Introduction

Table of contents (../toc.ipynb)

Motivation for this lecture series

- Basically, almost all jobs in engineering require (or will require) programming skills due to digital transformation (https://en.wikipedia.org/wiki/Digital_transformation).
- Digital transformation requires skill in digital products (software), technologies (software and system engineering), safety and security, data analysis, and the like.

Improve your skills

If **you** want to be prepared for a digital job, e.g. if you want to apply for a job in autonomous driving, your chances are much are higher if you have expertise in

- C, C++, Python
- Clean Code, Version control (Git), Continuous integration (Jenkins, Travis)
- Software quality assurance (software testing)
- Distributed software development (Inner Source, Open Source)
- Cloud computing (Azure)
- IT safety and security
- Image Processing
- Machine learning
- *Linear Algebra*
- *Statistics*
- *Physics*
- *Vehicle Science*

But also check if your prospective employer is prepared

However, you should also **test the prospective employer** with famous Joel Test (https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/). Joel Spolsky (https://www.joelonsoftware.com/about-me/) was for instance CEO of stackoverflow until 2019. This test is a 20 years old test to determine how well software teams work.

The test is simple, just ask the company:

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

"A score of 12 is perfect, 11 is tolerable, but 10 or lower and you've got serious problems." [Spolsky2000] (../references.bib)

Outline of this lecture

In this lesson you will learn

- Why it is worth to learn Python
- How to create and manage Python environments
- Some Python syntax snippets.

You can find the course material in notebook style on github.com (https://github.com/StephanRhode/py-algorithms-4-automotive-engineering)

Why Python?

- Python is one of the most popular programming languages.
- Python was designed as teaching and scripting language in 1990s by Guido van Rossum wikipedia (https://en.wikipedia.org/wiki/Python_(programming_language)), hence it is easy to read and easy to learn.

- Python becomes more and more popular due to lack of "real" programmers in industry.

Here is the TIOBE index (https://www.tiobe.com/tiobe-index/) from December 2019 of popular programming languages. The index is based on number of skilled engineers world-wide, courses and third party vendors.

Programming Language	Ratings	Change
Java	17.253%	+1.32%
C	16.086%	+1.80%
Python	10.308%	+1.93%
C++	6.196%	-1.37%
C#	4.801%	+1.35%
Visual Basic .NET	4.743%	-2.38%
JavaScript	2.090%	-0.97%
PHP	2.048%	-0.39%
SQL	1.843%	-0.34%
wift	1.490%	+0.27%
Ruby	1.314%	+0.21%
Delphi/Object Pascal	1.280%	-0.12%
Objective-C	1.204%	-0.27%
Assembly language	1.067%	-0.30%
Go	0.995%	-0.19%
R	0.995%	-0.12%
MATLAB	0.986%	-0.30%

- Python is easy to learn, to read and to maintain.
- Python is interpreted. No need to compile the program and you can interact with the Python interpreter which is great for pre-development.
- Python supports scripting, functional programming and object orientated programming.
- Python is portable to almost any computer platform.

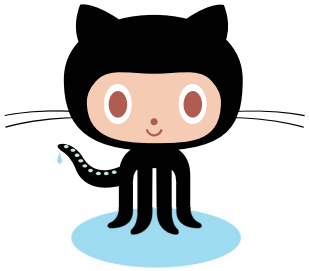
- Python lives from contributions of a large community and provides many great libraries.
- Recent popular methods like deep learning, speech recognition and the like are usually programmed in Python.
- If you want to process data on a computer, chances are very high that someone created a Python library for this.

Numpy	Scipy	Matplotlib	Scikit-learn	IPython
				

KUDOS to Jake VanderPlas who wrote the beautiful Python introduction *A Whirlwind Tour of Python* [VanderPlas2016] (../references.bib), which is under CC0 license and hence I will re-use and build-on it in the latter.

- *NumPy (https://numpy.org/) provides efficient storage and computation for multi-dimensional data arrays.*
- *SciPy (https://scipy.org/) contains a wide array of numerical tools such as numerical integration and interpolation.*
- *Matplotlib (https://matplotlib.org/) provides a useful interface for creation of publication-quality plots and figures.*
- *Scikit-Learn (https://scikit-learn.org/) provides a uniform toolkit for applying common machine learning algorithms to data.*
- *Jupyter (https://jupyter.org/) provides an enhanced terminal and an interactive notebook environment that is useful for exploratory analysis, as well as creation of interactive, executable documents. For example, the manuscript for this report was composed entirely in Jupyter notebooks.*

- All of these packages are **open source!** and available on github (https://github.com/).



The Zen of Python

The philosophy of Python can be read with `import this` command. It is quite the counter part to Matlab's `why` :)

In [1]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```