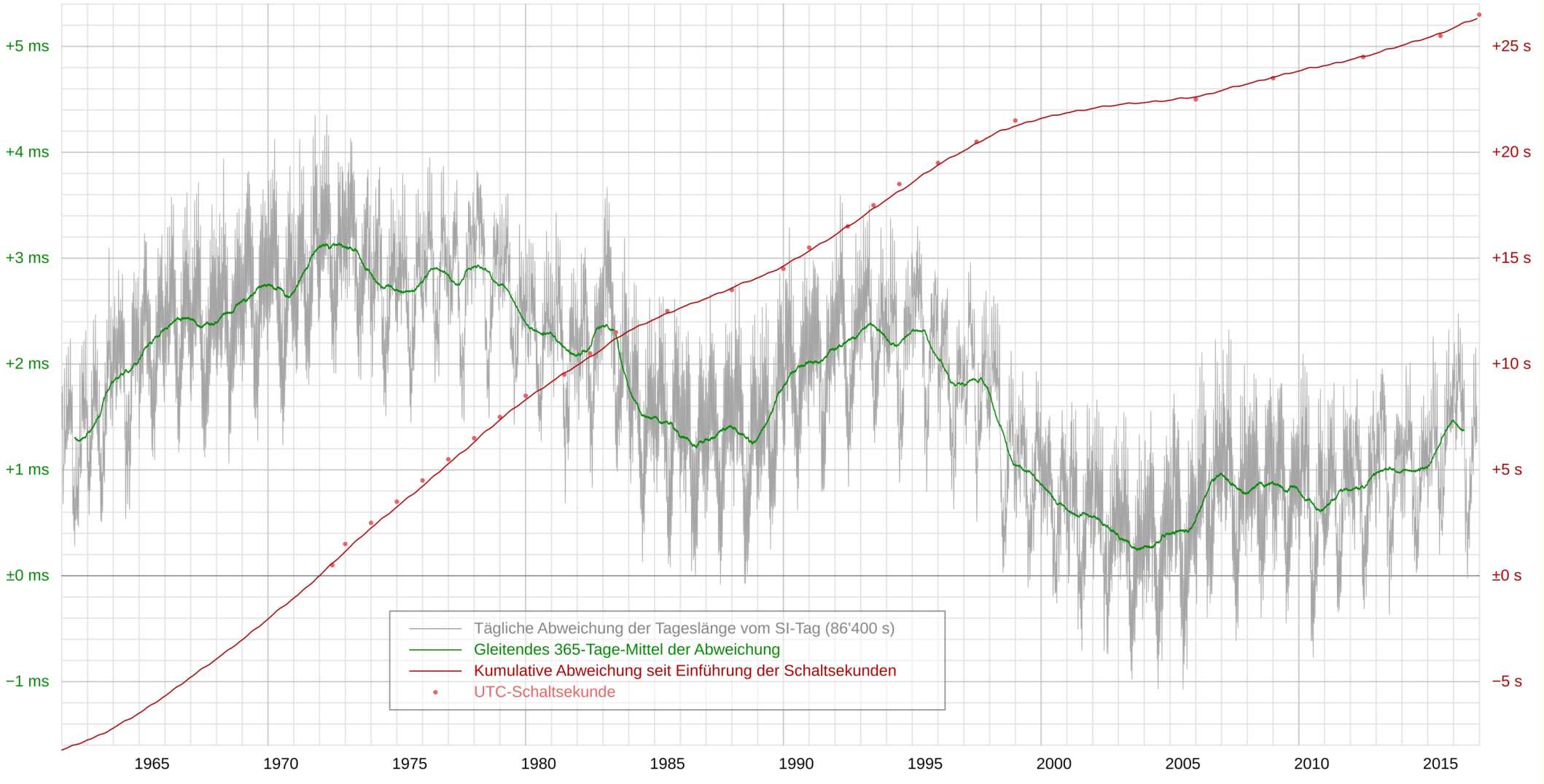
en.wikipedia.org/wiki/Equation_of_time

Universal Time UT1

Bis 1967: Sekunde astronomisch definiert

- Sonnensekunde: 1/86.400 des mittleren Sonnentages
- **UT1**: die mittlere Ortszeit auf 0 Grad Länge („Greenwich Meridian“): vormals GMT
  Eine universelle Zeitskala, berücksichtigt die Unregelmäßigkeiten der Erdrotation und der Erdbahn

Abweichung Tageslänge UT1 von Atomzeit TAI



Quelle: Wikipedia.de: Erdrotation - Abweichung der Tageslänge vom SI-Tag

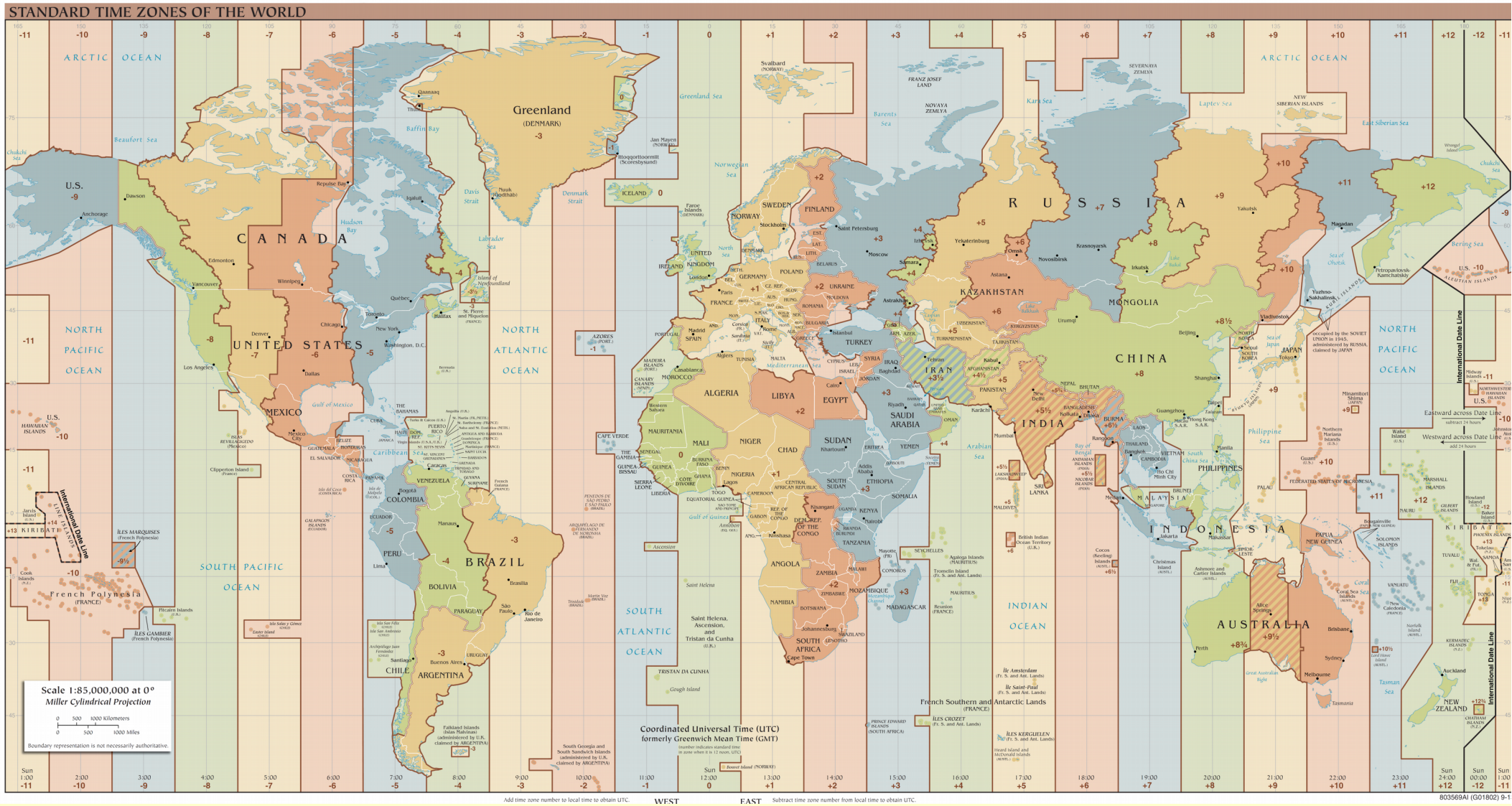
UTC - Weltzeit

- **Problem**: Die Atomzeit TAI läuft der astronomischen Universalzeit UT1 allmählich voraus:
  1.Jan. 1958 .. 1. Jan. 2017: 37 Sekunden
- Daher mit 1972 Einführung der **Coordinated Universal Time UTC**:
  Die UTC entspricht der UT1; zählt aber mit Atomsekunden;
- Synchronisierung der UTC mit der UT1 durch Einfügen von **Schaltsekunden** (letzte: 31. 12. 2016, 23:59:60!) notwendig

Zeitzonen

- **Ideal**: alle 15 Längengrade +/- 7,5 Grad eine Zeitzone mit jeweils 1 Stunde Zeitsprung:
  - 0 Grad: UTC
  - 15 Grad: MEZ (CET), etc ...
- **Praktisch**: sehr unregelmäßige Zeitzonen, kompliziert durch „Sommerzeiten“ bzw. „daylight saving times“
- Arthur David Olson: *tz database* alias *IANA time zone database* alias *zoneinfo database* alias *olson database* www.iana.org/time-zones

Zeitzonen



Quelle: upload.wikimedia.org/wikipedia/commons/8/88/World_Time_Zones_Map.png

DATUM
KALENDER

Kalender

Das Jahr:

- **Siderisches Jahr:** Sonne vor dem gleichen Fixsterhintergrund: in der Epoche J2000.0: 365,25636042 Tage
- **Anomalistisches Jahr**: Durchgang der Erde durch das Perihel (sonnennächster Punkt): in der Epoche J2000.0: 365,25963588 Tage
- **Tropisches Jahr:** Zeit zwischen zwei Durchgängen durch z.B. den Frühlingspunkt in der Epoche J2000.0: **365,242 190 52 Tage** (Abnahme um 0,5 s /100 Jahren)

Kalender

Julianischer Kalender seit 45 v.Chr.:

- Sonnenkalender von Julius Cäsar eingeführt
- Beseitigung beobachtbarer Unregelmäßigkeiten
- Das Jahr hat 365 Tage
- Jedes vierte Jahr (Schaltjahr) 366 Tage
- Entspricht: 365,25 Tage pro Jahr
- Gültig bis 1582, oft auch deutlich länger (Russland mit der Oktoberrevolution 1917)
- Frage des **Nullpunkts**: seit dem Frühmittelalter Christi Geburt (Achtung: **kein Jahr 0**!)

Kalender

Gregorianischer Kalender ab 1582
(Do, 4. Okt → Fr, 15. Oktober 1582)

- Probleme mit dem **Osterdatum**
- Von **Papst Gregor** eingeführt unter wesentlicher Beratung seinerzeitiger Astronomen

- Jahr 365 Tage
- Jedes vierte Jahr Schaltjahr mit 366 Tagen (2016 war ein Schaltjahr)
  - Ausnahme: Jahreszahl durch 100 teilbar (1900 war kein Schaltjahr)
  - Ausnahme der Ausnahme: Jahreszahl durch 400 teilbar (2000 war ein Schaltjahr)

ISO 8601

- Internationale Norm zur Darstellung von Zeit und Datum, z.B.:
- Datum und Zeit:
  YYYY-mm-ddTHH:MM:SS.f±zzzz
  2017-04-29T15:45:01.1222+0200
- Zeitperiode:
  YYYY-mm-ddTHH:MM/HH:MM
- .. und unzählige Variationen

Python

Modul **datetime**

Standardmodul: fix implementiert in jeder Python-Version (2.x, 3.x), objektorientiert, (proleptischer) Gregor. Kalender

- **datetime.date**: Datums-Klasse
- **datetime.time**: Zeit-Klasse (unabhängig vom Datum)
- **datetime.datetime: Kombination der beiden**
  - **„time zone naive“: ohne Zeitzoneninformation (default)**
  - **„time zone aware“: mit Zeitzoneninformation**
- **datetime.timedelta: Zeitdifferenzen für Berechnungen**
- **datetime.tzinfo (datetime.timezone): für Zeitzonen**

datetime.datetime Objekt

„naive“: ohne Zeitzoneninformation (default)

```
import datetime
print(datetime.datetime.now())
print(datetime.datetime.utcnow())
print(datetime.datetime(year=2017, month=3, day=12))
print(datetime.datetime(2017,4,29,15,15,30,123456))
```

```
2017-04-16 10:20:41.566196
2017-04-16 08:20:41.566236
2017-03-12 00:00:00
2017-04-29 15:15:30.123456
```

datetime.datetime

„naive“: ohne Zeitzoneninformation (default)

```
import datetime
d = datetime.datetime.utcnow()
print(d)
print(d.year, d.month)
d1 = d.replace(microsecond=0)
print(d.isoformat())
print(d1.isoformat())
print(d.date())
print(d.time())
```

```
2017-04-16 09:41:01.532461
2017 4
2017-04-16T09:41:01.532461
2017-04-16T09:41:01
2017-04-16
09:41:01.532461
```

datetime.timedelta

„naive“

```
import datetime
d = datetime.datetime(2017, 4, 29, 12)
dt = datetime.timedelta( days=0, seconds=0,
microseconds=0,milliseconds=0, minutes=60, hours=1,
weeks=0)
dd = [d + k*dt for k in range(5)] #list comprehension
for d in dd:
    print(d.isoformat())
```

```
2017-04-29T12:00:00
2017-04-29T14:00:00
2017-04-29T16:00:00
2017-04-29T18:00:00
2017-04-29T20:00:00
```

strftime/strptime

- **Formatierung** von date-, time- und datetime-Objekten für die **Ausgabe**
- **Eingabe** von Datums- und Zeit-Zeichenketten in date-, time- und datetime-Objekte
- Formatangaben werden als String übergeben, z.B.:

```
Dtform = '%Y-%m-%d %H:%M' →
2017-04-29 15:15
```

strftime/strptime
(einige Formatierungsanweisungen)

%Y	Jahreszahl (4-stell.)	2007
%y	Jahreszahl (2-stellig)	07
%m	Monat (2-stellig)	04
%b	Monat (abgekürzt, lokal)	Apr
%B	Monat (voll, lokal)	April
%d	Tag (2-stellig)	07
%H	Stunde (00..24)	15
%M	Minute (00..59)	09
%S	Sekunde (00..59)	17
%f	Mikrosekunde n	000703
%a	Wochentag (Abk., lokal)	Mo
%A	Wochentag (voll, lokal)	Monday, Montag, ...
….	….	….
Siehe z.B.: docs.python.org/3.6/library/datetime.html#strftime-and-strptime-behavior		

strftime() - Beispiel

```
import locale, datetime

now = datetime.datetime(2017,1,23,12,12,57)
form1 = '%Y-%m-%d %H:%M:%S'
print(now.strftime(form1))

form2 = '%d. %B %Y, %H:%M‘
locale.setlocale(locale.LC_ALL, 'de_AT.UTF8')
print(now.strftime(form2))

locale.setlocale(locale.LC_ALL, 'de_DE.UTF8')
print(now.strftime(form2))
```

```
2017-01-23 12:12:57
23. Jänner 2017, 12:12
23. Januar 2017, 12:12
```

strptime() - Beispiel

```
import datetime

form1 = '%Y-%m-%d %H:%M:%S'
datstr1 = '2016-12-31 16:33:12'
dat1 = datetime.datetime.strptime(datstr1, form1)
print(dat1)

form2 = '%d. %B %Y, %H:%M'
datstr2 = '31. December 2016, 16:33'
dat2 = datetime.datetime.strptime(datstr2, form2)
print(dat2)
```

```
2016-12-31 16:33:12
2016-12-31 16:33:00
```

datetime.tzinfo

- **datetime.tzinfo**: Abstrakte Basisklasse für Zeitzoneninformation:
  - Aus Sicht des Anwenders komplex und nicht der Mühe wert; Sinnvoller → **Modul pytz**
- Zur Not ab Python 3.2: **datetime.timezone** um aus *timezone naive* (*unaware*) Objekten *timezone aware* Objekte zu machen:

```
utc = datetime.timezone.utc
utc_aware = utc.unaware.replace(tzinfo=utc)
```

Modul pytz

*„**pytz** brings the **Olson tz database** into Python. This library allows accurate and cross platform timezone calculations using Python 2.4 or higher.“*

- Erste Wahl für die Berücksichtigung von Zeitzonen.
- **Nachteil:**
  in den Distros möglicherweise veraltet, z.B.:
  Ubuntu 16.04

```
In [1]: import pytz
In [2]: pytz.VERSION
Out[2]: '2014.10'
```

pytz - Winterzeit

```
import datetime
import pytz

form = "%Y-%m-%d %H:%M:%S %z (%Z)"
dt = datetime.datetime(2017, 3, 26, 0, 0)
utc = pytz.utc
tz = pytz.timezone('Europe/Vienna')
dt_utc = dt.replace(tzinfo=utc)
dt_tz = dt_utc.astimezone(tz)

print('dt naive    : ', dt.strftime(form))
print('dt utc      : ', dt_utc.strftime(form))
print('dt tz       : ', dt_tz.strftime(form))
```

```
dt naive    :  2017-03-26 00:00:00  ()
dt utc      :  2017-03-26 00:00:00 +0000 (UTC)
dt tz       :  2017-03-26 01:00:00 +0100 (CET)
```

pytz - Sommerzeit

```
import datetime
import pytz

form = "%Y-%m-%d %H:%M:%S %z (%Z)"
dt = datetime.datetime(2017, 3, 26, 1, 0)
utc = pytz.utc
tz = pytz.timezone('Europe/Vienna')
dt_utc = dt.replace(tzinfo=utc)
dt_tz = dt_utc.astimezone(tz)

print('dt naive    : ', dt.strftime(form))
print('dt utc      : ', dt_utc.strftime(form))
print('dt tz       : ', dt_tz.strftime(form))
```

```
dt naive    :  2017-03-26 01:00:00  ()
dt utc      :  2017-03-26 01:00:00 +0000 (UTC)
dt tz       :  2017-03-26 03:00:00 +0200 (CEST)
```

Modul dateutil

„The dateutil module provides powerful extensions to the standard datetime module, available in Python.“

- Ostern berechnen
- Zeit- und Datums-Strings „out of the box“ parsen
- Komplexe Datums-Wiederholung erzeugen
- Zeitzonen

dateutils.easter

```
import datetime, dateutil.easter as e

form = "%m-%d"
for y in range(2014, 2021):
    east_j = e.easter(y,1).strftime(form)
    east_o = e.easter(y, 2).strftime(form)
    east_g = e.easter(y,3).strftime(form)
    print('{:d} |  J:{}  O:{}  G:{}'.format(y, east_j,
                                             east_o,
                                             east_g))
```

```
2014 |  J:04-07  O:04-20  G:04-20
2015 |  J:03-30  O:04-12  G:04-05
2016 |  J:04-18  O:05-01  G:03-27
2017 |  J:04-03  O:04-16  G:04-16
2018 |  J:03-26  O:04-08  G:04-01
2019 |  J:04-15  O:04-28  G:04-21
2020 |  J:04-06  O:04-19  G:04-12
```

dateutils.parser

```
import datetime, dateutil.parser as parser
s1 = '12.4.2016 13:23'
print('String1          : ', s1)
dt1 = parser.parse(s1)  # Standard
print('Datetime1(US)    : ',dt1)
dt2 = parser.parse(s1, dayfirst=True)  # De
print('Datetime2(DE)    : ',dt2)
s3 = '2017-06-12 13:23 CET'  # ISO-nahe
print('String3          : ', s3)
dt3 = parser.parse(s3)
print('Datetime (~ISO) : ',dt3)
s4 = '2016-04-12T13:23Z'  # ISO
print('String4          : ', s4)
dt4 = parser.parse(s4)
print('Datetime3 (~ISO): ',dt4)
```

dateutils.parser

```
String1           :  12.4.2016 13:23
Datetime1(US)     :  2016-12-04 13:23:00
Datetime2(DE)     :  2016-04-12 13:23:00
String3           :  2017-06-12 13:23 CET
Datetime (~ISO)   :  2017-06-12 13:23:00+02:00
String4           :  2016-04-12T13:23Z
Datetime3 (~ISO):  2016-04-12 13:23:00+00:00
```

dateutil.rrule

The rrule module offers a small, complete, and very fast, implementation of the recurrence rules documented in the iCalendar RFC, including support for caching of results.

dateutil.readthedocs.io/en/stable/rrule.html

dateutil.rrule()

```
import datetime
from dateutil.rrule import rrule,MONTHLY

form = "%Y-%m-%d, %A"
start_date = datetime.datetime(2017, 1, 31)
result = list(rrule(freq=MONTHLY, count=4,
                    dtstart=start_date,
                    bymonthday = -1))
for d in result:
    print('Monatsletzter: ', d.strftime(form))
```

```
Monatsletzter:  2017-01-31, Tuesday
Monatsletzter:  2017-02-28, Tuesday
Monatsletzter:  2017-03-31, Friday
Monatsletzter:  2017-04-30, Sunday
```

Modul arrow

„*better dates and times for Python*“¶

„Arrow is a Python library that offers a sensible, human-friendly approach to creating, manipulating, formatting and converting dates, times, and timestamps.“

- Ein Modul für datetime- und pytz-Funktionalität
- *„Time zone aware by default“*
- Einfachere Lesbarkeit von ISO8601-Formate
- Etwas langsamer als datetime
- Nicht datetime-Zuweisungs-kompatibel (aber Methoden verfügbar)

arrow – time zone aware

```
import arrow
form = "%Y-%m-%d %H:%M %z (%Z)"


d_utc = arrow.utcnow()
print(d_utc.strftime(form))
d_tz = d_utc.to('CET')
print(d_tz.strftime(form))
d_tz = d_utc.to('Europe/Moscow')
print(d_tz.strftime(form))
```

```
2017-04-21 09:50 +0000 (UTC)
2017-04-21 11:50 +0200 (CEST)
2017-04-21 12:50 +0300 (MSK)
```

arrow – „timedeltas“

```
import arrow

form = "%Y-%m-%d %H:%M %Z"
d_utc = arrow.get('2017-03-25T23:00')
for h in range(4):
    d = d_utc.replace(hours=h)
    d_tz = d.to('CET')
    print(d.strftime(form), '-→',
          d_tz.strftime(form))
```

```
2017-03-25 23:00 UTC --> 2017-03-26 00:00 CET
2017-03-26 00:00 UTC --> 2017-03-26 01:00 CET
2017-03-26 01:00 UTC --> 2017-03-26 03:00 CEST
2017-03-26 02:00 UTC --> 2017-03-26 04:00 CEST
```

Stief- und Waisenkinder

Nichtbehandelte Bibliotheken und Module

- Standardbibliothek:
  - datetime.date, datetime.time
  - time, calendar
- Fremdbibliotheken: `via pip install ...`
  - pendulum: pendulum.eustace.io
  - delorian: delorean.readthedocs.io
  - updatetime: udatetime-fast-rfc3339-date-time-python-library

Stief- und Waisenkinder
Nichtbehandelte Bibliotheken und Module

- Dazu noch die date-time Funktionalität diverser umfassender Bibliotheken
  - numpy
    - datetime arrays
  - matplotlib
    - eigenen Datums-Typ für Plots
  - pyephem
    - hochgenaues (Julian-)Date-Objekt mit Konversions-Fumktionen von/zu datetime
  - …..

Empfehlungen
Regel 1

Keine Ein-Ausgabe unterschiedlicher Zeitzonen erforderlich:

Konvention:

- Verwende Modul **datetime** „*timezone naive*“
- Benütze nur **UTC**-Zeiten
- **Kommuniziere** die Konvention

Achtung: Zeitzone nirgends ablesbar!

Empfehlungen
Regel 2

Ein-Ausgabe unterschiedlicher Zeitzonen:

- Verwende Modul **datetime** und **pytz** (Version!) bzw. **arrow**, bei Bedarf **dateutil**
- Wandle jede Eingabezeit Zeit sofort um in: **UTC - time zone aware**
- Alle **Kalkulationen** in UTC
- Wandle **Ausgabe** in die gewünschte **Zeitzone** um
- Falls möglich: verwende **ISO-8601-Format**(e)

Empfehlungen
Regel 3

Falls zweckmäßig,

missachte Regel 1 und 2

Literatur

- **Allgemein:**
  - **Urban**, S.E; **Seidelmann** P.K.: Explanatory Supplement to the Astronomical Almanac, Third Edition, University Science Books; 2012
  - P. **Tippler**: Physik, Spektrum Akademischer Verlag, 1994
  - Wikipedia (meist für Abbildungen):
    - de.wikipedia.org/wiki/Universal_Time
    - de.wikipedia.org/wiki/Zeitgleichung
    - de.wikipedia.org/wiki/Atomzeit
    - en.wikipedia.org/wiki/Time_zone
    - en.wikipedia.org/wiki/Time_zone
  - Timezone-DB: www.iana.org/time-zones
  - Korpela: Info on ISO 8601

Literatur

Python

- docs.python.org/3/library/datetime.html
- pythonhosted.org/pytz/
- dateutil.readthedocs.io/en/stable/
- arrow.readthedocs.io/en/latest/

Ergänzendes

- Maggiolo: how-much-is-time-wrong-around-the-world
- Ronnacher, A.: Eppur si muove
- Danjou, J.: Timezones and Python

E N D E

Dietmar Thaler
Grazer Linuxtage
29. April 2017

glt.foehnwall.at/glt17.html