

Programming with PyFoam

Automating the boring stuff

Bernhard F.W. Gschaider

HFD Research GesmbH

Virginia Tech, USA

24. June 2020

Outline I

1 Introduction

- This presentation
- Who is this?
- What are we working with
- Before we start

2 Python Ecosystem

- Python versions
- Packages
- Virtual environments

3 The workhorses

- Library structure
- File handling by PyFoam
- Data structures in Python and OpenFOAM
- Parsing and writing OpenFOAM dictionaries

Outline II

- 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data

- 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class

- 6 Other
 - Paraview
 - Database

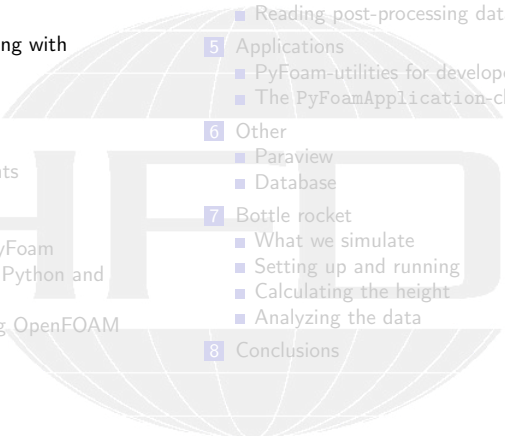
- 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data

Outline III

8 Conclusions



Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Outline

1 Introduction

■ This presentation

- Who is this?
- What are we working with
- Before we start

2 Python Ecosystem

- Python versions
- Packages
- Virtual environments

3 The workhorses

- Library structure
- File handling by PyFoam
- Data structures in Python and OpenFOAM
- Parsing and writing OpenFOAM dictionaries

4 Reading data

- Recycling PyFoam data
- Reading post-processing data

5 Applications

- PyFoam-utilities for developers
- The PyFoamApplication-class

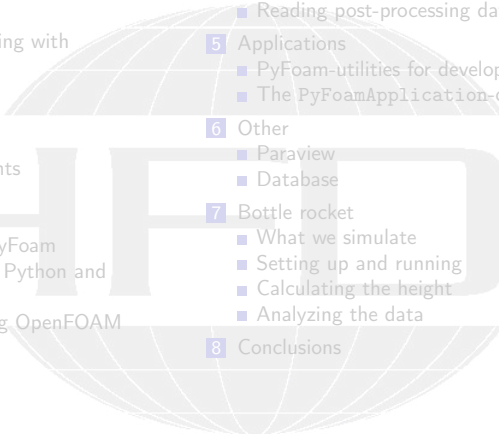
6 Other

- Paraview
- Database

7 Bottle rocket

- What we simulate
- Setting up and running
- Calculating the height
- Analyzing the data

8 Conclusions



The topic

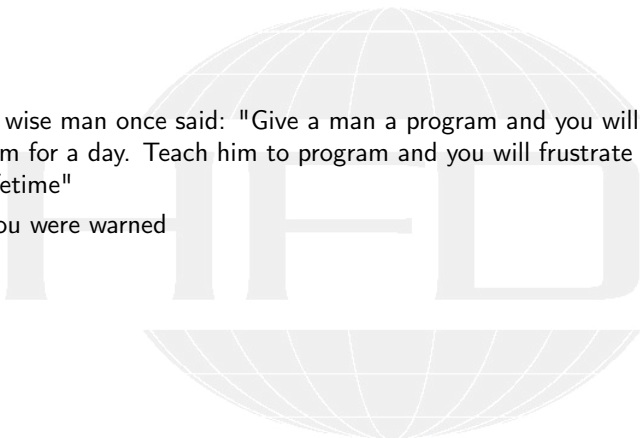
- Most PyFoam-users only use the utilities
 - That is fine
- But more flexibility can be had when programming your own workflows
 - Combining
 - 1 PyFoam
 - 2 with the strengths of the "python scientific computing ecosystem"
- This presentation gives an introduction to the most important concepts of the PyFoam library

Intended audience

- OpenFOAM users
 - especially those who are stuck with large repetitive tasks
- who know a little bit of Python programming
- who think it is a good deal
 - to invest some hours of programming time
 - and in return don't do more hours of repetitive work
- Also: when you program something you can put it under version control
 - then it is documented
 - and it is reproducible
 - and reviewable

On the other hand

- A wise man once said: "Give a man a program and you will frustrate him for a day. Teach him to program and you will frustrate him for a lifetime"
- You were warned



Format of the presentation

- There are two parts
 - 1 **Theory** : an introduction to the PyFoam-library with short examples
 - 2 An example that bind all this information together
 - simulation of something you can build yourself
 - we'll try to optimize it
 - **Caution** we might get wet

Outline

1 Introduction

- This presentation
- **Who is this?**
- What are we working with
- Before we start

2 Python Ecosystem

- Python versions
- Packages
- Virtual environments

3 The workhorses

- Library structure
- File handling by PyFoam
- Data structures in Python and OpenFOAM
- Parsing and writing OpenFOAM dictionaries

4 Reading data

- Recycling PyFoam data
- Reading post-processing data

5 Applications

- PyFoam-utilities for developers
- The PyFoamApplication-class

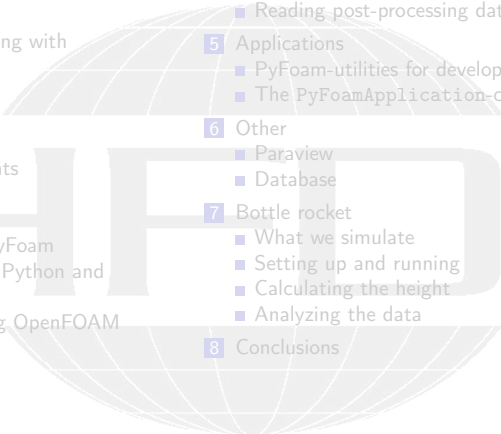
6 Other

- Paraview
- Database

7 Bottle rocket

- What we simulate
- Setting up and running
- Calculating the height
- Analyzing the data

8 Conclusions



Bernhard Gschaider

- Working with OPENFOAM™ since it was released
 - Still have to look up things in Doxygen
- I am **not** a core developer
 - But I don't consider myself to be an *Enthusiast*
- My involvement in the OPENFOAM™-community
 - Janitor of the `openfoamwiki.net`
 - Author of two additions for OPENFOAM™
 - `swak4foam` Toolbox to avoid the need for C++-programming
 - `PyFoam` Python-library to manipulate OPENFOAM™ cases and assist in executing them
 - `ansibleFoamInstallation` "Universal build script for OpenFOAM"
 - Organizing committee for the OPENFOAM™ *Workshop*
- The community-activities are not my main work but *collateral damage* from my real work at ...

Heinemann Fluid Dynamics Research GmbH

The company



- Subsidiary company of *Heinemann Oil*
 - Reservoir Engineering
 - Reservoir management

Description

- Located in Leoben and Vienna, Austria
- Works on
 - Fluid simulations
 - OPENFOAM™ and Closed Source
 - Software development for CFD
 - mainly OPENFOAM™
- Industries we worked for
 - Automotive
 - Processing
 - ...

Outline

1 Introduction

- This presentation
- Who is this?
- **What are we working with**
- Before we start

2 Python Ecosystem

- Python versions
- Packages
- Virtual environments

3 The workhorses

- Library structure
- File handling by PyFoam
- Data structures in Python and OpenFOAM
- Parsing and writing OpenFOAM dictionaries

4 Reading data

- Recycling PyFoam data
- Reading post-processing data

5 Applications

- PyFoam-utilities for developers
- The PyFoamApplication-class

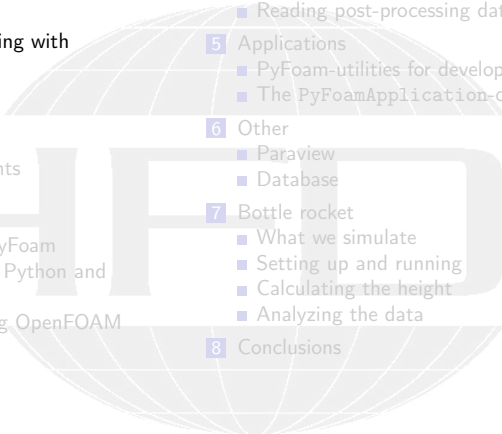
6 Other

- Paraview
- Database

7 Bottle rocket

- What we simulate
- Setting up and running
- Calculating the height
- Analyzing the data

8 Conclusions



What is PyFoam

- PyFoam is a library for
 - Manipulating OpenFOAM-cases
 - Controlling OpenFOAM-runs
- It is written in Python
- Based upon that library there is a number of utilities
 - For case manipulation
 - Running simulations
 - Looking at the results
- All utilities start with pyFoam (so TAB-completion gives you an overview)
 - Each utility has an online help that is shown when using the --help-option
 - Additional information can be found
 - on <https://openfoamwiki.net/index.php/Contrib/PyFoam>
- An introduction to this (and swak4Foam) was held yesterday

What is swak4Foam

From <https://openfoamwiki.net/index.php/Contrib/swak4Foam>

swak4Foam stands for SWiss Army Knife for Foam. Like that knife it rarely is the best tool for any given task, but sometimes it is more convenient to get it out of your pocket than going to the tool-shed to get the chain-saw.

- It is the result of the merge of
 - funkySetFields
 - groovyBC
 - simpleFunctionObjectsand has grown since
- The goal of swak4Foam is to make the use of C++ unnecessary
 - Even for complex boundary conditions etc
- We'll use here. But there will be no deeper explanation

Outline

1 Introduction

- This presentation
- Who is this?
- What are we working with
- **Before we start**

2 Python Ecosystem

- Python versions
- Packages
- Virtual environments

3 The workhorses

- Library structure
- File handling by PyFoam
- Data structures in Python and OpenFOAM
- Parsing and writing OpenFOAM dictionaries

4 Reading data

- Recycling PyFoam data
- Reading post-processing data

5 Applications

- PyFoam-utilities for developers
- The PyFoamApplication-class

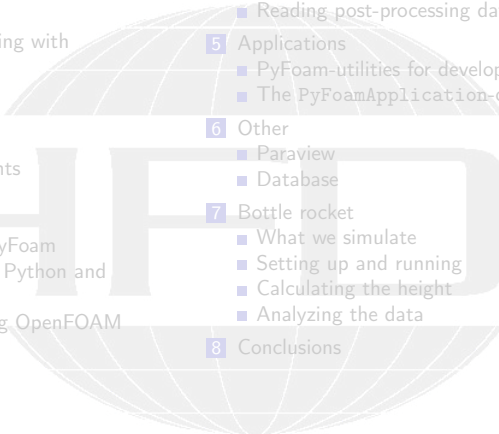
6 Other

- Paraview
- Database

7 Bottle rocket

- What we simulate
- Setting up and running
- Calculating the height
- Analyzing the data

8 Conclusions



Command line examples

- In the following presentation we will enter things on the command line. Short examples will be a single line (without output but a ">" to indicate *input*)

> ls \$HOME

- Long examples will be a grey/white box
 - Input will be prefixed with a > and blue
 - Long lines will be broken up
 - A pair of <brk> and <cont> indicates that this is still the same line in the input/output
 - «snip» in the middle means: "There is more. But it is boring"

Long example

```
> this is an example for a very long command line that does not fit onto one line of the slide <brk>
  <cont>but we have to write it anyway
first line of output (short)
Second line of output which is too long for this slide but we got to read it in all its glory and<brk>
  <cont> will be probably broken
```

Work environment

- You will use two programs
 - A terminal
 - with python
 - A text-editor
- For the text-editor you have the choice (these should be installed):
 - Emacs (king of text-editors)
 - VI
 - Kate with KDE
 - Gedit with Gnome
 - nano
 - jedit
 - geany (this is pre-installed on the Docker images)
 - ...

Getting onto the same page

- We need a machine with
 - OpenFOAM v1912
 - but older versions work as well
 - and other forks like foam-extend or Foundation v7 (but with that the re-implementation of lumpedWall would be pointless)
 - swak4foam
 - PyFoam
 - Text editors: emacs, vim, gedit

Open a shell and set us up for work

Assuming that you have a machine with those things installed

```
> mkdir pyFoamProgramming
> cd pyFoamProgramming
> . ~/OpenFOAM/OpenFOAM-v1912/etc/bashrc
```

Docker image with pre-installed PyFoam and swak4Foam

- Docker is a technology to run pre-packed containers based on Linux
 - Can be run on Linux, Windows and Mac OS X
 - Saves the work of installing requirements and compiling software
 - Only docker is needed (see <https://www.docker.com/>)
 - Image downloads may be rather big
- There is an image prepared for this training
 - Found at https://hub.docker.com/r/bgschaid/openfoam_by_ansible
 - Based on Ubuntu 18.04 LTS
 - OpenFOAM ESI v1912
 - Most recent release (2020.05) of PyFoam
 - Most recent release (2020.06) of swak4Foam
 - has **no** ParaView. Sorry
- The image was prepared with <https://openfoamwiki.net/index.php/Installation/Ansible>

Pulling the Docker-Image

Problems here:

- The image is over 3.6 Gig.
 - Depending on your network this might take some time
- You have to have docker installed on your machine

Pulling the

This will download the container the first time around

```
> docker pull bgschaid/openfoam_by_ansiible:training_programming_pyfoam_ofv15
```

Getting the script

```
> wget https://bit.ly/ofv15docker -O runFoamContainer.sh  
> chmod a+x runFoamContainer.sh
```

The actual URL for the script is <http://hg.code.sf.net/p/openfoam-extend/ansiibleFoamInstallation/raw-file/f7b5a1b60e3f/scripts/runFoamContainer.sh>

Starting the container

```
> ./runFoamContainer.sh bgschaid/openfoam_by_ansiible:training_programming_pyfoam_ofv15
```

After that you're on a shell inside the container

What runFoamContainer.sh does

The purpose of this script is to make using the Docker container as painless as possible

- Without an argument the script lists the **locally** available containers compatible with the script
- With an image name it starts the image in a new container
- mounts the working directory on the host machine to /foamdata on the container
 - data written to that directory is written to the host machine
 - and can be read during the next start of the machine
- Sets the user id of the user in the container to the id of the user on the host machine
 - Can read and write the same files as the host user

Starting the container

This demonstrates how data written inside the container is written to the host machine (rechenknecht001 is the name of the host. testuser is the name of the user on the host)

```

Till: Default
1:testuser@rechenknecht001:~$ 
[testuser@rechenknecht001 ~]$ ls
runFoamContainer.sh
[testuser@rechenknecht001 ~]$ ./runFoamContainer.sh
Start with image and current directory: ./runFoamContainer.sh <image>
Start with image and specified directory: ./runFoamContainer.sh <image> <directory>

Proper images

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
bgschaid/openfoam_by_ubuntu1804_with_of7_swak4foam  7936de0accdf       12 days ago        2.09GB
[testuser@rechenknecht001 ~]$ ./runFoamContainer.sh bgschaid/openfoam_by_ubuntu1804_with_of7_swak4foam
User: dockeruser with UID: 2003 and GID: 2003
(OF:7-Opt) dockeruser@fd98dac65372:/foamdata$ ls
runFoamContainer.sh
(OF:7-Opt) dockeruser@fd98dac65372:/foamdata$ mkdir test
(OF:7-Opt) dockeruser@fd98dac65372:/foamdata$ ls
runFoamContainer.sh  test
(OF:7-Opt) dockeruser@fd98dac65372:/foamdata$ exit
exit
[testuser@rechenknecht001 ~]$ ls
runFoamContainer.sh  test
[testuser@rechenknecht001 ~]$
  
```

Figure: Docker container started and data written to local machine

Getting the Material

With docker

The docker image has the use-case in the directory /Examples


```
> runFoamContainer.sh bgschaid/openfoam_by_ansiible:training_programming_pyfoam_ofw15
User dockeruser with UID: 2000 and GID: 2000
> (venv) (OF:v1912-Opt) dockeruser@f565cbbd864d:/foamdata$ ls /Examples/
bottleRocketTemplate  calcRocketHeight.py  collectData.py  requirements.txt  <brk>
<cont>rocketWithScripts.pvsm  runRocket.py  runVariation.sh
> (venv) (OF:v1912-Opt) dockeruser@f565cbbd864d:/foamdata$
```

Non-docker


The use case has been archived in an archive

```
> wget https://openfoamwiki.net/images/2/20/PyFoamProgramming_VATech2020_Material.tar.gz
> tar xvzf PyFoamProgramming_VATech2020_Material.tar.gz
PyFoamProg/bottleRocketTemplate/
PyFoamProg/bottleRocketTemplate/system/
PyFoamProg/bottleRocketTemplate/system/setFieldsDict
PyFoamProg/bottleRocketTemplate/system/blockMeshDict
PyFoamProg/bottleRocketTemplate/system/fvSchemes
PyFoamProg/bottleRocketTemplate/system/meshQualityDict
...
```

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Python 2 vs Python 3

- New Python versions are usually backwards-compatible
 - Makes sure that old code doesn't break with new Python versions
 - Problem is that this hinders some extensions
- In 2008 Python 3 was released
 - Has some changes that are incompatible with Python 2
 - Since then people are asked to migrate to Python 3
 - There are tools and libraries to assist developers with this
- Since 2020 Python 2 is no longer supported
 - But in reality it is still widely used
 - Default python in Linux LTS (*Long Term Support*) distributions like RedHat/CentOS and Ubuntu 18.04
 - LTS us usually used in work and server systems
 - Python 3 can usually by installed in parallel and is available as python3

Which version does PyFoam support

- PyFoam works on Python 2 **and** 3
 - by avoiding Python 3 features that can not be emulated in Python 2
 - and using the `six` library
 - a library that "irons out" differences in the version
- Advantage
 - PyFoam runs on almost every currently used Linux-distro
- Disadvantage
 - Some cool features of newer Python 3 versions can't be used
 - But that might also break compatibility with older Python 3 versions as well
- PyFoam is tested with these Python versions
 - 2.7 Python 2.6 **might** still work. It is broken for Python 2.5
 - 3.4 **and newer** Older Python 3 versions are not supposed to be stable

Which Python version should I use?

Depends


On the computation machine

- Be conservative
- Use the Python that is the default on this machine
 - Python 2.7 is currently the default
- It is usually not worth the trouble to install a special Python
 - Library/ABI incompatibilities
 - Administrative red tape if you're not the admin of that machine

On your workstation

- Be adventurous
- Use the newest Python you can get binary packages for on your machines
- Which is usually some Python 3 (usually not the newest)
- If the scripts have to work on a cluster try to install a similar Python version for testing

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Additional libraries

- PyFoam tries to use as little external libraries as possible
 - The most essential ones it brings in `ThirdParty`
 - Allows "freezing" the versions that are known to work
 - Slightly adopts them
 - The only external library requirement it has is `numpy`
- Most PyFoam-utilities are happy with these requirements
- But it uses some external libraries as options
 - Some of these are not available as binary packages for all distros
 - or in outdated form
- So the best thing is to install them by them-self
- PyFoam tries not to replicate existing libraries but leverage them if they're installed

Installing with pip

- The de-facto standard for Python package installation is pip

Allows searching for packages

```
> pip search PyFoam
PyFoam (2020.5) - Python Utilities for OpenFOAM
  INSTALLED: 2020.5 (latest)
droneCFD (0.1.3) - A virtual wind tunnel based on OpenFOAM and PyFOAM
```

And installing them

```
> pip install PyFoam
```

Upgrading if there is a new release

```
> pip install --upgrade PyFoam
```

Installing for the current user if you haven't root privileges

```
> pip install --user PyFoam
```

numpy and scipy

- numpy is the foundation for most numerical/scientific Python-libraries
 - implements fast operations on multi-dimensional numerical array
- scipy is a collection of numerical algorithms
 - ODE solving
 - optimization
 - ...

Getting the largest Eigenvalue of a matrix

```
import numpy as np
# matrix with 1000x1000 random numbers between [-1,1]
a=np.random.rand(1000,1000)*2-1
# Largest Eigenvalue (not that we need it)
print(np.linalg.eigvals(a).max())
```

matplotlib

- Not as fast as Gnuplot but better looking

Set up the plot

```
from matplotlib import pyplot as plt
plt.xkcd()
plt.xlabel('nr')
plt.ylabel('value')
plt.show()
```

The result

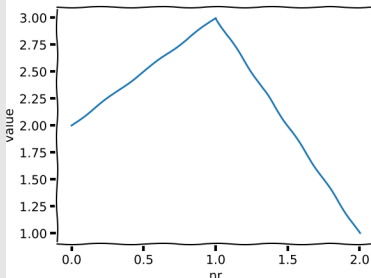


Figure: A sketchy figure

pandas

Pandas

- A library for table-type data
- Based on numpy and uses matplotlib for plotting
- Easy import and export from/to text files, Excel, CSV, databases, HTML
.....
- Basically *Excel for grown-ups*
 - A bit harder to use but gives reproducible results

Reading data from the net

```
import pandas as pd
water=pd.read_html("https://wiki.anton-paar.com/en/<brk>
<cont>water/")[0]
water=water.rename(mapper=lambda n:n.strip().rsplit(<brk>
<cont>maxsplit=1)[0],axis=1).set_index("Temp.")
water.describe()
water.plot()
```

Output

	Dyn. Viscosity	Kin. Viscosity	Density
Temp.			
2	1.6735	1.6736	0.9999
3	1.6190	1.6191	1.0000
4	1.5673	1.5674	1.0000
5	1.5182	1.5182	1.0000
6	1.4715	1.4716	0.9999
7	1.4271	1.4272	0.9999
.....			

ipython

- `python` without arguments starts a REPL (*Read, Eval, Print, Loop*) shell
- `ipython` is an improved REPL
 - Improved auto-completion with the Tab key
 - even assists for `import`
 - this includes the file system
 - Improved history of commands
 - Utility functions prefixed with `%`
 - For instance: `%time` to get the execution time of a command
 - Interaction with known libraries
 - `%pylab` loads `numpy` and `matplotlib` in such a way that it is similar (as far as possible) to a *Matlab* environment
 - Most important: it uses color

Getting help in ipython

- In the pyrthon REPL help on a method can be got like this

```
help(str.rstrip)
```

- In ipython only a ? has to be post-fixed

```
str.rstrip?
```

- For "improved help" (usually the source code - if available) use ??

```
import PyFoam  
PyFoam??
```

Developing a program with ipython

- We're CFD-experts
 - Not pandas / matplotlib / PyFoam experts
- So the best way is to use ipython during the development of a script:
 - 1 start up ipython
 - 2 try the first steps interactively in the REPL
 - 3 if it works: copy to the script in the text-editor
 - 4 run the script

```
ipython -i theScript.py
```


(-i starts the REPL at the end of the script)

- 1 Try the next steps
- 2 Copy to editor and repeat 4 until finished

Jupyter notebooks

- this used to be ipython notebook
 - but now is much more
 - and supports other languages
- Instead of the REPL you get a notebook interface in a web-browser
 - allows adding documentation via Markdown cells
- a truly OpenSource alternative to MatLab/Mathematica-notebooks
- but this is beyond the scope of this training

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - **Virtual environments**
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Problem with library dependencies

- All theses nice libraries bring problems
 - "There are no official binaries"
 - "The admin doesn't want me to pollute the system with all this stuff"
 - "I need to re-use this script. What are the required libraries?"
 - "An update of the library broke my script"
 - "Project A needs version X, project B version Y of this library"
- *Virtual environments* are the solution to these problems

The venv module

- Is part of Python since 3.3
 - Some distros (looking at you Ubuntu) require you to install it separately
- It creates a *Virtual environment* which basically is
 - Symbolic links to the "real" python and other utilities (pip for instance)
 - **no** libraries
- To use the *Virtual environment* it has to be *activated*
 - and afterwards it can be *deactivated*
- Install the necessary libraries into the environment
- Start working

Creating a new environment

■ In the project directory

```
> python3 -m venv venv
> ls venv
bin  include  lib  lib64  pyvenv.cfg  share
> ls venv/bin
activate  activate.csh  activate.fish  easy_install  easy_install-3.6
pip  pip3  pip3.6  python  python3
```

- Make sure that the `python`-binary is the version of Python you want to use
- The second `venv` is the name of the virtual environment
 - Choose a different one if necessary

Activating and deactivating

- To use the environment has to be activated
 - This means that `python` is now the `python` from the environment
 - and the libraries are the ones from the environment
 - this is only for the current shell
- It can be deactivated as well
 - but not necessarily: when the shell ends it ends

```
> which python
/usr/bin/python
> source ./venv/bin/activate
> which python
/path/to/the/project/venv/bin/python
> deactivate
> which python
/usr/bin/python
```

- Usually `(venv)` on the shell prompt indicates that `venv` is activated

Creating a requirements.txt file

- In an activated environment packages can be installed

```
> pip install pandas
```

- Then a list of all installed libraries can be generated
 - this includes the version numbers

```
> pip freeze >requirements.txt
```

- Put requirements.txt under version control
 - And make sure to **exclude** venv from version control
- Afterwards you can remove the environment completely

```
> deactivate  
> rm -r venv
```

- Later you can reproduce the environment on a different machine

```
> python3 -m venv venv  
> source ./venv/bin/activate  
> pip install -r requirements.txt
```

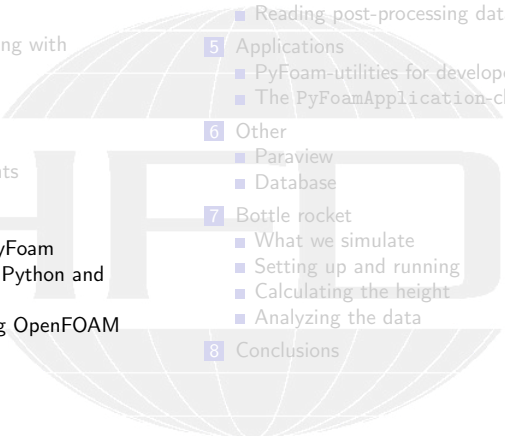
Activating the environment in the docker image

On the docker image the necessary venv is already activated


```
> runFoamContainer.sh bgschaid/openfoam_by_ansible:training_programming_pyfoam_ofw15
User dockeruser with UID: 2000 and GID: 2000
(venv) (OF:v1912-Opt) dockeruser@3cf5c10effd1:/foamdata$ which python
/venv/bin/python
(venv) (OF:v1912-Opt) dockeruser@3cf5c10effd1:/foamdata$ ipython
Python 3.6.9 (default, Apr 18 2020, 01:56:04)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.14.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

The basic sub-modules

The PyFoam library is structured into some sub-modules

Basics basic classes used by other modules

Execution executes and controls a running OpenFOAM-application

RunDictionary representing OpenFOAM-data on disk

LogAnalysis analyzing the output of OpenFOAM applications

Wrappers adapters for other libraries

Infrastructure classes that let PyFoam interact with the outside world

Applications concrete implementations of the utilities

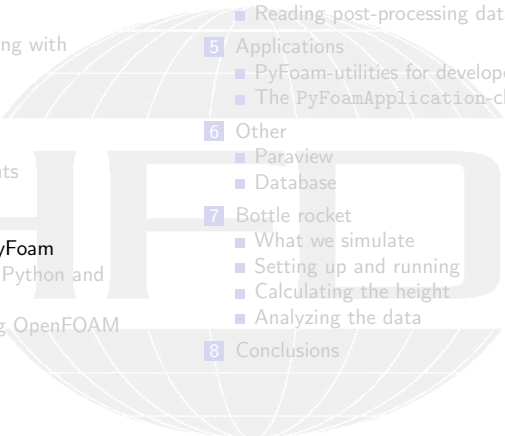
Paraview interacting with the famous post-processor

ThirdParty libraries from other developers that are included to ease deployment

Unit-tests

- The directory `unittests` has the unit-tests of PyFoam
- Unit-tests ensure that the basic functionality of a library is not broken by changes
- These unit-tests are run on different Python versions before each release
 - Sadly only a fraction of the library is covered by the tests
- The unit tests are run by `py.test`

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - **File handling by PyFoam**
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

To zip or not to zip

- OpenFOAM transparently zips and unzips files
 - when OpenFOAM tries to access a file T it needs one of two files
 - T.gz the zipped version of the file
 - T the plain text version
- This might be problematic for scripts: looks for T but there is only T.gz
- PyFoam tries to transparently handle files in the same way

The File-basis class

- The class `FileBasis` is in charge of reading and writing a file (or its .gz-brother)
 - The variant `FileBasisBackup` automatically creates a backup of the file if something goes wrong
- Reading of the file has to be initiated with `readFile()`
- The member `content` of the object is a string with the content
 - This can be manipulated
- `writeFile()` writes content back
 - zipped if the original was zipped
 - `writeFileAs()` writes to a different file

Example: Reading a file

Adding to a file

```
from PyFoam.RunDictionary.FileBasis import FileBasis
f=FileBasisBackup("pitzDaily/system/controlDict", backup=True)
f.readFile()
f.content+="\n// This file was touched by PyFoam\n"
f.writeFile()
```


The result

```
> tail -1 pitzDaily/system/controlDict
// This file was touched by PyFoam
> tail -1 pitzDaily/system/controlDict.backup
// ***** //
```

SolutionDirectory

- The class `SolutionDirectory` represents a OpenFOAM-directory
 - Has methods to access the important sub-directories
 - and the `controlDict`
 - and time directories
 - gives info about processor-directories
 - manipulates the directory
 - clear time-steps
 - clones the directory
 - etc
- It is important and. But we won't talk about it today

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - **Data structures in Python and OpenFOAM**
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Comment on the OpenFOAM file-format

- That format was conceived in the 90s
 - **Before** everyone and his dog used XML in their projects
- It is less rigid than XML
 - ... but not as "type safe"
- It is basically equivalent to "modern" ASCII file formats
 - JSON
 - YAML
- None of these formats would solve the problems with the OpenFOAM file format
- So the question: "why not use JSON/YAML/blah" is pointless

Dictionaries

In OpenFOAM

```
a 1;  
b nix  
c {  
  d 3.1415;  
  e "This_is_a_string";  
}
```

In Python

```
{  
  "a" : 1,  
  "b" : "nix",  
  "c" : {  
    "d" : 3.1415,  
    "e" : "This_is_a_string",  
  }  
}
```

Lists and tuples

Difference: lists are homogeneous in OpenFOAM

In OpenFOAM

```
a (1 2 3 4);  
b (x y z);  
c foo 2;
```

In Python

```
{  
    "a" : [1, 2, 3, 4],  
    "b" : ["x", "y", "z"],  
    "c" : ("foo", 2),  
    "d" : ["a", 2, 2.3] # not possible  
}
```

Problem: Ambiguities of the file format

Is it a vector

```
a (1 2 3);
```

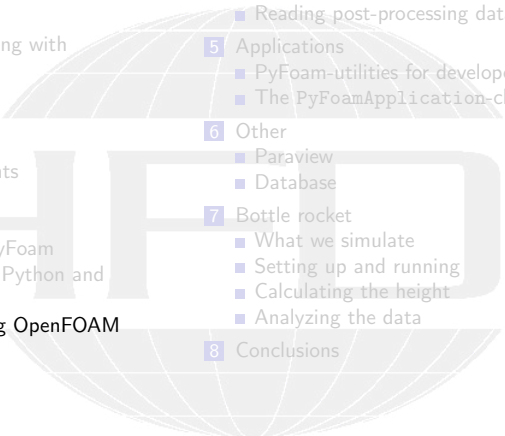
could be

- a list of label (integer)
- a list of floating point numbers (that happens to be 3 elements long)
- a vector

How OpenFOAM resolves this

- Only the OpenFOAM-program that reads this knows what it expects here
 - Because it calls the C++-functions to construct a value
- Everyone else has to **guess**
 - That includes other OpenFOAM-programs
- Guessing leads to misunderstandings

Outline

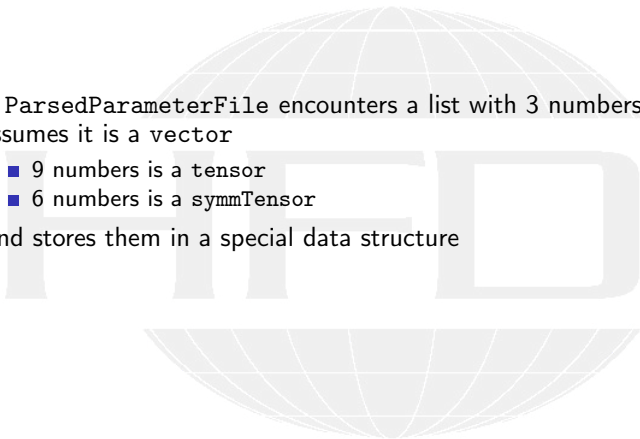
- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

ParsedParameterFile

- This class does the heavy lifting in PyFoam
 - Reads an OpenFOAM-file and transforms it into Python-data structures
 - Understands extensions of the OpenFOAM file format like Macro expansions and regular expressions as dictionary keys
 - Tries to resolve ambiguities in a sensible way
 - After reading can be treated like a regular dictionary
 - items can be read, added and modified
 - The result can be written to disk
 - Messes up the formatting (nothing that couldn't be fixed with 2 weeks of programming and the funding for it)
- It is implemented using the ply parser generator
 - A python library modeled after the yacc / bison utility
 - The library is included in ThirdParty
- It is **not** good for parsing huge data files: too slow

Resolution of ambiguities

- If `ParsedParameterFile` encounters a list with 3 numbers it assumes it is a vector
 - 9 numbers is a tensor
 - 6 numbers is a `symmTensor`
- And stores them in a special data structure



Different starting points

- In the standard form `ParsedParameterFile` assumes that the file is a dictionary with a header
 - this is not always the case:
 - some files have no header
 - in some files the content is a list
 - in such cases a different "starting point" has to be specified by an option to the constructor
- There are other options to the constructor that change the behaviour
 - avoid macro expansion
 - don't "guess" that a 3 element list is a vector
 - etc

Example: Reading and writing controlDict

Only 10 writes

```

from PyFoam.RunDictionary.ParsedParameterFile import ParsedParameterFile
cd = ParsedParameterFile("pitzDaily/system/controlDict")
cd["writeInterval"] = int(cd["endTime"] / 10)
cd.writeFile()

```

controlDict

```

// -*- C++ -*-
// File generated by PyFoam - sorry for the ugliness

FoamFile
{
  version 2.0;
  format ascii;
  class dictionary;
  location "system";
  object controlDict;
}
...
endTime 2000;
...
writeInterval 200;

```

FoamFileGenerator

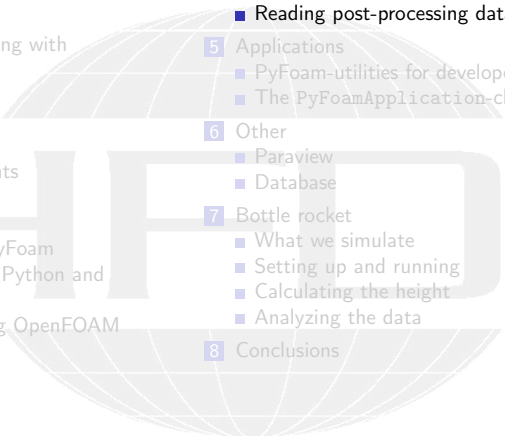
- This class transforms Python structures into OpenFOAM-format
 - usually there is no need to use it directly as `ParsedParameterFile` does it
- If it has to be used the easiest way is through the `makeString`-utility function

Writing Python data as OpenFOAM


```
In [1]: from PyFoam.Basics.FoamFileGenerator import makeString

In [2]: makeString({"a":2, "b": [2.3, 4,5]})
Out[2]: 'a_2;\nb_(2.3_4_5);\n'
```

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

The pickle format

- This is a binary format that "pickles" Python data
 - and allows unpickling this data later in almost the same form
- PyFoam uses it to store data in the *.analyzed directories
 - to be re-used by other programs
- to read the data use the pickle library directly
- Utilities that use that data are
 - pyFoamEchoPickledApplicationData.py prints that information
 - pyFoamRedoPlot.py reads stored plot data and re-generates the plot
 - but also allows using the plot data with pandas

Data written by PyFoam

- Usually there are 4 files found in the analyzed-directory
 - `pickledData` data about the run
 - `pickledUnfinishedData` written at regular intervals so that a killed process leaves **some** data
 - `pickledStartData` written at the start
 - `pickledPlots` data that would usually be plotted
- in the `pickledData` are things like
 - when the run was started and how long it ran
 - solver used and which OpenFOAM-version
 - number of time-steps
 - whether the run ended OK
 - the **last** value of analyzed data (from the `customRegex`)
 - `uniqueid` : a id for the run. Calculated in such a way that no two runs in the universe should have the same id
 - ...

Example: Average time spent on each timestep

Read Data

```
import pickle
data = pickle.load(open("PyFoamRunner.simpleFoam.analyzed/pickledData", "rb"))
print(data["wallTime"] / data["time"])
```

Note: because it is a binary format the b in "rb" is necessary

Outline

1 Introduction

- This presentation
- Who is this?
- What are we working with
- Before we start

2 Python Ecosystem

- Python versions
- Packages
- Virtual environments

3 The workhorses

- Library structure
- File handling by PyFoam
- Data structures in Python and OpenFOAM
- Parsing and writing OpenFOAM dictionaries

4 Reading data

- Recycling PyFoam data
- Reading post-processing data

5 Applications

- PyFoam-utilities for developers
- The PyFoamApplication-class

6 Other

- Paraview
- Database

7 Bottle rocket

- What we simulate
- Setting up and running
- Calculating the height
- Analyzing the data

8 Conclusions

What is post-processing data?

My definition:

- Text file data that was written to the directory `postProcessing` by function objects

Not anything in the time-directories

- Typically that data is
 - Timelines: time dependent values typically written by
 - probes
 - swak4Foam
 - ...
 - Profiles: spatially distributed data written at certain time-steps
 - sampledSet
- PyFoam has classes for handling those

Timelines

- `TimelineDirectory` is in charge of handling those
 - Constructed with the parameters
 - 1 Case name
 - 2 Sub-directory
 - Maintains a hierarchy of data
 - 1 Time-Directory. Necessary if there have been restarts of the simulation (0 is used by default)
 - 2 Files in these (called positions)
 - 3 The values in this form
- Operator `[]` gives back the data at one position as a `SpreadsheetData`-object
 - a wrapper around a numpy array
 - method `getData()` returns a pandas `DataFrame`
- This will be used later in the example

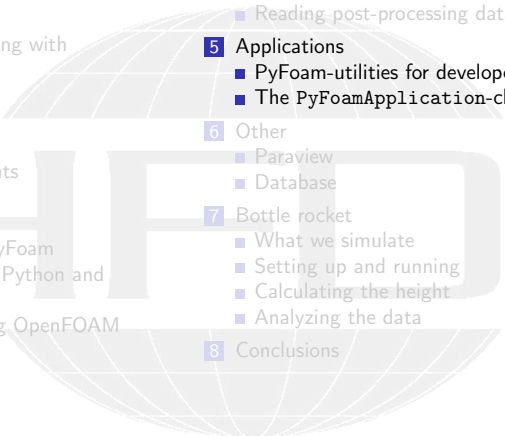
Profiles and Lagrangian data

- `SampleDirectory` is a similar class for profiles
 - Here the time-directory is important because
- `LagrangianCloudData` reads lagrangian particle data and makes it available as a `DataFrame`
 - Possibly broken in some OF-versions because of different file formats
- `LagrangianPatchData` reads data from a function object that records incidences on a patch

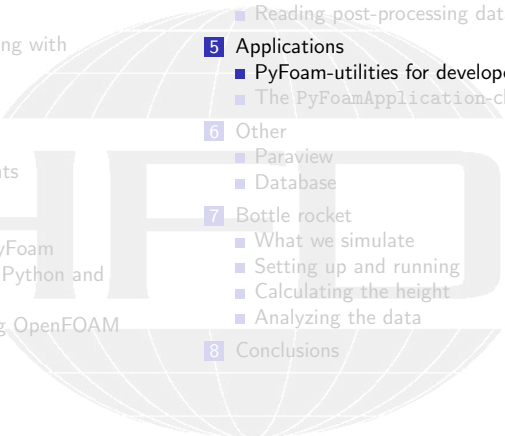
PyFoamDataFrame

- This is a wrapper around the DataFrame in pandas
 - Assumes that the *index* is a monotonically rising row of numbers
 - this is certainly true for time, often for x , y or z
- Adds some functionality
 - `weightedAverage()` averages curves values weighted by their time-steps
 - `integrate()` integrates values over time using the trapezoid-rule
 - this and `weightedAverage` are included in the output of `describe()`
 - extends the sub-script operator `[]` to make the table behave like a continuous function
 - if the argument is a number it interpolates on the *index*
 - if it gets a list of numbers it returns a list with the values interpolated for these indices
 - otherwise it defaults to the `[]` operator of the regular DataFrame

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Developer options

Every PyFoam-utility has some options that are of interest for programmers (see `--help` in the Debugging-section)

- `-traceback-on-error` prints a complete traceback if there is an error
- `-interactive-debugger` drops the user to an interactive REPL for debugging if there is an error
 - based on `ipython` if that is installed
- `-i-am-a-developer` switches on these and a number of other related options
- `-interactive-after-execution` if utility runs `ok` drops the user to an interactive REPL
 - just like the regular `-i` option of `python`
 - allows

Result data

- When dropped to the REPL with `-interactive-after-execution` the variable `self` is the application class
 - See below on that
- `self` can be inspected
 - theoretically all methods could be used
- `self.getData()` returns a dictionary with data "that might interest you"
 - for the Runner-utilities this is the data that would be written to `pickledData`
 - the analyzed sub-dictionary has the `last` values from things you see in plots
 - some utilities (for instant `pyFoamRedoPlot.py`) have instructions like `--pandas-data` who make sure that the data that usually would be plotted is found as a `DataFrame` there

Example: examining data after the run

Inside the pitzDaily case

```
> pyFoamRunner.py --progress --interactive-after-execution auto
```

```
t = 280
```

```
Dropping to interactive shell ... found IPython ...up-to-date IPython
```

On the REPL

```
In [1]: self.getData()
```

```
Out[1]:
```

```
{'lines': 2887,  
'uniqueid': '00af2198-b59a-11ea-984e-0242ac110002',  
'logfile': './PyFoamRunner.simpleFoam.logfile',  
'casefullname': '/tmp/pitzDaily',  
'casename': 'pitzDaily',  
'solver': 'simpleFoam',  
'solverFull': 'simpleFoam',  
'commandLine': 'simpleFoam',  
'hostname': 'fd719ea531f7',
```

```
In [2]: d=self.getData()
```

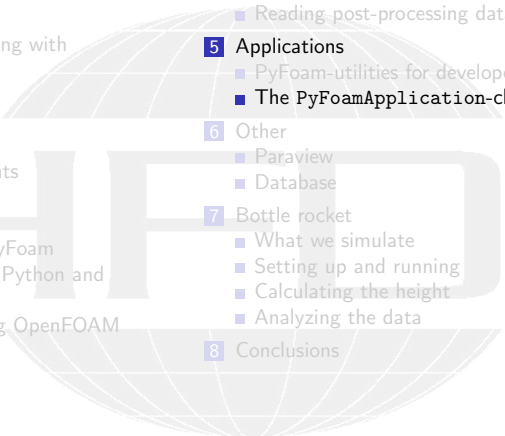
```
In [3]: d["wallTime"]/d["stepNr"]
```

```
Out[3]: 0.03570623823574611
```

```
In [4]: d["analyzed"]["Iterations"]
```

```
Out[4]: {'Ux': 5.0, 'Uy': 6.0, 'p': 5.0, 'epsilon': 3.0, 'k': 4.0}
```

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - **The PyFoamApplication-class**
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

One class to implement them all

- Almost all PyFoam-utilities are based on the PyFoamApplication-class
- This ensures a consistent behavior
 - error handling
 - similar initialization
 - common command line options
- all these classes are found in the Applications sub-module
 - classes that start with Common implement functionality that is the same for 2 or more classes
 - for instance common options for pyFoamPlotRunner.py and pyFoamPlotWatcher.py
- when constructed the application immediately executes its run()-method
- an object of that class is the self we see with --interactive-after-execution

The boilerplate script

- Most utilities are implemented in the same way:
 - 1 import the application class
 - 2 construct an object of that class
 - which implicitly executes `self.run()`

```
pyFoamPlotRunner.py
```

```
#!/venv/bin/python3  
  
from PyFoam.Applications.PlotRunner import PlotRunner  
  
PlotRunner()
```

Using the application class

Using an application class in a program is almost the same as in the utility

- 1 Import the application class
- 2 Construct it
 - give the constructor an `args`-argument
 - otherwise it would use arguments from the command line
 - this is a list of strings
 - each string is an option from the command line options of that utility
 - this immediately runs it
 - including side-effects like printing
 - afterwards results can be collected with `getData()`

What you say on the command line

```
> pyFoamRunner.py --clear --progress auto
```

What to say in the program

```
from PyFoam.Applications.Runner import Runner
theSolver = "simpleFoam"
result = Runner(args["--progress", "--clear", theSolver]).getData()
```

Outline

1 Introduction

- This presentation
- Who is this?
- What are we working with
- Before we start

2 Python Ecosystem

- Python versions
- Packages
- Virtual environments

3 The workhorses

- Library structure
- File handling by PyFoam
- Data structures in Python and OpenFOAM
- Parsing and writing OpenFOAM dictionaries

4 Reading data

- Recycling PyFoam data
- Reading post-processing data

5 Applications

- PyFoam-utilities for developers
- The PyFoamApplication-class

6 Other


- Paraview
- Database

7 Bottle rocket

- What we simulate
- Setting up and running
- Calculating the height
- Analyzing the data

8 Conclusions

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Python in Paraview

- Paraview uses python as a scripting language
 - including *Programmable Filter* and *Programmable Source*
- So PyFoam can be easily integrated
 - Tricky part is that the python in ParaView finds it
 - Either point the PYTHONPATH environment variable on the shell where you call paraview to it
 - Manipulate `sys.path` (not very portable)

Paraview support in PyFoam

- Paraview support in PyFoam is of varying quality
 - Some of the code was developed with Paraview 3.x
- All is found in the module Paraview
 - Submodules are

SimpleSources, **SimpleFilters** filters and sources that were developed before the *Programmable Sources/Filters*

StateFile manipulates the .pvsm files (used in pyFoamPVSsnapshot.py)

Data reads timeline and profile data and makes it available as vtkTable

- Under the hood most of them use a function caseDirectory() that
 - searches the internal data structures of ParaView for the OpenFOAM-reader
 - returns the location of the case on disk
- With that function scripts can find all other info

Examples of using PyFoam.Paraview.Data

These can be found in `rocketWithScripts.pvsm` in the `/Examples`

A *Programmable Source* that returns a `vtkTable`

```
import PyFoam.Paraview.Data as pfData
pfData.setTimelineData(output, "swakExpression_thrustCoeff")
```

Can be directly visualized in a line-plot view

A *Programmable filter* that returns the location of the highest velocity

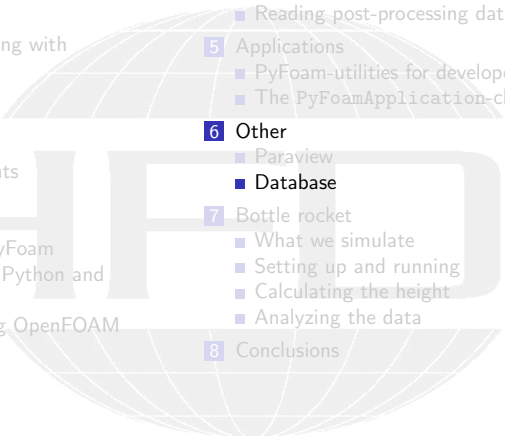
```
import PyFoam.Paraview.Data as pfData
import numpy as np

actualDir=pfData.checkDir("postProcessing/swakExpression_highSpeedLoc")
data=pfData.TimelinePlot(args=[pfData.case().name,
                              "--directory="+actualDir,
                              "--basic-mode=lines",
                              "--numpy"]).data

index=np.where(data["time"]>=pfData.vTime())
pos=data[index][0]
for name in pos.dtype.names:
    output.RowData.append(pos[name], name)
```

Resulting `vtkTable` needs to go through a `TableToPoint` and then into `Glyph`

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - **Database**
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

SQLite

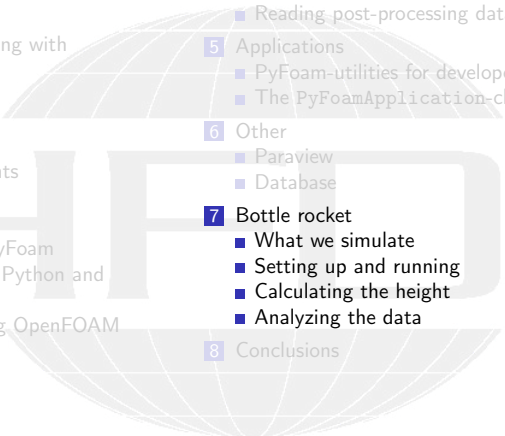
- SQLite as a serverless SQL database
- Used in many products as a safe way to store data
 - old *iPods* use it to store data
 - *Android* phones often use it to store application data
 - ...
- Python has support for it in its standard-library

So PyFoam uses it too


SQLite support in PyFoam

- There is a class `RunDatabase` that stores data that usually is in `pickledData` into a SQLite database
 - "flattens" dictionaries to fit into the table format
 - creates new columns "on demand"
- most important method is `db.modify(id,dict)` that modifies the data of the case with ID `id` with the data in `dict`
- based on this there are two utilities
 - `pyFoamAddCaseDataToDatabase.py` that adds `pickledData` to a database
 - creates the database if needed
 - `pyFoamDumRunDatabaseToCSV.py` dumps the run database into a format that Excel can understand
- With these utilities you can consistently collect data and analyze it later with well known tools
 - or with `pandas`. See below

Outline

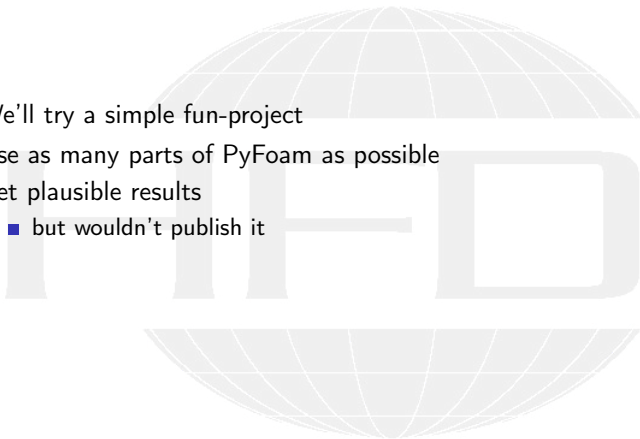
- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - **What we simulate**
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Overview

- We'll try a simple fun-project
- Use as many parts of PyFoam as possible
- Get plausible results
 - but wouldn't publish it



The DLR water rocket

- This is a toy-project specified by DLR (German Aerospace Center) for pupil science education
 - <http://bit.ly/1hUIP91> (sorry: didn't find an English version)

Principle

- Plastic bottle
 - Valve attached at the neck of the button
- Bottle is partially filled with water
- Bicycle pump is attached to the valve
 - Air in bottle is pressurized
- Valve is opened
 - Pressurized air pushes the water out
 - Water pushes the bottle high into the air


Example



The question we're asking

- There are two parameters we have
 - 1 Amount of water
 - more water: more possible force
 - but less air "stores" less energy
 - 2 Pressure of the air
 - Higher pressure stores "more energy"
 - but there are limits (because of the material)
- The questions are:
 - 1 What is the optimum amount of water
 - energy "stored" by the air vs momentum of the water
 - 2 Do high pressure have a diminishing return
 - Does higher pressure result in proportionally bigger heights?

Outline

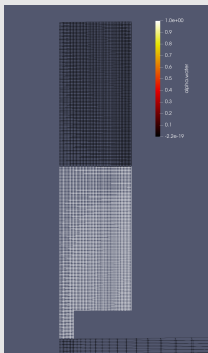
- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - **Setting up and running**
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Geometry, Solver and boundary conditions

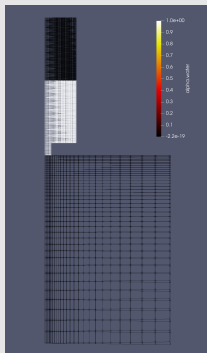
- Axial-symmetric mesh for fast calculations
 - evenly spaced mesh inside the bottle
 - extended mesh region outside the bottle-neck to avoid effect of the boundary
- Solver: `compressibleInterFoam`
 - VOF with a compressible phase
- Boundary conditions on the bottle wall are trivial: "Thou shall not pass"
 - Far-field boundary everywhere else
- Initial conditions
 - 0 velocity everywhere
 - higher pressure in the bottle
 - bottle partially filled with liquid
- We don't open a valve. We just start the simulation

Meshes

The actual bottle



Bottle with extended mesh



Preparing and running the case

- Python-script with two parameters
 - 1 height of water in bottle (in *cm*)
 - 2 initial pressure (in *Pa*)
- What the script does
 - 1 Clones a template case
 - 2 manipulates `setFieldsDict` with the initial conditions
 - 3 manipulate precision in `controlDict` to avoid a problem with `collapseEdges`
 - 4 runs `pyFoamPrepareCase.py`
 - could run all utilities by hand as well
 - 5 reset `controlDict`
 - 6 run the solver

Administrative part of the script

Start of runRocket.py

```

#!/usr/bin/env python3

import sys
from os import path

from PyFoam.Applications.PrepareCase import PrepareCase
from PyFoam.Applications.CloneCase import CloneCase
from PyFoam.Applications.Runner import Runner
from PyFoam.RunDictionary.ParsedParameterFile import ParsedParameterFile

if len(sys.argv) != 3:
    print("Script needs 2 argument: <initial_height> and <initial_pressure>")
    sys.exit(1)

initHeight = float(sys.argv[1])
initPressure = float(sys.argv[2])

if initHeight < 0:
    print("Initial height should be bigger than 0")
    sys.exit(1)

if initPressure < 0:
    print("Initial pressure should be bigger than 0")
    sys.exit(1)

caseName = "bottleRocket_h={:.1f}cm_p={:.2f}bar".format(100*initHeight,
                                                    initPressure/1e5)

CloneCase(args=["bottleRocketTemplate",
                caseName])

```


Preparing for setFields

runPocket.py continued

```
setFields = ParsedParameterFile(path.join(caseName, "system", "setFieldsDict"))
setFields["initPressure"] = initPressure
setFields["initHeight"] = initHeight
setFields.writeFile()
```

system/setFieldsDict

```
initPressure 5e5;
initHeight 0.15;

defaultFieldValues
(
    volScalarFieldValue alpha.water 1
    volScalarFieldValue p_rgh $initPressure
    volScalarFieldValue p $initPressure
);

regions
(
    boxToCell
    {
        box (-10 $initHeight -1) (10 1 1);
        fieldValues
        (
            volScalarFieldValue alpha.water 0
        );
    }
    boxToCell
```

Parameterized Geometry

constant/bottleParameters

```

bottleRadiusCm 5;
bottleHeightCm 20;
bottleWeightGram 32;

```

system/blockMeshDict

```

#include "../constant/bottleParameters"

scale 0.01;

vertices
(
    (0 -2 -0.1)
    (1 -2 -0.1)
    (1 0 -0.1)
    (0 0 -0.1)
    (0 -2 0.1)
    (1 -2 0.1)
    (1 0 0.1)
    (0 0 0.1)
    (0 $bottleHeightCm -0.1)
    ( 1 $bottleHeightCm -0.1)
    ( 1 $bottleHeightCm 0.1)
    (0 $bottleHeightCm 0.1)
    ($bottleRadiusCm 0 -0.1)
    ($bottleRadiusCm $bottleHeightCm -0.1)
    ($bottleRadiusCm $bottleHeightCm 0.1)
    ($bottleRadiusCm 0 0.1)

```

Make checkMesh happy

meshCreate.py will be executed by PrepareCase

```
#!/bin/sh
rm -r constant/polyMesh
blockMesh
makeAxialMesh -overwrite
collapseEdges -overwrite
topoSet
```

If collapseEdges works with writePrecision 6 then checkMesh won't like the resulting mesh

runRocket.py continued

```
controlDict = ParsedParameterFile(path.join(caseName, "system", "controlDict"),
                                  backup=True)
controlDict["writePrecision"] = 15
controlDict.writeFile()
```



... and lift-off

prepareCases.sh executed by PrepareCase

```
#!/bin/sh

setFields -time 0
```

runRocket.py continued and end

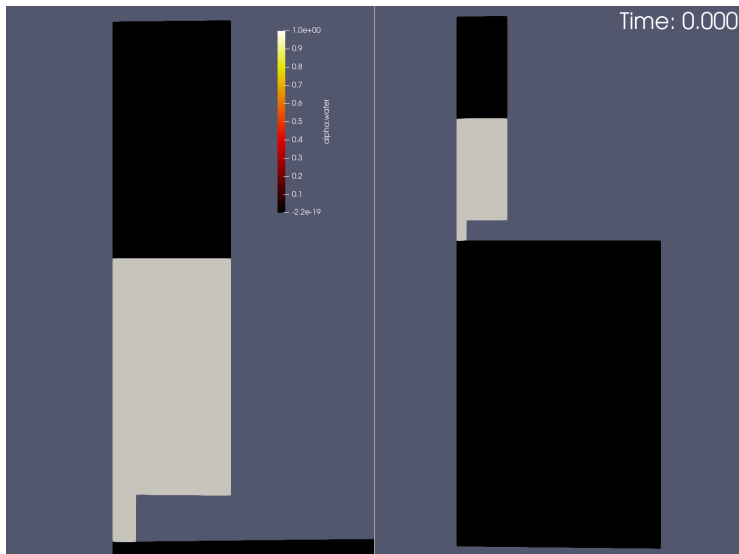
```
PrepareCase(args=["--execute-in-case-directory",
                 caseName])

controlDict.restore()

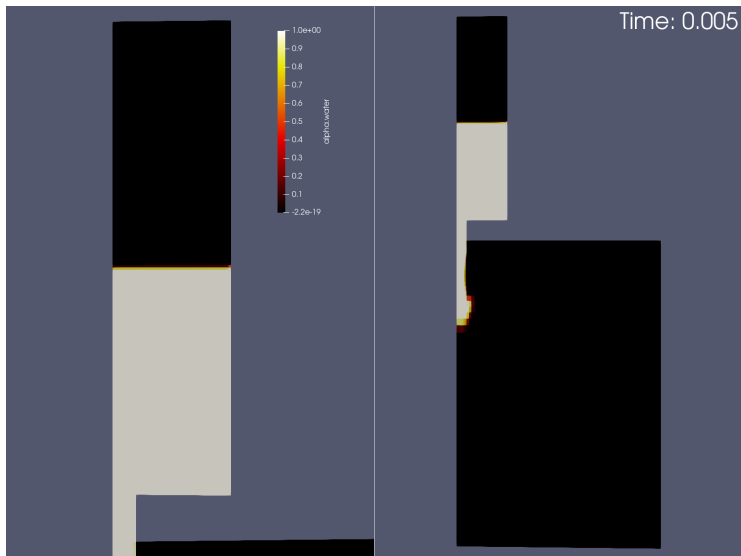
Runner(args=["--progress",
            "--parameter=initPressure:{}".format(initPressure),
            "--parameter=initHeight:{}".format(initHeight),
            "auto",
            "-case", caseName])
```

--parameter adds information to pickledData

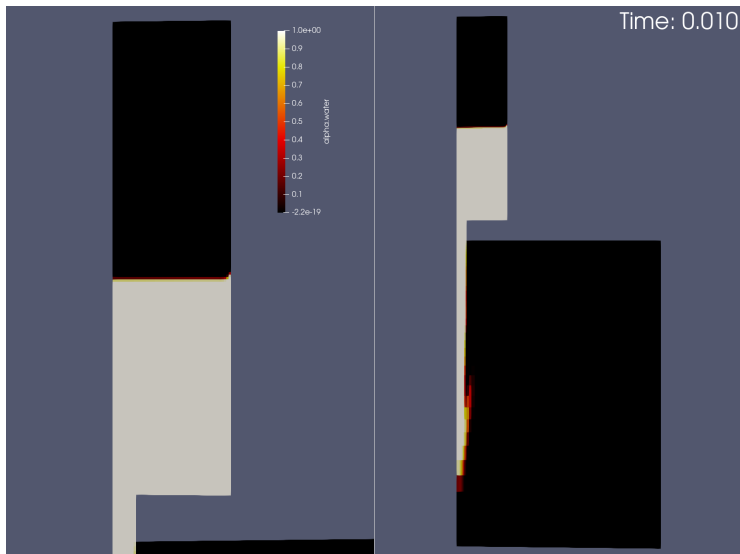
Initial condition



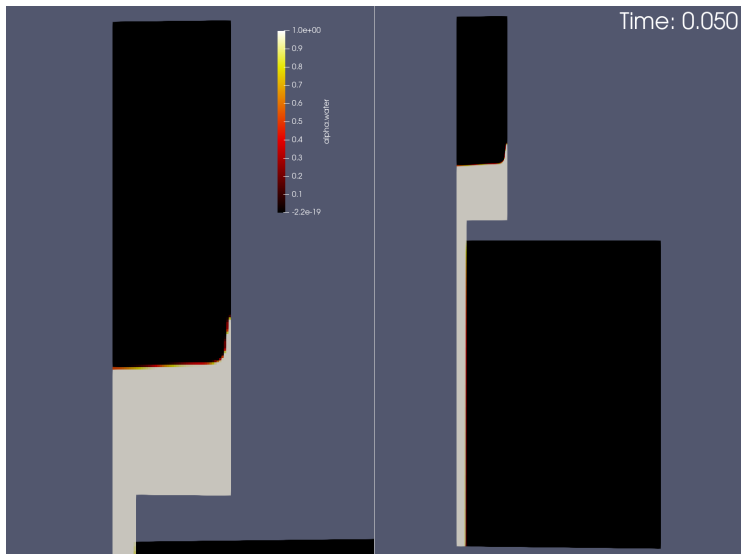
Start moving



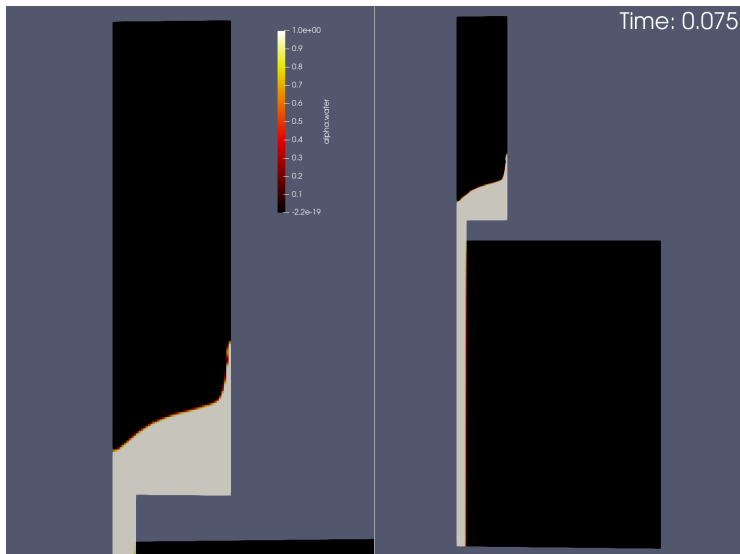
Moving



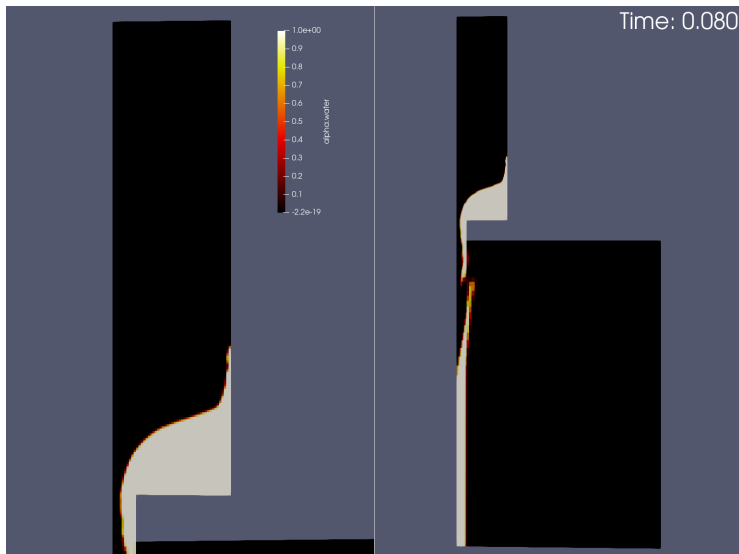
Steady flow



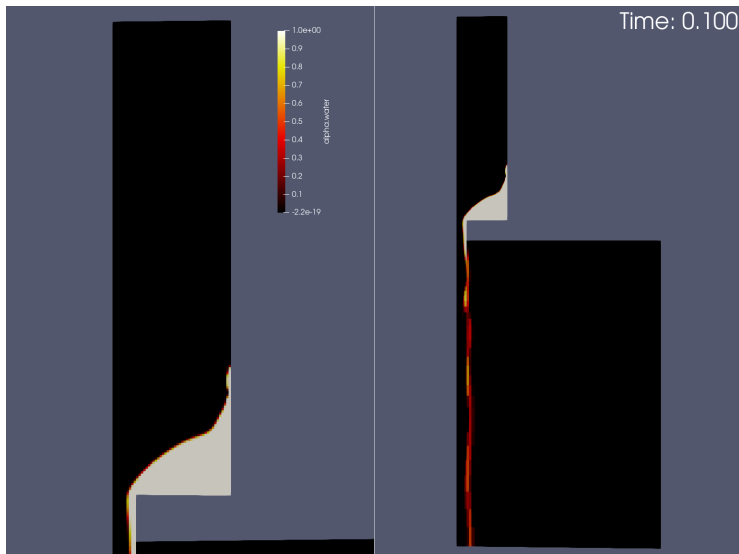
Things get interesting



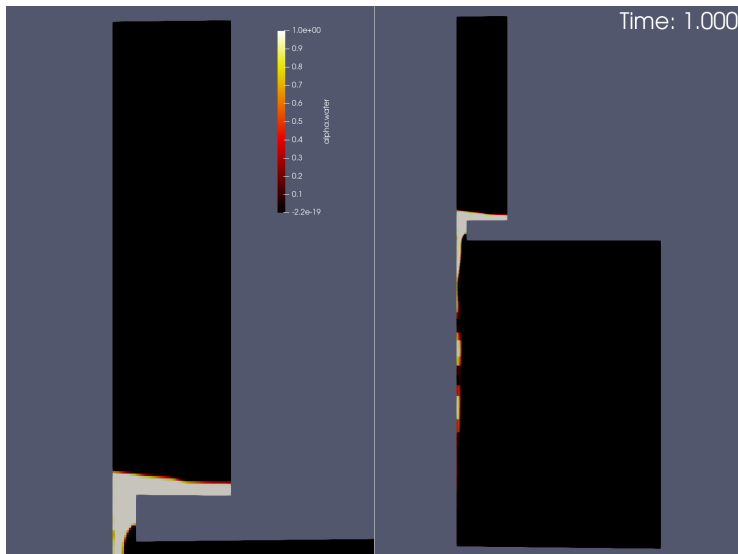
Break-through



Slow draining



Only gravitation to empty the bottle



Running a series of simulations

With the queuing manager SLURM this runs filling heights 0, 1, 2, 20 cm with a pressure of 2 bar

```
> sbatch runVariation.sh 2e5
```

This might differ on your queuing manager

runVariation.sh

```
#!/bin/bash
#SBATCH --time=7-0 --partition=longrun
#SBATCH --ntasks=1
#SBATCH --array=0-20

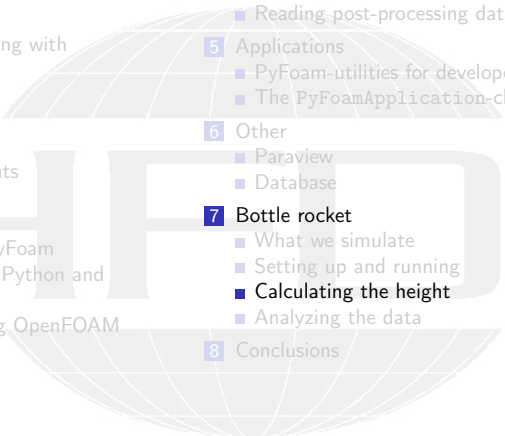
source ~/OpenFOAM/OpenFOAM-v1912/etc/bashrc
cd ~/pyfoam; . setDevelopmentPath.sh ; cd -

height=$(python -c "print(0.01*($SLURM_ARRAY_TASK_ID))")
pressure=$1

echo "Running with filling height: $height Pressure: $pressure"
hostname

./runRocket.py $height $pressure
```

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 **Bottle rocket**
 - What we simulate
 - Setting up and running
 - **Calculating the height**
 - Analyzing the data
 - 8 Conclusions

Acceleration of the bottle

Total acceleration

$$a_{total} = a_{thrust} - g + a_{drag} \quad (1)$$

with the three components

- thrust is produced by the water flushing out
- gravitation g : makes sure the bottle comes back
- drag force. Without it the bottle would go to over 100m (unrealistic)

Thrust depends on the mass flow and the outlet velocity

$$a_{thrust} = \frac{\dot{m}_{out} * v_{out}}{m(t)} \quad (2)$$

Drag depends on the mass and the bottle velocity

$$a_{drag} = -\frac{1}{2} \frac{\rho_{air}}{m(t)} \vec{v} |\vec{v}| A \quad (3)$$

How to calculate the height

- Solve the ODEs $\dot{x} = v$, $\dot{v} = a_{total}$
 - This will be done by `scipy`
- $m(t)$ and $a_{thrust}(t)$ are calculated during the simulation
 - by `swak4Foam`
 - written to disk as timelines
- Drag coefficient will be calculated from values in `bottleParameters`
- Solution of the ODE will be converted to a `PyFoamDataFrame` for easier handling
- For laughs the trajectory without drag will be calculated as well
- Material properties will be read from the case (if possible)

Calculate acceleration and mass during the simulation

This is done with swak4Foam in controlDict

Mass flow

```
totalMass {
    type swakExpression;
    valueType cellSet;
    setName inner;
    #include "../constant/bottleParameters"
    variables (
        "totalMass=sum(rho*vol())+<brk>
        <cont>$bottleWeightGram/1000/72;<brk>
        <cont>"
    );
    expression "totalMass*72";
    accumulations (
        max
    );
}
```

Acceleration

```
acceleration {
    type swakExpression;
    valueType faceZone;
    zoneName toAir;
    variables (
        "totalMass{cellSet'inner}=sum(rho*vol<brk>
        <cont>()+$bottleWeightGram<brk>
        <cont>/1000/72;"
    );
    expression "-U.y*rho*phi/totalMass";
    accumulations (
        sum
    );
}
```

Total mass over time

Bottle mass

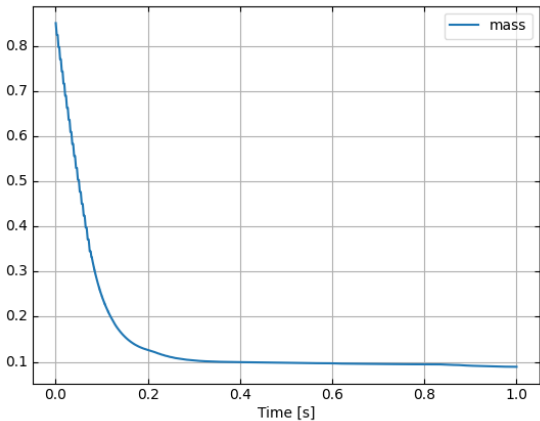


Figure: Mass of liquid **inside** the bottle

Mass flow over time

Mass flow out

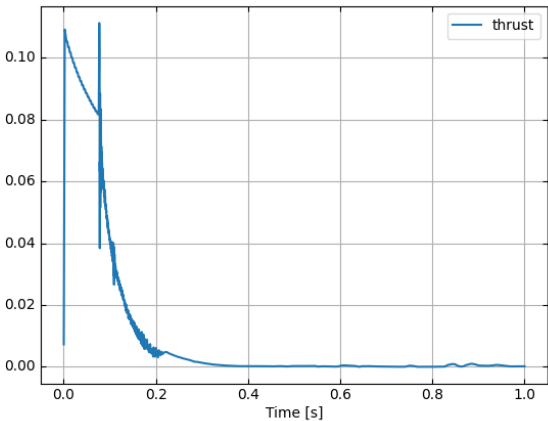


Figure: Mass flow through the neck of the bottle

Acceleration of the bottle

Acceleration

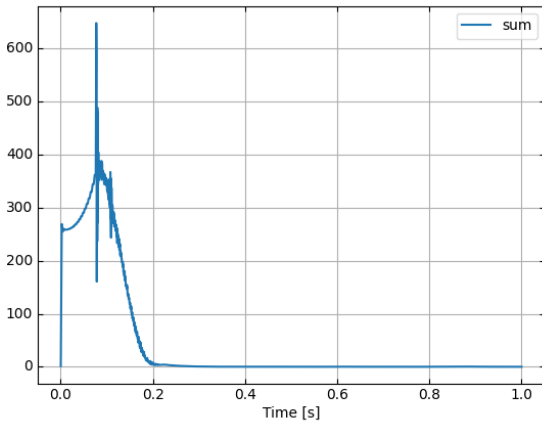


Figure: Acceleration due to thrust

Setting up the evaluation script

Start of calcRocketHeight.py

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

from os import path

from PyFoam.Applications.RedoPlot import RedoPlot
from PyFoam.RunDictionary.TimelineDirectory import TimelineDirectory
from PyFoam.RunDictionary.ParsedParameterFile import ParsedParameterFile

from scipy.integrate import odeint, solve_ivp
import numpy as np
```

Reading results and parameters from the case

Instead of hard-coding values we read them from the case

calcRocketHeight.py continued (start of function)

```
def calcHeights(caseName):
    rhoWater = ParsedParameterFile(path.join(caseName,
                                             "constant",
                                             "thermophysicalProperties.water")
                                   )["mixture"]["equationOfState"]["rho0"]
    bottleParameters = ParsedParameterFile(path.join(caseName, "constant", "brk"
                                                    <cont>bottleParameters"))

    t1 = TimelineDirectory(case=caseName,
                          dirName="postProcessing/swakExpression_acceleration")
    accel2 = t1["acceleration"]().getData()

    totalMassData = TimelineDirectory(case=caseName,
                                      dirName="postProcessing/swakExpression_totalMass")
    totalMass = totalMassData["totalMass"]().getData()
```

Solving without drag

`solve_ivp` means *Solve initial value problem*. It needs

- a function describing the derivative
- time range (up to 10s)
- the initial values (rocket is at rest)
- a vector with the times at which we want the solution

`calcHeights` continued in `calcRocketHeight.py`

```
grav = 9.81

def func(t, x):
    return [x[1],
           accel12[t]["acceleration_t=0_sum"].values[0]-grav]

times = np.linspace(0, 10, 100000)

sol = solve_ivp(func, [0, 10], [0, 0], t_eval=times)
```

Solving with drag

- Got the drag coefficient from the internet (not a very good reference)
- Geometry we read from the file that was used to set up the geometry

calcHeights continued in calcRocketHeight.py

```

densAir = 1.15
Cd = 0.82      # Long cylinder

massBottle = bottleParameters["bottleWeightGram"]/1000.

bottleRadius = bottleParameters["bottleRadiusCm"]/100.
bottleHeight = bottleParameters["bottleHeightCm"]/100.

area = bottleRadius*bottleRadius*3.1415
vol = area*bottleHeight

def funcDrag(t, x):
    mass = totalMass[t]["totalMass_t=0_max"].values[0]
    dragForce = 0.5*densAir*Cd*area*x[1]*abs(x[1])/mass
    velNoDrag, accelNoDrag = func(t,x)
    return [velNoDrag,
            accelNoDrag-dragForce]

solDrag = solve_ivp(funcDrag, [0, 10], [0, 0], t_eval=times)

return sol,solDrag

```


Creating a DataFrame from the solution

This will be handy later

calcRocketHeight.py continued

```
def solutionToDataFrame(sol):  
    from PyFoam.Wrappers.Pandas import PyFoamDataFrame  
  
    return PyFoamDataFrame(data={"h": sol.y[0], "vel": sol.y[1]},  
                           index=sol.t)
```

Use it as a stand-alone utility

- The if makes sure that this part is not used when the file is imported as a library

calcRocketHeight.py continued and end

```

if __name__ == "__main__":
    import sys
    if len(sys.argv) != 2:
        print("Script needs 1 argument: <case directory>")
        sys.exit(1)

    caseName = sys.argv[1]

    solNoDrag, solDrag = calcHeights(caseName)

    noDrag = solutionToDataFrame(solNoDrag)
    withDrag = solutionToDataFrame(solDrag)

    print("Max height without drag (solve_ivp) at t={}".format(noDrag.h.max(), noDrag.h.<brk>
        <cont> idxmax()))
    print("Max velocity without drag (solve_ivp) at t={,} h={}".format(noDrag.vel.max(), <brk>
        <cont> noDrag.vel.idxmax(), noDrag.h[noDrag.vel.idxmax()]])
    print("Max height with drag (solve_ivp) at t={,} h={}".format(withDrag.h.max(), withDrag.h.<brk>
        <cont> idxmax()))
    print("Max velocity with drag (solve_ivp) at t={,} h={}".format(withDrag.vel.max(), <brk>
        <cont> withDrag.vel.idxmax(), withDrag.h[withDrag.vel.idxmax()]])

```

Getting the results for one run

- The script now allows calculating the heights for one case
- But doing that for
 - 21 filling heights times
 - 9 initial pressures
- is tedious
 - and may lead to copy/paste errors

Running the script

```
> ./calcRocketHeight.py bottleRocket_h=10.0cm_p=7.00bar
Max height without drag (solve_ivp) 95.02943323016414 at t=4.486944869448695
Max velocity without drag (solve_ivp) 41.56696361178429 at t=0.1975019750197502 h<brk>
<cont>=5.043644903526841
Max height with drag (solve_ivp) 29.734145968350088 at t=2.1064210642106422
Max velocity with drag (solve_ivp) 38.26591408658745 at t=0.16920169201692017 h<brk>
<cont>=3.7235871308834385
```

Playing interactively with the data

-i gets us to the REPL

```
> ipython -i ./calcRocketHeight.py bottleRocket_h=10.0cm_p=7.00bar
```

REPL lets us use the variables of the script

```
%pylab
> drag=solutionToDataFrame(solDrag)
> noDrag=solutionToDataFrame(solNoDrag)
> drag.plot()
> noDrag.plot()
> drag.describe()
```

	h	vel
count	100000.000000	100000.000000
mean	-8.375094	-7.467186
std	32.699416	12.621959
min	-74.675001	-15.331439
25%	-36.360768	-15.303381
50%	0.491664	-14.597914
75%	22.096334	-3.776312
max	29.734146	38.265914
integral	-83.748040	-74.671843
valid length	10.000000	10.000000
weighted average	-8.374804	-7.467184

Comparing drag and no drag solutions

No drag

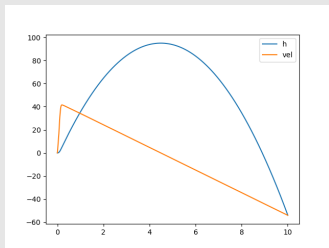


Figure: Gravity's rainbow

With drag

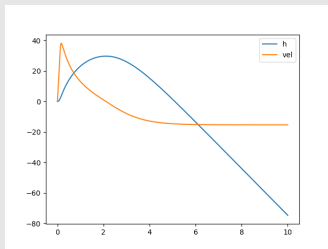


Figure: Drag limits maximum and minimum velocity

Calculating the height

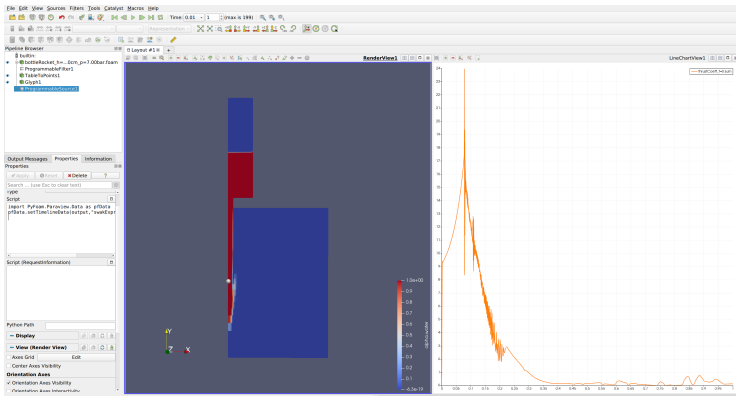
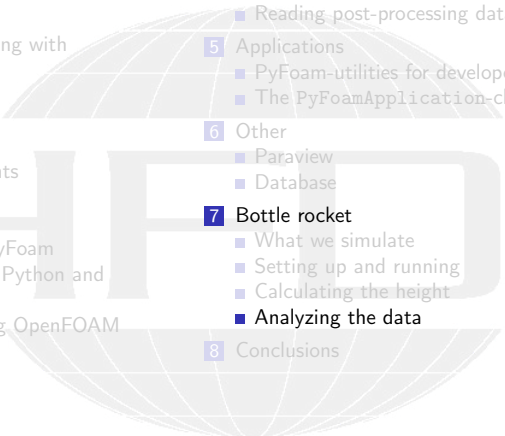
Using `rocketWithScripts.pvsm`

Figure: On the left there is a sphere at the location of the maximum velocity. On the right a timeline loaded from the case

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Collecting and analyzing the data

Write a script

- Collects the pickledData from all runs in the directory into a SQLite database
- Loops through the runs
 - runs calcHeight on the case
 - adds heights from that to the database
 - uses uniqueid to add the data to the correct run

Then we work interactively

- read the database with the utility `pyFoamDumpRunDatabaseToCSV.py`
- interactively play with the data
 - use the pivot functionality of pandas to arrange the data in a table

If you're more comfortable with that you can read the CSV into Excel

Setup and Database creation

It is easier to use the application class than constructing the database from scratch

Beginning of collectData.py

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

from glob import glob
from os import path
from pickle import Unpickler
from pprint import pprint

from PyFoam.Applications.AddCaseDataToDatabase import AddCaseDataToDatabase
from PyFoam.Basics.RunDatabase import RunDatabase

from calcRocketHeight import calcHeights, solutionToDataFrame

pattern = "bottleRocket_h=*bar"
dbName = "bottleRocketData.db"
pickleFile = "PyFoamRunner.compressibleInterFoam.analyzed/pickledData"

cases = glob(pattern)

print("Adding basics from {} cases".format(len(cases)))
AddCaseDataToDatabase(args=[dbName] + [path.join(c,pickleFile) for c in cases] +
["--create"])

db = RunDatabase(dbName)
```

Loop over all the runs

- Our previous script is used as a library and the results are added to the database

collectData.py continued and end

```

for c in cases:
    print(f"Processing {c}")
    data = Unpickler(open(path.join(c,pickleFile), "rb")).load()
    runId = data["uniqueid"]
    pressure = data["parameters"]["initPressure"]
    height = data["parameters"]["initHeight"]
    print(f"ID: {runId} p={pressure} h={height}")

    solNoDrag, solDrag = calcHeights(c)
    noDrag = solutionToDataFrame(solNoDrag)
    withDrag = solutionToDataFrame(solDrag)

    evaluations = {"noDrag": { "hMax": noDrag.h.max(), "hMaxTime": noDrag.h.idxmax(),
                              "velMax": noDrag.vel.max(), "velMaxTime": noDrag.vel.idxmax()
                              <cont> () ,
                              "hVelMax": noDrag.h[noDrag.vel.idxmax()]},
                  "withDrag": { "hMax": withDrag.h.max(), "hMaxTime": withDrag.h.idxmax()
                              <cont> ,
                              "velMax": withDrag.vel.max(), "velMaxTime": withDrag.vel.
                              <cont> idxmax() ,
                              "hVelMax": withDrag.h[withDrag.vel.idxmax()]}}

    pprint(evaluations)
    db.modify(runId,{"evaluations": evaluations})
  
```

Analyzing the data in the database

- Now we can play with the data
 - For details on `pivot` check the `pandas` documentation

Getting the data into a REPL

```
> pyFoamDumpRunDatabaseToCSV.py bottleRocketData.db bottleRocketData.csv --interactive -<brk>  
<cont>after
```

On the REPL

```
%pylab  
data=self.getData()["dump"]  
piv=data.pivot("parameters//initHeight","parameters//initPressure")  
piv["evaluations//withDrag//hMax"].plot(marker="o")  
plot(piv["evaluations//withDrag//hMax"].idxmax(),piv["evaluations//withDrag//hMax"].max(),'<brk>  
<cont>r',linewidth=4)
```

The resulting plot

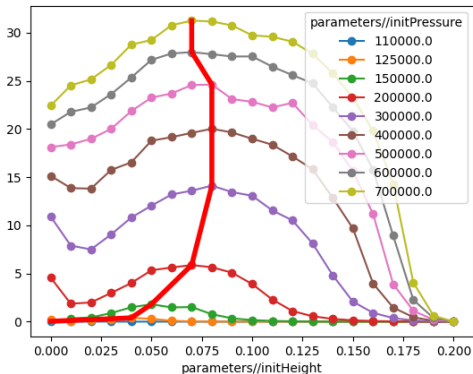


Figure: Maximum height as a function of the two parameters (red line is the maximum for each pressure)

Analysis of the results

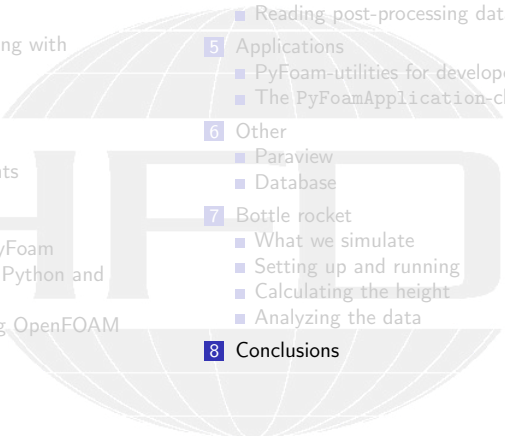
- Heights of 20-30 m seem reasonable
 - In reality the rocket has stability problems and starts going sideways at 10+ m
- Recommendation of DLR to "fill one third" seem reasonable
- High pressures show diminishing results
- For low pressures rockets without water fly almost as high as filled rockets
 - But with water it is more fun
 - *Fun* is hard to model (not just in OpenFOAM)

Critique of the model

This simulation will not revolutionize *bottle rocket science*

- geometry needs improvement
- turbulence model is very crude
- `g` in `compressibleInterFoam` is constant
 - doesn't take into account that this is an accelerated reference system
 - solver would have to be modified
- drag force could be calculated better
 - we have a flow solver

Outline

- 
- 1 Introduction
 - This presentation
 - Who is this?
 - What are we working with
 - Before we start
 - 2 Python Ecosystem
 - Python versions
 - Packages
 - Virtual environments
 - 3 The workhorses
 - Library structure
 - File handling by PyFoam
 - Data structures in Python and OpenFOAM
 - Parsing and writing OpenFOAM dictionaries
 - 4 Reading data
 - Recycling PyFoam data
 - Reading post-processing data
 - 5 Applications
 - PyFoam-utilities for developers
 - The PyFoamApplication-class
 - 6 Other
 - Paraview
 - Database
 - 7 Bottle rocket
 - What we simulate
 - Setting up and running
 - Calculating the height
 - Analyzing the data
 - 8 Conclusions

Further presentations

Other presentations about PyFoam that are not specifically about programming it but might help are

- about `pyFoamPrepareCase.py` (which is also something like programming)
 - Uses something called *templates*
 - See "Automatic case setup with `pyFoamPrepareCase`" from the Ann Arbor Workshop 2015
 - an updated version was given at the Shanghai Workshop 2018
- Writing and handling data is explained in
 - "PyFoam for the lazy" from Guimares Workshop in 2016

Contribute

- The official source repository of PyFoam is at <https://sourceforge.net/p/openfoam-extend/PyFoam/ci/default/tree/>
- To clone it use Mercurial

```
> hg clone http://hg.code.sf.net/p/openfoam-extend/PyFoam PyFoam
```

Mercurial is similar to git (but with a better user-interface)

- Pull requests are most welcome
- An introduction for contributors comes with the sources: <https://sourceforge.net/p/openfoam-extend/PyFoam/ci/default/tree/DeveloperNotes.md>

Goodbye to you



Thanks for listening
Questions?

License of this presentation

This document is licensed under the *Creative Commons Attribution-ShareAlike 3.0 Unported* License (for the full text of the license see <https://creativecommons.org/licenses/by-sa/3.0/legalcode>). As long as the terms of the license are met any use of this document is fine (commercial use is explicitly encouraged). Authors of this document are:

Bernhard F.W. Gschaider original author and responsible for the strange English grammar. Contact him for a copy of the sources if you want to extend/improve/use this presentation