What can be said in 30min about Python?

Center for Mathematical Sciences, Lund University

eScience Program Language Day, April 2023

Outline

- some personal remarks about Python
- Language "goodies" (a selection)
- NumPy/ Pandas
- Python wrapping C
- Python and Linux
- MicroPython

Some Personal Remarks

My first program ... Freiburger Code (Zuse, 1968),

ZRA Uni Marburg: Zuse Z22 1963 - 1969

my first academic program ALGOL60 (1972), then

- FORTRAN60 (1976), FORTRAN77 - the "end do/if " (1984), FORTRAN90 (ca 2000)
- MATLAB (1978)
- Python (2006), Python Courses in Lund since ca 2008, 2 course books about Scientific Python

Python goodies

1. Readability becomes syntax - the indent

```
for i in range(200):
    if i%2==0:
        print(f'{i} is an even number')
```

The end of the end statement

2. List comprehension - readable for/if statements

```
L=['apple', 'lemon', 'grape', 'apple', 'banana']

applePlaces=\
[i for i, fruit in enumerate(L) if fruit=='apple']

print(f'There are {len(applePlaces)} places with
                                apples.')
```

Python goodies

3. The use of `or`

```
def func(f, fprime=None):
    fp = fprime or lambda x: (f(x+1.e-4)-f(x))/h
    ...
```

Python goodies

4. Infinite loops (generators)

```
def odd_numbers():
    "generator for odd numbers"
    k=0
    while True:
        k+=1
        if k % 2 == 1:
            yield k

# Find the first odd_number greater than 68:
on=odd_numbers()

for nu in on:
    if nu > 68:
        print(nu)
        break
```

NumPy/ Pandas

NumPy's `ndarray` is the datatype primarily used for numeric arrays (homogeneous type entries)

Pandas' `dataframe` is the dataype for mixed type arrays

both are well suited for column/ rowwise mathematical/ startistical evaluations and graphs.

Pandas - A Guiding Example

Solar cells in Södra Sandby

solarWatts.dat

```
Date;Watt
            :
2019-10-20 08:22:00 ; 44.0
2019-10-20 08:23:00 ; 61.0
2019-10-20 08:24:00 ; 42.0
            :
```

price.dat

```
Date;SEK
            :
2019-10-20 01:00 ; 0.32
2019-10-20 02:00 ; 0.28
2019-10-20 03:00 ; 0.29
            :
```

rates.dat

```
Date;Euro_SEK
        :
2019-10-21 ; 10.7311
2019-10-22 ; 10.7303
2019-10-23 ; 10.7385
        :
```

Typical questions?

- When was the highest power production?
- Which day delivered the most energy?
- Which was the economical outcome in Euro per month?
- Was there a day where the solar cells were out of work?

Pandas Dataframe versus Numpy Array

Let's start by just looking at an example

```
A=array(
      [[1.,2.,3.],
       [4.,5.,6.]])
print(A)
```

results in

```
[[1. 2. 3.]
 [4. 5. 6.]]
```

```
import pandas as pd
AF = pd.DataFrame(A)
print(AF)
```

results in

```
     0    1    2
0  1.0  2.0  3.0
1  4.0  5.0  6.0
```

We see that a Pandas dataframe has extra labels of the rows and columns - called *index* and *columns*. These are metadata of a dataframe.
Here they coincide with NumPy's indexing.

Dataframe: Index and Column Metadata

Index and Column Metadata gives a Pandas dataframe the possibility to label the data as known from classical tabel design

```
AF.columns = ['C1','C2','C3']
AF.index   = ['R1', 'R2']
```

This gives

```
     C1   C2   C3
R1  1.0  2.0  3.0
R2  4.0  5.0  6.0
```

... and allows indexing to construct subframes:

```
AF.loc[['R1'],['C2']]
```

returns

```
     C2
R1  2.0
```

Datetime objects as indexes (Cont.)

We often want to make simple arithmetic with date and time information for this end the list of strings has to be converted in a list of so-called `datetime` objects first

```
date_index=pd.to_datetime(
           ['2020-01-19 11:45', '2020-01-20 02:14']
           )
AF.index=date_index
```

This way we can easlily find out how much time passed between the measurements `AF.index[1] - AF.index[0]`. The result is a `timedelta` object displayed here as

```
Timedelta('0 days 14:29:00')
```

Dataframe from a datafile

```
solarWatts = pd.read_csv("solarWatts.dat",
                         sep=';',
                         index_col='Date',
                         parse_dates=[0],
                         infer_datetime_format=True)
```

What do these arguments tell? The data is semicolon seperated, which column defines the index, which column contains dates, should dates converted from string to timestamp and how.

A typical entry: `solarWatts.iloc[100]`
The last date of the measurements: `solarWatts.iloc[-1]`

Merging Dataframes

Here we merge production data with electricity prices:

```
# Creating dataframes from csv-files
solarWatts = pd.read_csv("solarWatts.dat", ... )
prices     = pd.read_csv("price.dat", ...)
rates      = pd.read_csv("rates.dat", ...)
```

and

```
# Merging dataframes
solar_all=pd.merge(solarWatts, prices, how='outer',
                                sort=True, on='Date')
solar_all=pd.merge(solar_all, rates, how='outer', sort
                                =True, on='Date')
```

'outer' tells that the union of the index entries is taken as a new index.

Missing data

In the example:
power is updated every minute, price is updated every hour, rate is updated every day

This gives enties like:

Date	Watt	SEK	Euro_SEK
2019-10-21 13:59:00	2650.0	NaN	NaN
2019-10-21 14:00:00	1901.0	0.48725	NaN
2019-10-21 14:01:00	1436.0	NaN	NaN
2019-10-21 14:02:00	1040.0	NaN	NaN
2019-10-21 14:03:00	714.0	NaN	NaN
2019-10-21 14:04:00	498.0	NaN	NaN

What can be done?
Use frame methods like: `dropna`, `fillna`, `interpolate`

Missing data Cont.

We use here `fillna`:

```
solar_all=solar_all.fillna(method='pad',axis=0)
```

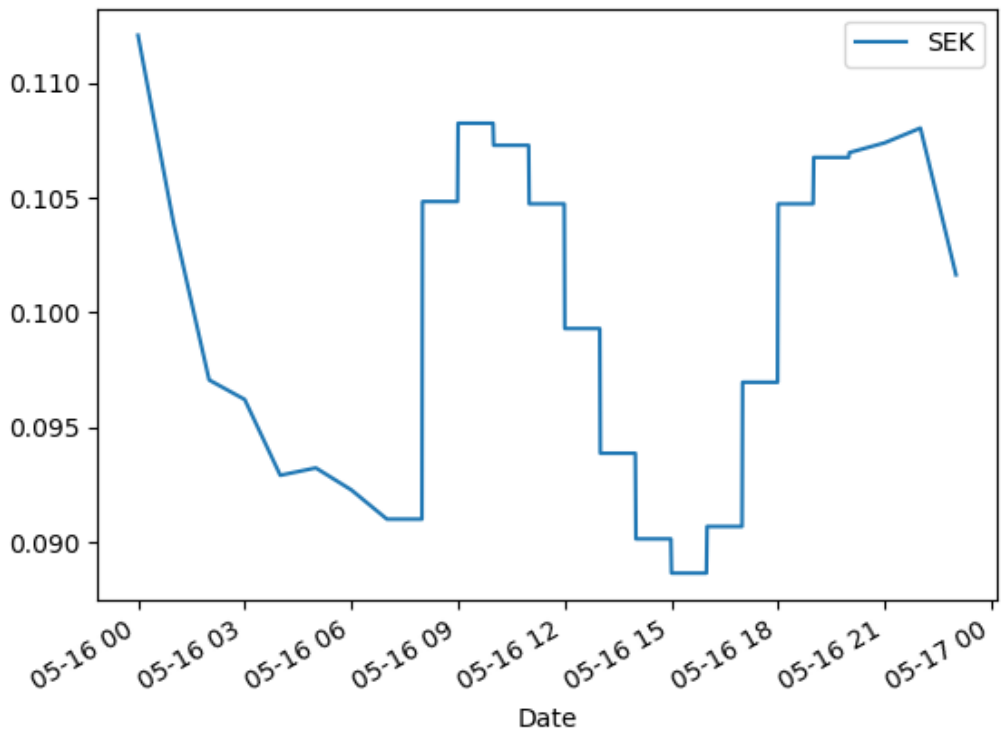
Note, this method is no *inplace* operation, i.e. a new object is created.

		Watt	SEK	Euro_SEK
Date				
2019-10-21	13:59:00	2650.0	0.47572	10.7311
2019-10-21	14:00:00	1901.0	0.48725	10.7311
2019-10-21	14:01:00	1436.0	0.48725	10.7311
2019-10-21	14:02:00	1040.0	0.48725	10.7311
2019-10-21	14:03:00	714.0	0.48725	10.7311
2019-10-21	14:04:00	498.0	0.48725	10.7311

Plotting dataframes

We demonstrate this by plotting the price variation along one day.

```
solar_all.loc['2020-05-16'].plot(y='SEK')
```



ASSIMULO: Python wrapping C with Cython

Background:
Professional package to solve ordinary differential equations
SUNDIALS (Lawrence Livermore Lab) written in C

Teaching Simulation Tools (very, very few students know C):

Wrapping: Python (solver control, plot) –> C (SUNDIALS) –>
Python (mechanical model to simulate)

Cython Wrappers - Ingredients

A c-file (examples.c):

```
#include <stdio.h>
#include "examples.h"
void hello(const char *name) {
    printf("hello %s\n", name);
}
```

A header file (examples.h):

```
#ifndef EXAMPLES_H
#define EXAMPLES_H
void hello(const char *name);
#endif
```

A wrapper file (example.pyx):

```
cdef extern from "examples.h":
    void hello(const char *name)

def py_hello(name):
    hello(name)
```

... and a setup file and a makefile etc,

Cython Wrappers - Usage

```
import pyexamples

pyexamples.py_hello(b"world")
```

Python and Linux

1. A (pure) Linux pipe example to find out, if I am using my computer at home

```
ifconfig|grep -A1 wlp0s20f3 | grep 192.168 | wc -l
```

2. A Python script (atHome.py) that interprets a string to see if my home IP occurs:

```
#!/usr/bin/env python3

import sys

intext = sys.stdin.read()
status = '' if '192.168' in intext else 'not'

print(f"I am {status} at home")
```

and its use in a Linux shell

```
ifconfig | ./atHome.py
```

Micropython

Python to flash code on microprocessors
Usefull for

- Monitoring experiments (accelerations, vibrations, temperature, ultra sound, light)
- small web interfaces
- MQTT messages

Serial ports, timers, deep sleep modes, GPIO's can directly be accessed from Python

Micropython

```
import dht
import machine

d = dht.DHT11(machine.Pin(4))
d.measure()
d.temperature()
d.humidity()
```

see also https://www.instructables.com/Getting-Started-With-Python-for-ESP8266-ESP32/

Last Slide