# Python Introduction and Jupyter

Python's design goals are easy accessibility, easy development processes, and generality. Unlike other programming languages used in science and engineering, like C++ or Fortran, it is not a compiled language. The processes of compilation and linking is completely omitted from the development cycle when running pure Python. Python is an interpreted language. The interpreter is also called Python and can be used on almost any operating system.

There are two ways to use Python. In a shell, you can start the interpreter by invoking the python interpreter by simply typing "python" and hitting return. On Windows, the interpreter is often in "C:\pythonxx", where "xx" is the version number. The easiest solution to get python for Windows systems is to install "anaconda", which includes the "conda" package manager for python. To quit your session, press ctrl+d. This mode is usually only good for quick testing or small calculations, since nothing is saved and to reuse code you'd have to program functions or classes.
The second way is to write a script. This is simple text in a textfile, which can later be executed by invoking the Python interpreter and providing the filename as an argument. The most comfortable way, and this is highly recommended for developing, is using an integrated development environmet (IDE). These programs offer rich features and comfortable shortcuts, as well as good error recognition and solution suggestions. Big names for Python are Spyder and pyCharm.

## Jupyter

In this course, you won't need to install Python on your system. Instead, we will work with Jupyter notebooks, such as the one you're looking at right now. Jupyter notebooks are successors to IPython (interactive Python) and were initially developed to support JUlia, PYThon, and R, hence the name. Since then, support for a multitude of kernels was added and it's difficult to find a widely adopted language that isn't supported. Jupyter notebooks aim to provide an easy-to-use interface for programming tasks in a presentable fashion. The notebook is subdivided into cells, of which you will find two kinds: markdown cells, such as this one, or code cells, such as the one below:

In [1]:

```
print("This is a code cell")
```

```
This is a code cell
```

You can change the type of a cell under "Cell" -> "Type". Markdown cells allow, well, usage of markdown, which enables rich text formatting options. See for example this markdown cheat sheet: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet) To edit the text of a markdown cell, double-click on it. To submit your changes and also to execute code in a code cell click on it, then hit "shift" + "enter". Try this with the following cell.

In [3]:

```
print(5*100/5.9)
```

```
84.7457627118644
```

The output is shown directly below the code cell.

You can also directly input $\LaTeX$-code (mainly math-related code) by either enclosing it in `$dollar signs\$` or starting an environment like `\begin{equation} \end{equation}`, just like you're probably used to.

That's pretty much all you need to know to work with the notebooks. For more info and a guided tour, click on "Help", then "User Interface Tour". For shortcuts, see "Help", then "Keyboard Shortcuts".