Zurich University of Applied Sciences

School of Engineering

P01 – Introduction to programming

1. On-site exercises (on-site week 1)

Take part in the lab group according to your personal class schedule. You will get further information regarding the nature and assessment criteria for the IT_PROG labs.

Week 1 will consist of three exercises without a computer needed. Note down what you learned from those three exercises.

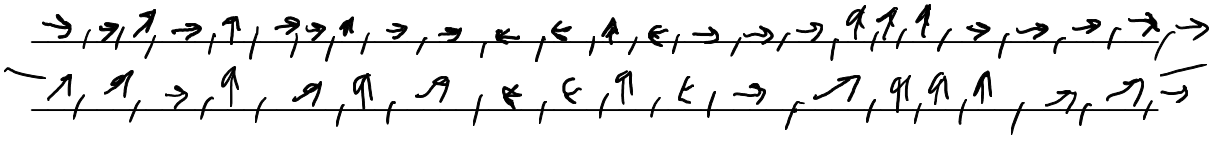
Take home message pair exercise in *drawing with unidirectional communication*:

präzise anweisungen, keine kontext, alles erklären, koordinaten system, präzise

Take home message classroom exercise in *everyday life decomposition*:

-aufstehen, -klassenzimmer verlassen,-gehe zur treppe,-treppe runter,-gehe zur mensa,-karte checken wenn geld dann kaffe, sonst aufladen dann kaffee,-kaffee asuwählen,-becher auswählen, -bezahlen, -kaffee nehmen,-trinken

Take home message individual exercise *jump'n'run*:



Work on the following exercises 2 and 3 individually and at home.

2. Reading assignment (at home)

Read the extract of Hans-Peter Hutter's text *"PPRG – Prozedurale Programmierung"* in the file *"PPRG_ComputerBasics.pdf"* (in German, 12 pages). Make some notes and/or highlight text passages that are important for you.

Room for notes:

➔ *Prepare to answer some of your advisor's questions regarding this text in week 2.*

3. Installing Anaconda Python (at home, if not already done)

3.1 Background

Installing Python 3.6 from https://www.python.org/download/ has many important advantages (it's the official, free, open-source reference installation not bound to any company, it comes with the light-weight IDLE development environment, it is available for all platforms, it is always up to date etc.).

It has also shown some disadvantages in the context of IT_PROG:

- Under MacOS, IDLE is unstable and sometimes crashes, e.g. when hitting the **^**-Key
- For a non-programmer, it is difficult to install additional modules on Windows and MacOS (mainly because the Python package management system PIP expects to find a fully prepared developer's machine with installed C compilers and the like); this hinders executing / working on some of the more interesting examples and exercises

Thus, we invite everybody to replace the standard Python installation (with IDLE) to the more advanced scientific Python distribution "Anaconda". Anaconda offers:

- Easy installers for Windows, MacOS & Linux
- All necessary libraries already included
- A slightly more sophisticated development environment called "Spyder"

We will use Spyder as our standard development environment and later make use of some of the libraries that come with Anaconda (such as **`numpy`**).

3.2 Removing standard Python

First, you have to remove your self-installed Python environment (if you have installed any; you don't have to remove anything that came pre-installed with your system).

On Windows:

- Go to the Windows "Control Panel" → "Uninstall a program" ("Systemsteuerung" → "Programm deinstallieren")
- Search the list for all Names that include "Python 2.7" and remove them using the "uninstall" menu item from the pop up menu (right click or button above the list)
  → normally everything should be fine already; you might also work through the next steps to be completely safe
- Remove the "c:\python27\" entry (might look slightly different on your machine) from the PATH environment variable:
  - Hit WIN+Pause (WIN is the Windows key on your keyboard) → you see the Control Panel's "System" page
  - Click on "Advanced system settings" ("Erweiterte Systemeinstellungen")
  - Hit the button labeled „Environment Variables" ("Umgebungsvariablen")
  - Look for the entry with Variable="Path" in the list of system variables (lower half of the window)
  - Edit it and remove any part (between two "**;**" or at the end of the line) that refers to your original Python directory
  - Close all dialogs with "ok"
- Delete your original Python directory (normally "C:\Pythons27") using the windows explorer (you may have to restart your system before)

On MacOS:

- No preparation needed. Anaconda will update the PATH variable in ~/.profile and prepend the ~/anaconda directory, hence Python will automatically be the Anaconda version.

3.3 Downloading Anaconda

- Download the correct version for your system from here: http://continuum.io/downloads#all
  - Be sure to use the Version for Python 3.6, not Python 2.7
  - Take the 64-bit variant if you have a 64 bit system
  - For Mac: MacOS X – 64bit python 3.6 graphical installer (ca. 300 MB)

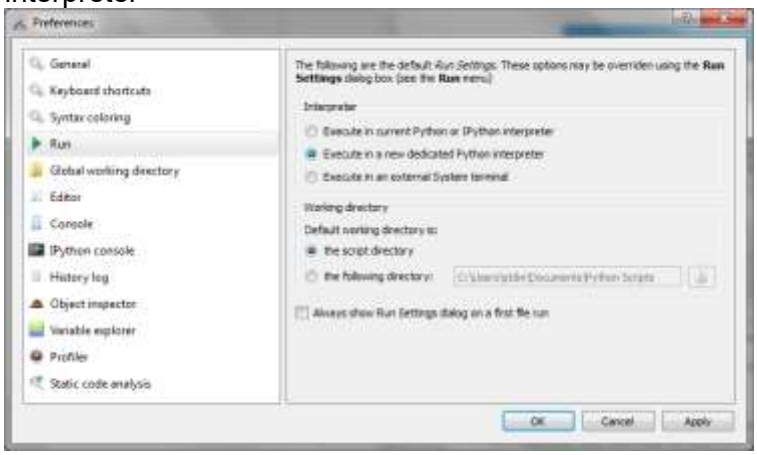
3.4 Installing Anaconda

- Follow the instructions in the setup process
  - You can safely accept all standard choices
  - On Windows: If you are asked if you want to install for everyone or just yourself → choose "for everyone"
  - On Windows: If you are asked if Anaconda should be your standard Python 2.7 environment → conform with "yes"
  - On MacOS X: if the installer says something like 'cannot install', choose 'just install for me'.

3.5 Configuring Spyder

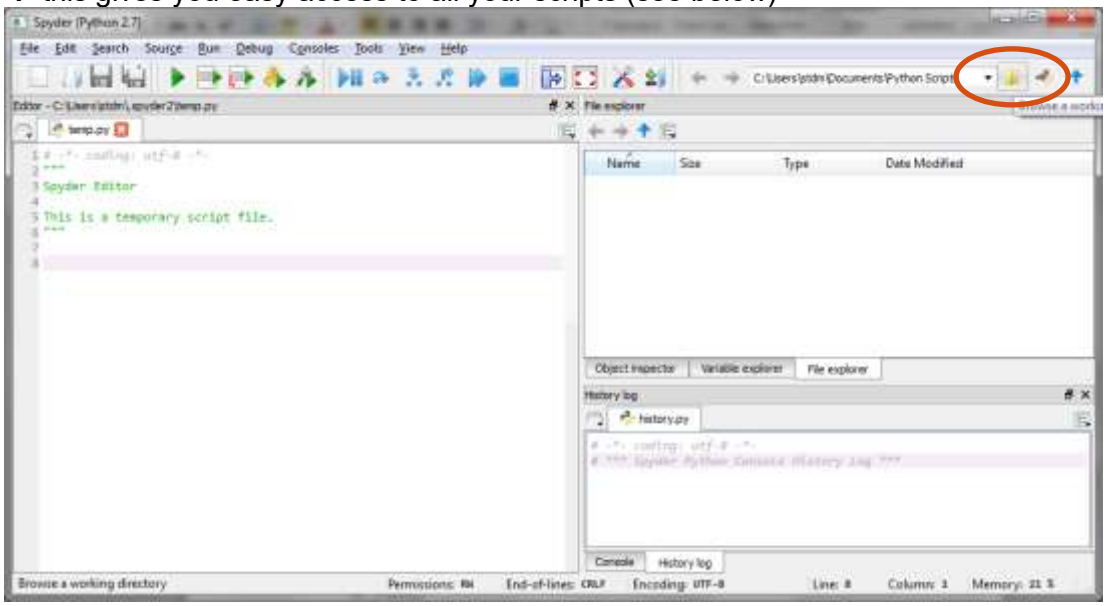
The following settings make Spyder look tidier:

- Start Spyder (on Mac: go to your home/anaconda and click on Launcher, then start Spyder; if the launchen doesn't work, just type `spyder` in a console)
- Go to "Tools" → "Preferences"
  - On the "Run" page change "Interpreter" to "Execute in a new dedicated Python interpreter"

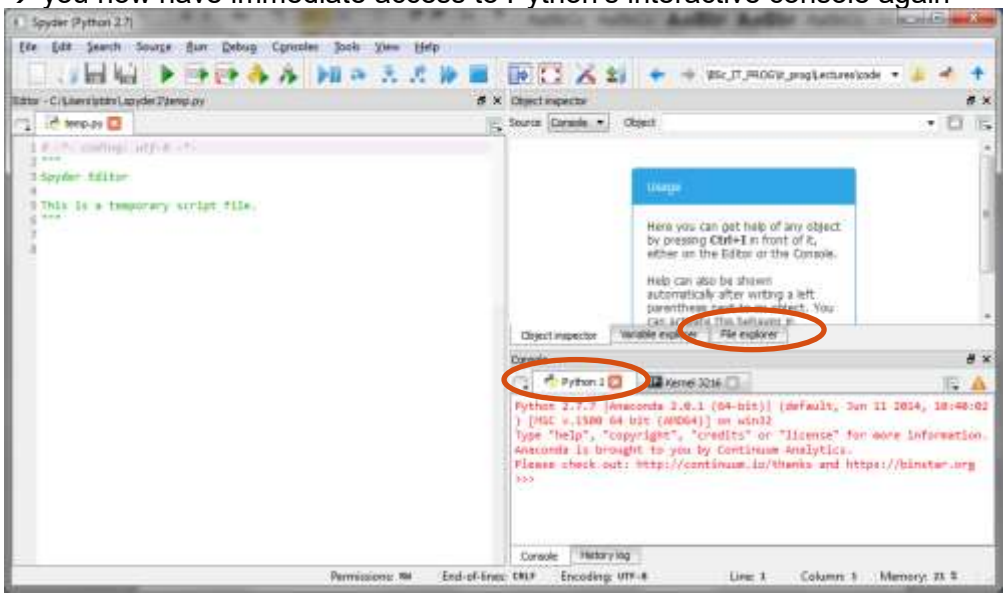


  - Close the Preferences with "ok"
- Go to "View" → "Panes" and disable the "IPython console" and "History log"
- Click the "Browse a working directory" Button at the right end of the menu bar and choose the folder where you store your scripts

→ this gives you easy access to all your scripts (see below)



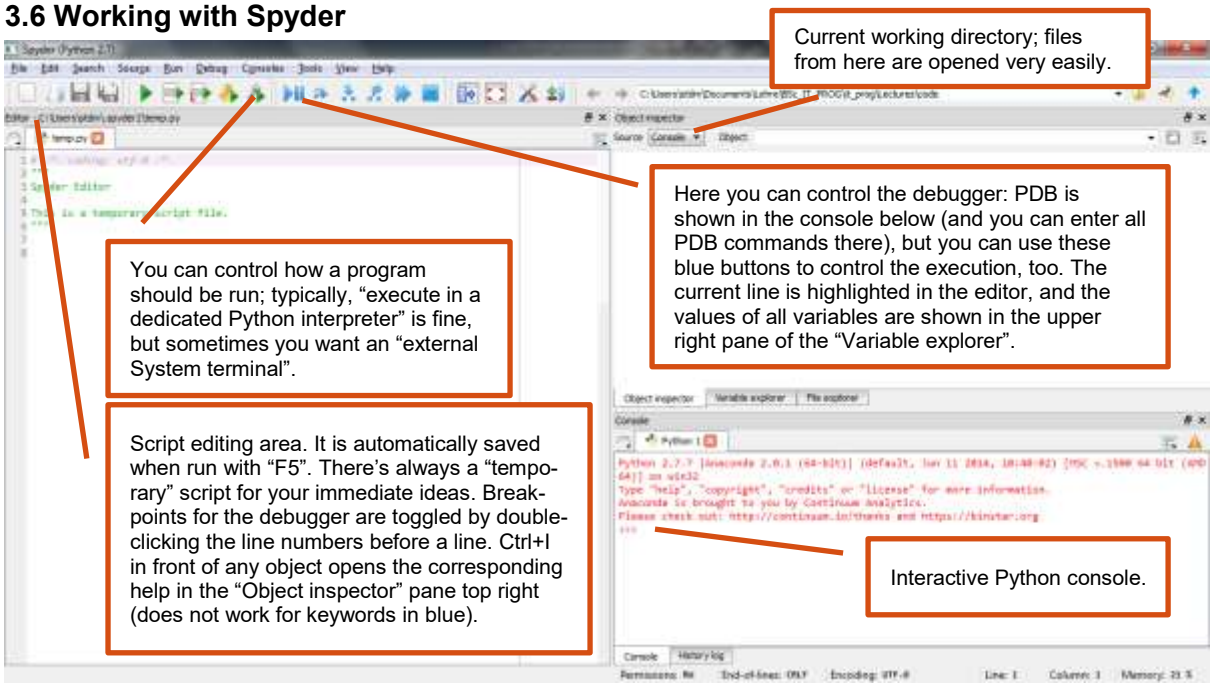
- In the lower middle part of the window, change from the “History log” tab to the “Console”, and therein to the “Python 1” tab
  → you now have immediate access to Python’s interactive console again



- In middle of the right half of the window, switch to the “File explorer” tab
  → you now see a list of all the scripts in the current working directory and can easily open them

Close Spyder and start it again: All your settings should be stored and recovered.

3.6 Working with Spyder



- Go to the interactive console tab (bottom right) and use it as a calculator
- Got the temp.py in the code editor (left) and write your individual "hello world" script. Run it, modify it, re-run it.

4. First steps with Python (on-site week 2)

Solve as many of the following small programming exercises as you can. Feel free to form study pairs (this is also true for the remainder of this semester: You may solve programming lab exercises pairwise).

Each exercise should ultimately be solved by writing a small script, saved to an individual file.

4.1 Execute a python script

Open an editor. Write following rows in the text file:

```
a = 144
result = a / 2
print(result)
```

Save the file with the name "`division.py`" somewhere on your file system.

Open a command shell. Go to the folder, where you have saved your python script. To run your new python script execute following command:

```
python division.py
```

4.2 Braking Distance – Compute the distance it takes to stop a car
A car driver, driving at velocity *v0*, suddenly puts on the brake. What braking distance *d* is needed to stop the car?

One can derive, from basic physics, that

- *d = 0.5*v0^2 / (mu*g)*

*Hint – In Python ^ (power of) is written as ***

Develop a program for computing *d* using the above formula when the initial car velocity *v0* and the friction coefficient *mu* are provided via the *input* function (see hint below). The gravitational acceleration constant *g* is 9.81.

Run the program for two cases:

- *v0* = 120 km/h and
- *v0* = 50 km/h, both with *mu* = 0.3 (*mu* is dimensionless).

Remember to convert the velocity from km/h to m/s before inserting the value in the formula!

Hint – How to input a value in Python:

```
#reads a floating point number from keyboard input and stores it in a variable named "v0":

v0 = float(input("v0 [km/h]? "))
```

4.3 Celsius to Fahrenheit – Convert degrees Celsius to degrees Fahrenheit
Write a program that reads a temperature in Celsius from the console and converts it into degrees Fahrenheit.

There's a simple arithmetic formula for the relationship of these two scales – just google "fahrenheit to celsius" and find it on the web (a very common task for every programmer). Don't forget to prove your web source by at least double checking the formula with a second source.

4.4 Distance between to aircrafts
Create a program that reads the 2D coordinates (x, y) from two planes from the console and computes their distance. Ask for x and y individually.

You can use the standard Euclidean distance as a distance measure: It is the square root of the sum of the squared differences between the respective x- and y coordinates.

Hint – In Python, you can use a build-in function to compute the square root:

```
#compute the square root of a quantity (number) a and store it in a variable square_root_of_a:
import math
square_root_of_a = math.sqrt(a)
```

4.4 Litres to Kilograms – Another conversion task

Write a small program that asks the quantity (in litres l) and density (:= mass per volume in kg/l) of some fluid from the user via the console and outputs the weight of it in kilograms.

The code and formulae is going to be rather simple; can you already guess it given the units of the two inputs? Otherwise, look it up on the web.

4.5 Parrot – Echo the input

Create a program that behaves like a well-trained parrot: it just repeats any input.

Again, the code itself will be quite simple (more so if you consider that for this automated parrot it is ok to do the trick only once and then quit). Use the time therefore to think for yourself of what steps this program can be composed and what python constructs you already know to realize them. Ignore all possible programs that need more Python knowledge than you yourself (currently) have under command.

4.6 Plot Trigonometry – Creating real graphical output (optional)

This exercise is different from the previous ones in that you will get a complete script and are asked to make it run and play with it in order to understand it.

You are not supposed to know how to do this already. In fact, you don't. When programming, you will often face such new (and somewhat scary) situations, but with time you learn that you have the ability to work through them. This is just a training occasion for such situations. You are *not* obliged to produce any outcome here; but we estimate that with 1-2 hours investment of time you would figure out the script completely. Let's give it a try and see what you can achieve in 15 minutes?

There are two challenges here:
1. Make it run.

First, here's the script:

```
import numpy as np
import pylab as pl

n = 256
X = np.linspace(-np.pi, np.pi, n, endpoint=True)
Ysin = np.sin(2 * X)
Ycos = np.cos(2 * X)

#pl.axes([0.025, 0.025, 0.95, 0.95])

pl.plot(X, Ysin, color='blue', alpha=1.00)
pl.plot(X, Ycos, color='green', alpha=1.00)
```

```
pl.xlim([-np.pi, np.pi])
pl.ylim([-2.5, 2.5])
pl.show()
```

If you are using the Anaconda Python Distribution, you are good to go.

If you have another python installation, to make it run, you maybe have to install two "libraries": `numpy` and `pylab` (part of something called `matplotlib`); the most convenient way to do this is via `pip`:

- install `pip`: http://pip.readthedocs.org/en/latest/installing.html#install-pip
- install `numpy`: open a console and execute "`pip install numpy`"
- install `matplotlib`: open a console and execute "`pip install matplotlib`"

If this is the case, 15 minutes probably won't suffice. If you are fascinated by the task, go on anyway.

2. Understand what's going on.

Ideas for finding out what the code does are:

- Manipulate individual lines and observe what happens if you re-run the script (e.g., change the values of "parameters")
- Render certain lines ineffective by commenting them out (:= putting a "`#`" at the start of a line so the code will be regarded as a comment by Python) and observe what they contribute to the overall script
- Do a web search for certain key words that appear in the code
- Use `pdb` to debug the script step by step
- Read on plotting with Python: http://www.ucs.cam.ac.uk/docs/course-notes/unix-courses/pythontopics/graphs.pdf