

Using Python in real world applications

Part 2

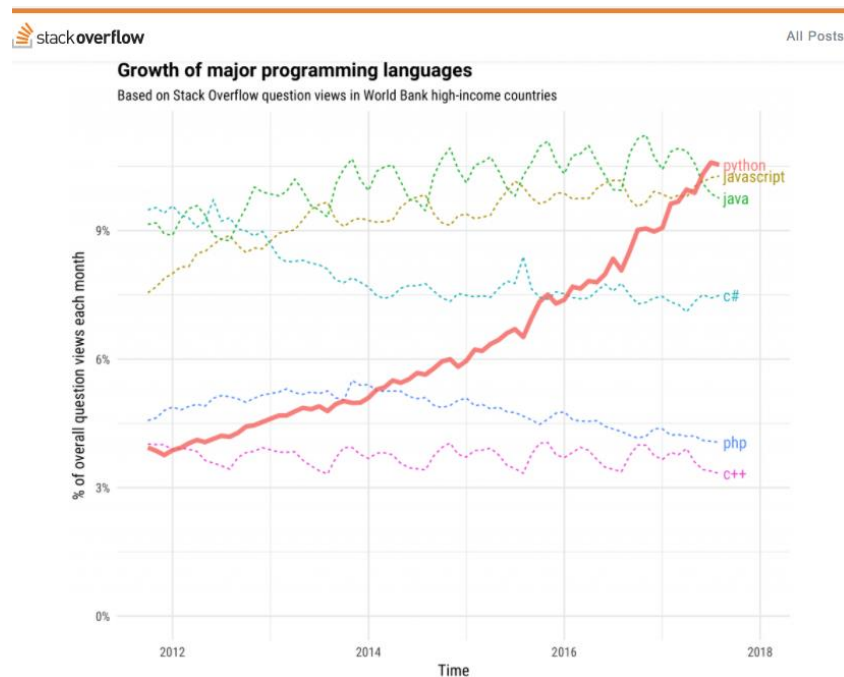
May 8th, 2019

1 Introduction to real world application

This is the introduction to applying Python in real world applications and in your everyday workflows. Most of mundane tasks could and should be automated, and it's up to you to get it done. Examples of such mundane tasks are:

- Import and export procedures
- Doing any form of repetitive work (generating reports, do simple calculations)
- Digitally signing hundreds of PDF documents (avoid tendonitis) like neighbor warnings etc..

Furthermore, Python is one of the world's fastest growing major programming language. Python can be used for web development, data science — including machine learning, data analysis, and data visualization, as well as general scripting.



Python is also getting embedded as an option inside lots of different softwares. This enables Python users to apply their skills doing more advanced tasks as part of more complex work flows. Examples of such softwares where Python can be applied:

- QGIS
- Dynamo for Revit
- FreeCAD

2 Getting started – running Python locally on your own machine

In part 1, you learned how to run simple python commands inside the jupyter notebook environment. The next step will be to run this on your own machine.

2.1 Installing Python

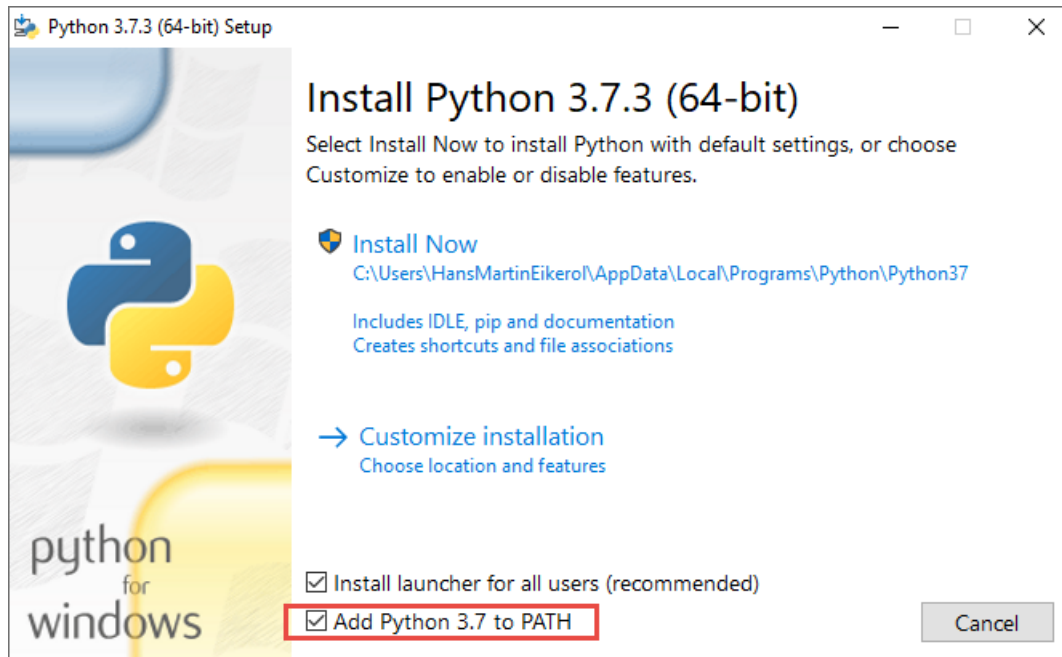
Default Windows machines does not ship with Python installed. In order to get Python up and running for stand-alone scripts, you must insall Python on your machine. Note that there are two major versions of Python: version 2.7.16 and 3.7.3. Both can be downloaded and installed from this link:

<https://www.python.org/downloads/>

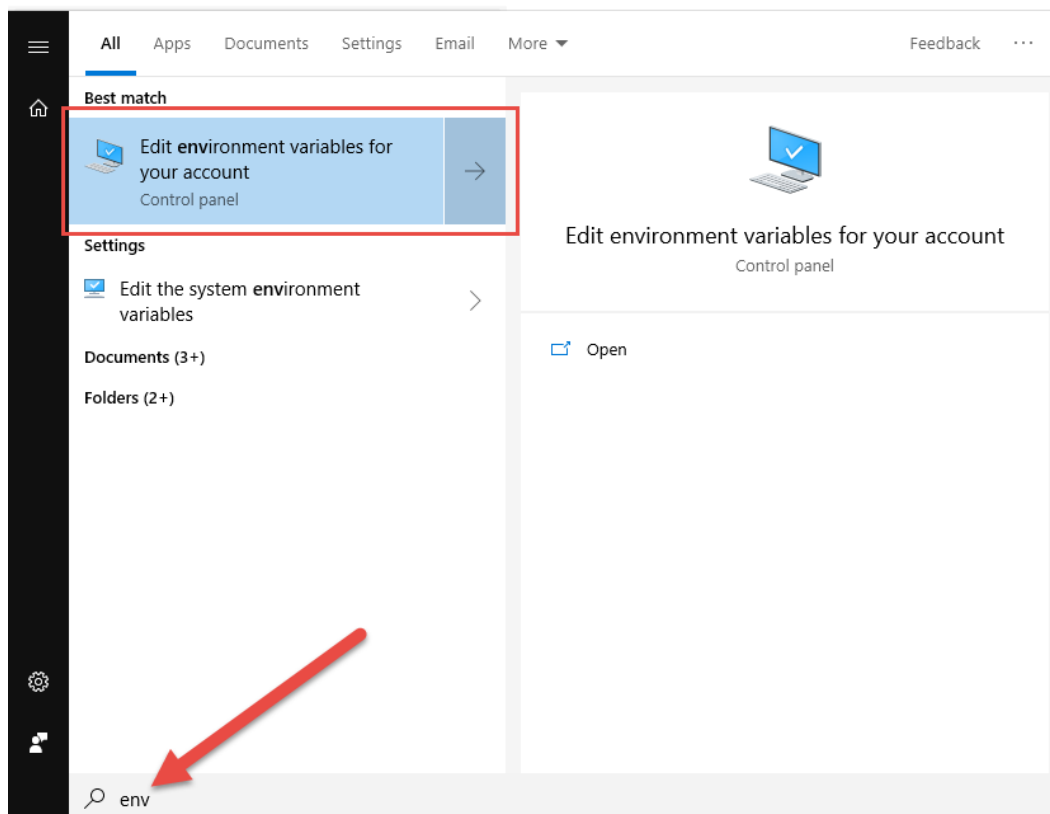
You must choose which of the versions of Python you will run. The elder version (2.7) is the legacy version and will not get any major updates. The 3.X-version is the future of Python. However, some older (and really good) libraries are not compatible with version 3.X. If you need to use any of them, you will need to run version 2.7 of Python.

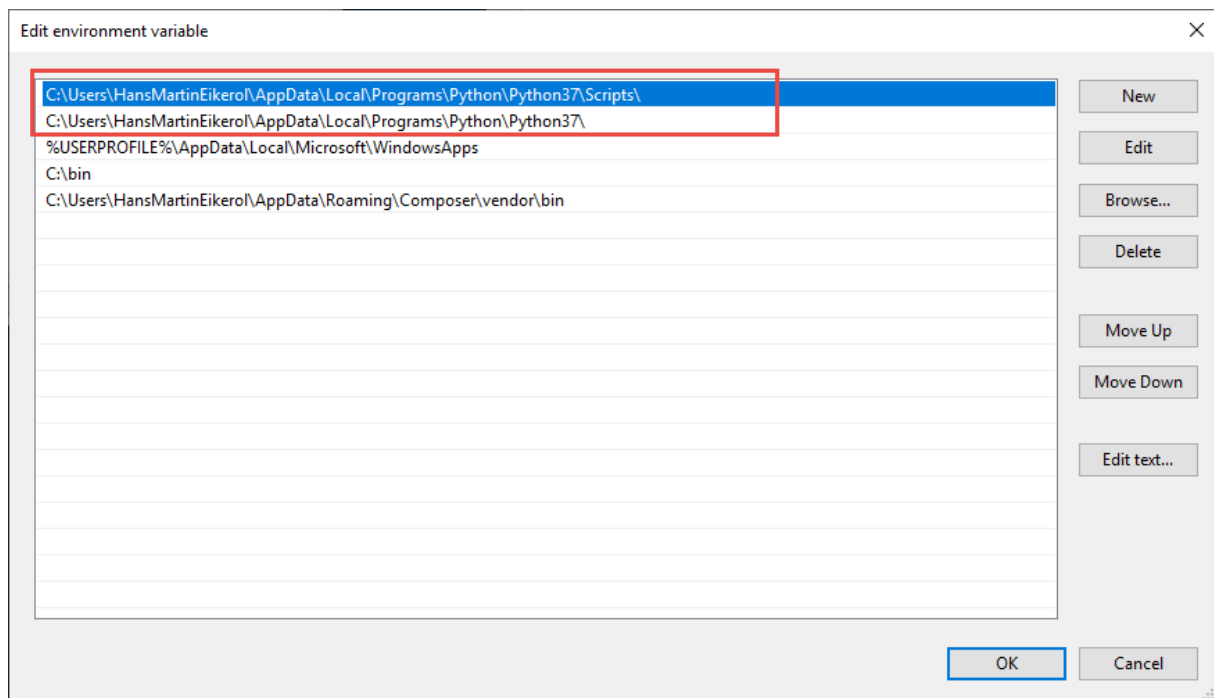
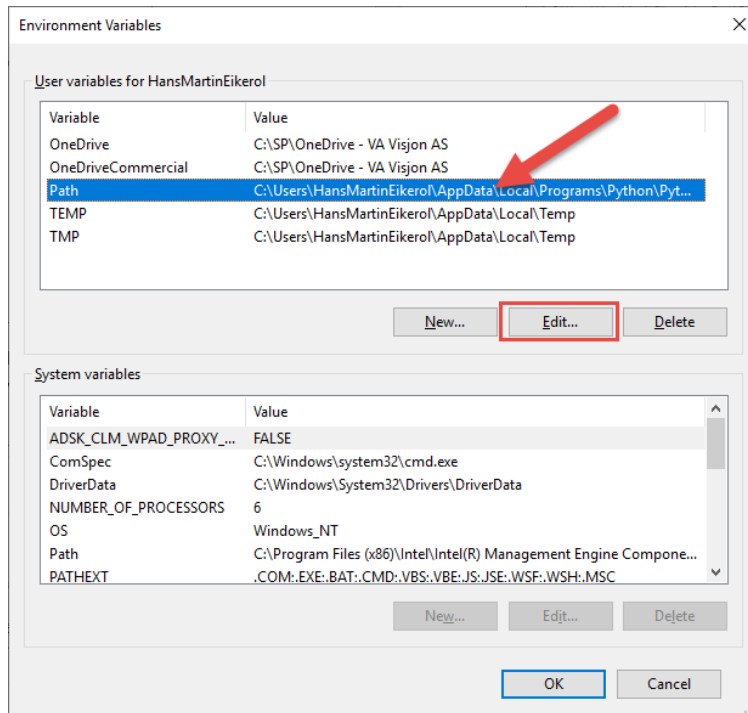
Files					
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		2ee10f25e3d1b14215d56c3882486fcf	22973527	SIG
XZ compressed source tarball	Source release		93df27aec0cd18d6d42173e601ffbbfd	17108364	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	5a95572715e0d600de28d6232c656954	34479513	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	4ca0e30f48be690bfe80111daee9509a	27839889	SIG
Windows help file	Windows		7740b11d249bca16364f4a45b40c5676	8090273	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	854ac011983b4c799379a3baa3a040ec	7018568	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a2b79563476e9aa47f11899a53349383	26190920	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	047d19d2569c963b8253a9b2e52395ef	1362888	SIG
Windows x86 embeddable zip file	Windows		70df01e7b0c1b7042aabb5a3c1e2fbd5	6526486	SIG
Windows x86 executable installer	Windows		ebf1644cdc1eeebacc92afa949cfc01	25424128	SIG
Windows x86 web-based installer	Windows		d3944e218a45d982f0abcd93b151273a	1324632	SIG

Download the installer and run it.



Be sure to check the “Add Python 3.7 to PATH” when you run the installer. This will enable you to run Python from command lines anywhere in your folder structure. This is what it does to your “System environment variables”:





Whether you are installing the 2.7 or 3.7-version of Python, take notice of where it installs to. This will be useful later. To help you out, these are the default locations:

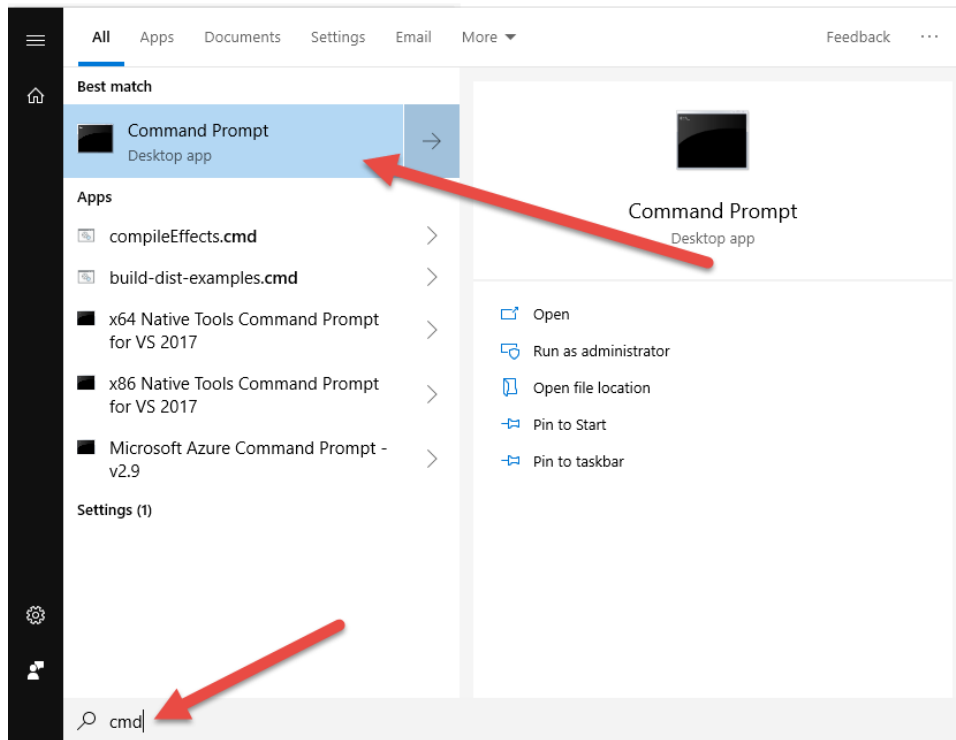
Python 2.7:
C:\Python27

Python 3.7:
C:\Users\<Your username>\AppData\Local\Programs\Python\Python37

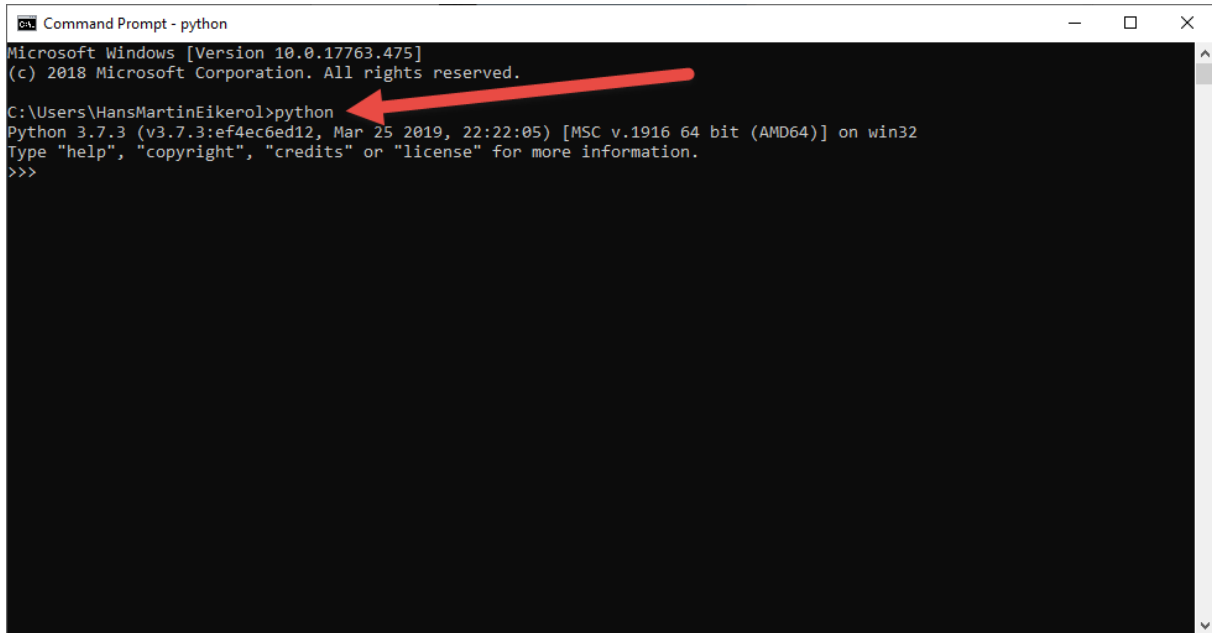
2.2 Checking if Python installed correctly

This is not really needed. Python did install correctly. However, if you managed to set the PATH environment variable properly, you should now be able to run **python** as a command in windows command prompt and Windows PowerShell, not having to rely on only running python code inside the python shell client. More about that in next section.

Try out running python in your command prompt. Open the command prompt by entering **cmd** in the windows search bar:

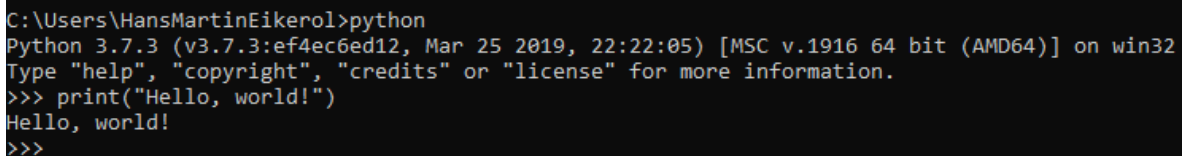


Start it. Type in the command **python**, and press enter. You should now see the python shell getting active. This is indicated by the three >>> at the bottom. You can also see that version 3.7.3 is the current version of Python running.



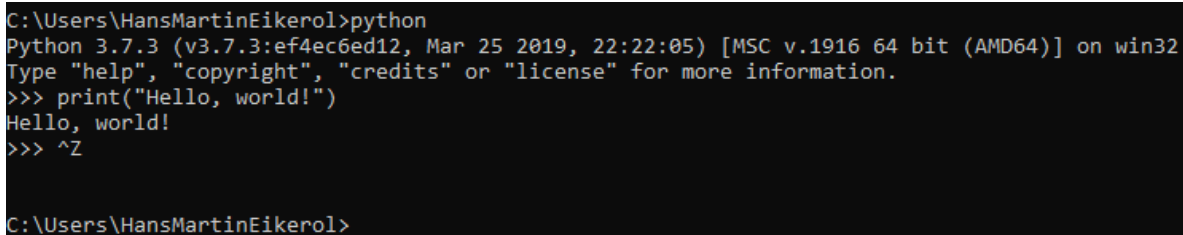
Let's try a simple command from part 1:

```
print("Hello, world!")
```



```
C:\Users\HansMartinEikerol>python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, world!")
Hello, world!
>>>
```

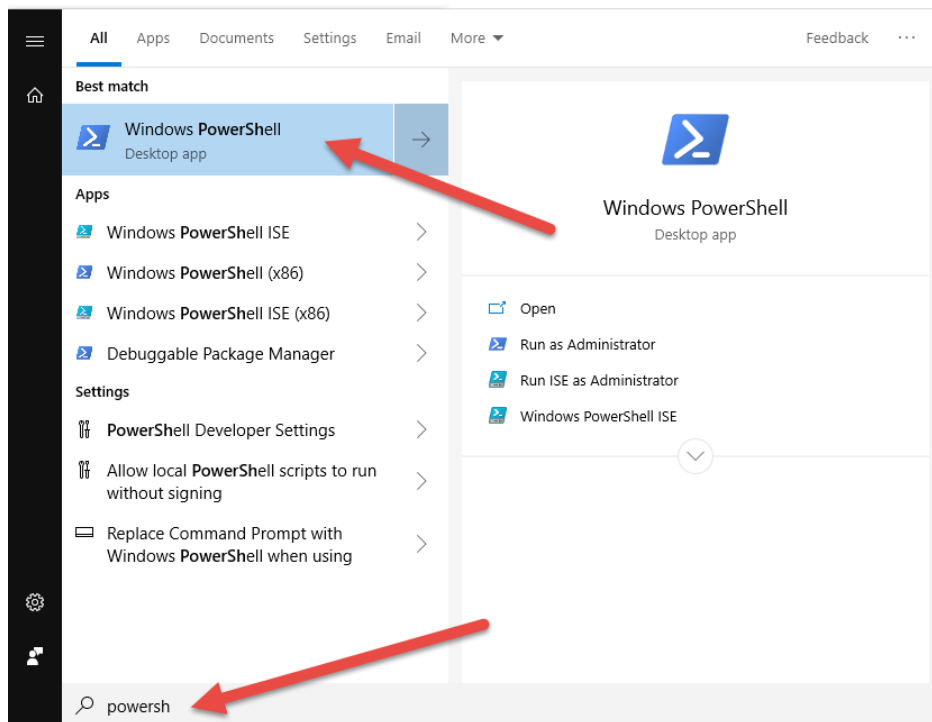
And we can see it is working! To exit the python shell, do a CTRL+Z, then press enter.



```
C:\Users\HansMartinEikerol>python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, world!")
Hello, world!
>>> ^Z

C:\Users\HansMartinEikerol>
```

Another shell client in windows is the Windows PowerShell. Try the same in PowerShell. You can find it in windows by searching for it.



You should get the same results in PowerShell as you got in the command prompt.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

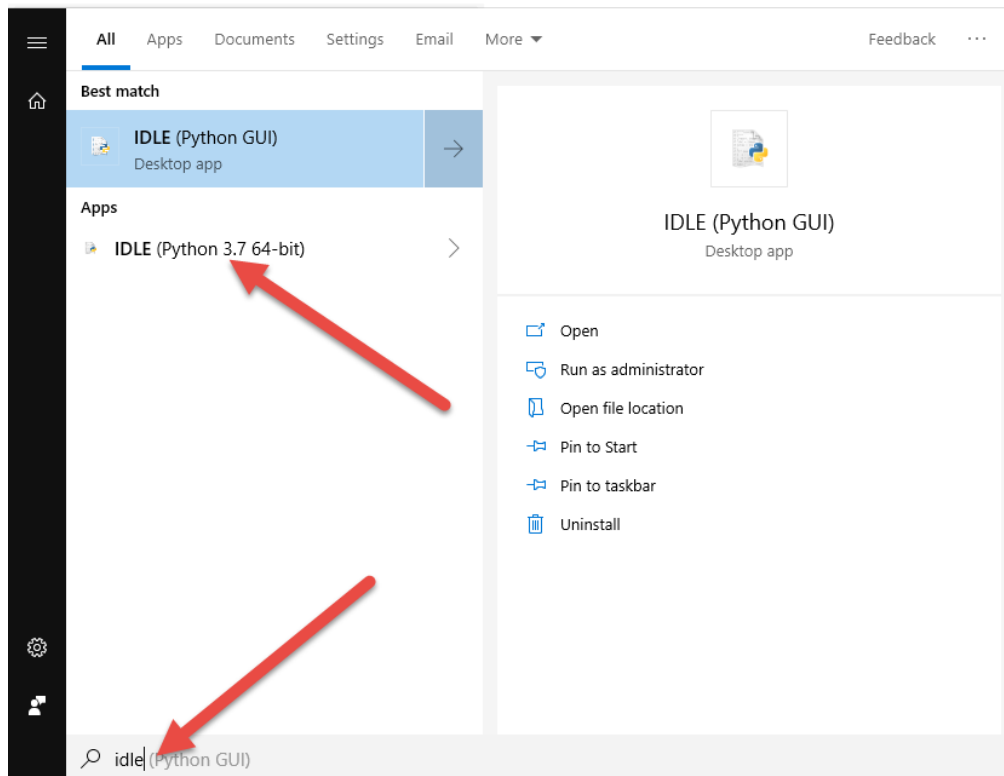
PS C:\Users\HansMartinEikerol> python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello, world!")
Hello, world!
```

PowerShell is the upgraded version of the traditional command prompt and can do many more tasks than the command prompt. You should pick using the PowerShell over the command prompt most of times.

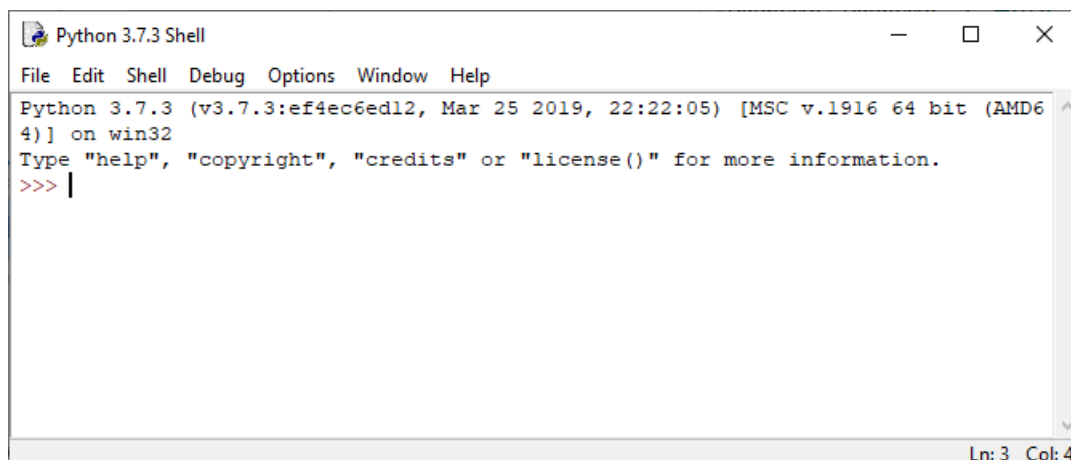
2.3 Writing your first script

Running small pieces of Python code in a shell is fun enough. However, writing and saving scripts to file is fundamental to create reusable code. We will create the first script now.

When you installed Python (any version), you will also get a Python shell called IDLE. Open IDLE by searching for it in windows.



If you have multiple versions of Python installed, different versions of IDLE will appear. For now, open IDLE for Python 3.7.



Since we want to write and save a script file, go to **File->New File**. This will open a simple file editor where you can write your own Python code, save the file, and at any time run your code to test it by hitting the **F5**-button.

You learned about basic variables and types in part 1 of this course. Now we will revisit integers in our small sample script. As previously noted, there are differences between version 2.X and 3.X of Python. One of the major differences will be illustrated by Sample1.py:

File name: Sample1.py

```
x = 1
y = 2

a = x/y
print(a)
```

What do you expect the output to be? 0.5, right? Here is the output:

Python 3.X:
0.5

Python 2.X:
0

This is due to something called integer division, or floor division. They changed this in version 3 to always use floating variable dividing, even when using integers. Knowing which version of Python you use will affect your code.

However, there is a way to create consistent answers between the versions:

File name: Sample2.py

```
x = 1.0 #Note this small change here
y = 2

a = x/y
print(a)
```

Output:
Python 3.X:
0.5

Python 2.X:
0.5

By using floating numbers as one either above or below the divisor, the language will know to use floating point division for the entire expression. Integer division is something that's implemented in many programming languages, like C#. Knowing to use floats as part of the divide operation can be crucial.

2.4 Working with files

Create a new file called **Sample 3.py**. In this script, you will use the **open**-command to write a simple file containing text to your disk. This doesn't require you to import any modules or libraries in order to run, as its natively part of the language.

File name: Sample 3.py

```
# Notice the "w" as the second argument.
# It tells the function to open the (new?) file with a write access
file = open("output/testfile.txt", "w")

# Notice the \n at the end of each write statement.
# This is the newline-character
file.write("Hello World\n")
file.write("This is our new text file\n")
file.write("and this is another line.\n")
file.write("Why? Because we can.\n")

file.close()
```

Now we will try to read the contents of the same file and printing its content by using the same command.

File name: Sample 4.py

```
file = open("output/testfile.txt", "r")
lines = file.readlines()
file.close()
print(lines)
```

Output:

```
['Hello World\n', 'This is our new text file\n', 'and this is
another line.\n', 'Why? Because we can.\n']
```

2.5 Exercise 1 – writing your first file

Now you shall use what you've learned about writing files. Create a new file called **Exercise 1.py**.

By using what you learned in part 1 about while-loops and strings, you shall output the numbers of 1 up to 100 on separate rows to the file at location *"output/numbers1to100.txt"*. The content of the file should be like this:

```
1
2
3
4
5
(...)
98
99
100
```

2.6 Exercise 2 – reading your file

Create a new file called **Exercise 2.py**. Use what you learned to print every third number from the file you just created to the command line using the **print**-command. Expected output is:

```
1
4
7
10
(...)
97
100
```

3 Helping Grethe

We will be continuing to work with Grethes hus in this part of the course. Since Grethe is planning on adding a third level to her house, she wants to send out neighborhood warnings to all neighbors within a 100 meter radius that can be affected by her new house elevation. Writing down all those addresses and creating documents for each one is something she finds quite mundane, so she decides to ask you as her friend to program her way out of it.

3.1 Getting real, using some modules

Now that you got a basic understanding of writing and reading files, we move on to reading the common file format CSV – comma separated values. Lots of datasets comes in this format, and its a default Excel file format.

In Python, there are libraries for doing about anything. Make sure you do a quick google search whenever in need for anything. Someone's probably already done the job for you.

This time, we will use the built-in module called **csv**. To read up on it, visit this link:
<https://docs.python.org/3/library/csv.html>

Statens Kartverk has lots of free data available for download on their website. Among those are the dataset "Matrikkelen – adresse". We will be using such a dataset of Oslo. All the free data is available for download here:

<https://www.geonorge.no/>

And the adresse-dataset:

<https://kartkatalog.geonorge.no/metadata/uuid/f7df7a18-b30f-4745-bd64-d0863812350c>

Matrikkelen - Adresse

Datasett

 Vis i kart	 Legg til nedlasting	 Vis dekningskart	 Hjelp	 Kontakt dataeier
 Vis produktark	 Vis produktspesifikasjon	 Vis tegneregler	 Nettside	 Vis produktside

Offisielle fysiske adresser registrert i Matrikkelen (Norges offisielle eiendomsregister). En offisiell adresse er den fullstendige adressen for en bygning, bygningsdel, bruksenhet, eiendom eller et annet objekt. I tillegg er adressens knytning til eiendom (matrikkelnummer, -ikke ned på seksjonsnivå), ogulike kretser (post, valg, tettsted, sokn, grunnkrets) med. En adresse er enten Vegadresse (Storgata 10) eller Matrikkeladresse (33/2-2). Det er et mål at alle matrikkeladresser skal erstattes av vegadresser. Flere distribusjoner er satt opp mot en løsning som gir noe forsinkelse fra det offisielle Matrikkelsystemet (fra ca. 10 minutters forsinkelse på WFS og for nedlasting av fritt valgt område fra kart, daglig generering av kommunevis filer og WMS og ukentlig for fylkes-/landsfiler).



Create a new file called **Grethe5.py**. In this file, we will learn how to use the CSV reader with the input-dataset of addresses in Oslo.

File name: Grethe5.py

```
import csv

with open('input/matrikkelenAdresse.csv') as csvfile:
    reader = csv.reader(csvfile)
    x = 1
    for row in reader:
        if (x % 1000 == 0):
            print(x) # Print for every thousand row
        if (x % 1000000 == 0):
            break # Make sure we kill the process at some point
        x += 1
```

Since this is a large dataset, it can be nice to get some output to the console while the program is running. Run this script. Try to understand what it does. Next up we will modify this script to start working with its content. Create a copy of the file which you call **Grethe6.py**.

While debugging and working with large datasets, its nice to have something that kills the process. We keep that part for now.

File name: Grethe6.py

```
import csv

with open('input/matrikkelenAdresse.csv', encoding='utf-8-sig') as csvfile:
    reader = csv.DictReader(csvfile, delimiter=";")
    x = 1
    for row in reader:
        north = row["Nord"]
        east = row["Øst"]
        if (x % 1000 == 0):
            print(east, north) # Print for every thousand row
        if (x % 10000 == 0):
            break # Make sure we kill the process at some point
        x+=1
```

Notice that we changed the reader to a DictReader. This allows us to access all cell values in a row by their header names coming from the CSV-file.

Next step is to extract all addresses from within a 100 meter radius from Grethes house. We know that Grethes hus is located in Storkenebbveien 1. We find it in the CSV-dataset, and fetch the coordinates. The house is at: 6648240.0 north, 596674.0 east.

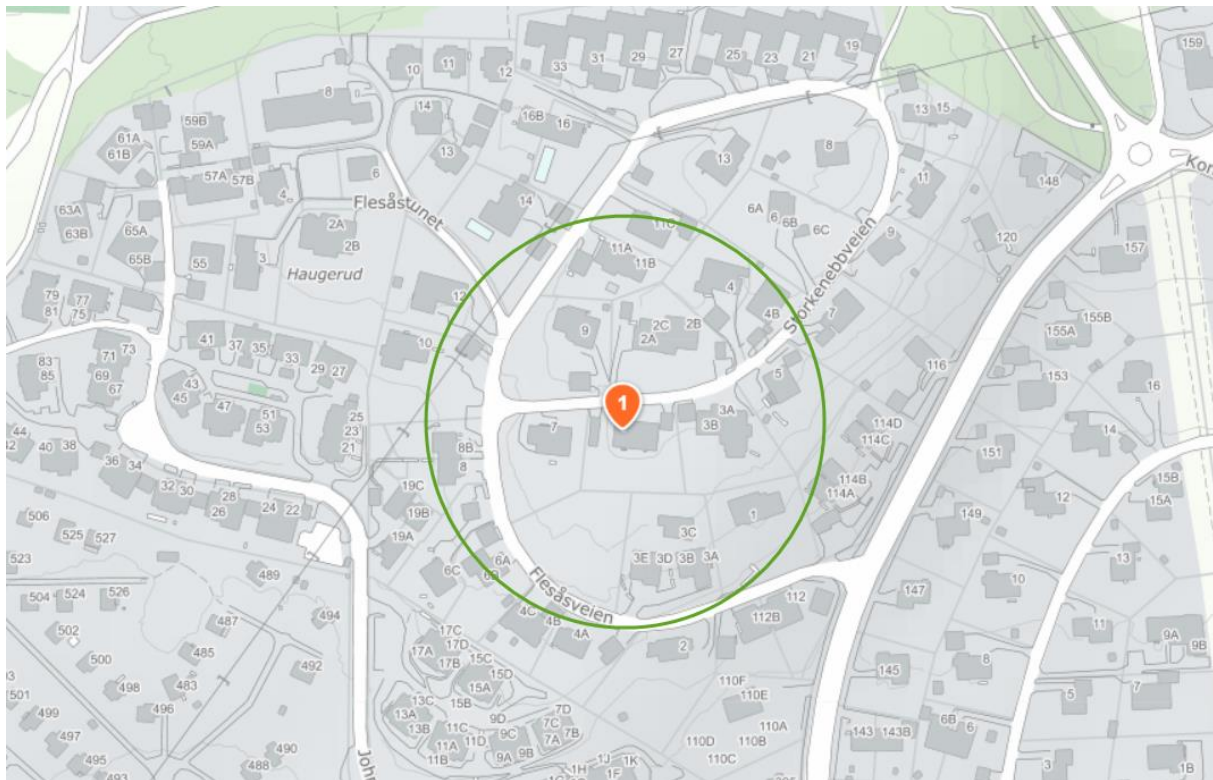
16762 Sognsveien	157	50	160	Sognsveien 157	Sognsveien 157	25832 6648321.0 596885.0	860 OSLO
16762 Sognsveien	159	50	161	Sognsveien 159	Sognsveien 159	25832 6648408.0 596907.0	860 OSLO
16762 Sognsveien	165	51	72	Sognsveien 165	Sognsveien 165	25832 6648553.0 596836.0	860 OSLO
17065 Storkenebbveien	1	51	29	Storkenebbveien 1	Storkenebbveien 1	25832 6648240.0 596674.0	860 OSLO
17065 Storkenebbveien	13	51	48	Storkenebbveien 13	Storkenebbveien 13	25832 6648377.0 596795.0	860 OSLO
17065 Storkenebbveien	15	51	40	Storkenebbveien 15	Storkenebbveien 15	25832 6648378.0 596804.0	860 OSLO

3.2 Exercise 3 – extract a list of Grethes neighbors

In this exercise, create a file called **GretheExercise 3.py**. With this script, create a text file that outputs Grethes neighbors within a 100 meter radius based on what you’ve learned until now. Here is a small function to help you out:

```
import math

def distance_to_point(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    distance = math.sqrt(dx**2 + dy**2)
    return distance
```



Format and write the results as separate rows to the text file "output/grethes_neighbors.txt". Also output it to the command line. The results should look like this:

```
Flesåsveien 1, Gnr/Bnr 51/24
Flesåsveien 8B, Gnr/Bnr 51/47
Flesåsveien 8, Gnr/Bnr 51/46
Flesåsveien 7, Gnr/Bnr 51/28
Flesåsveien 3C, Gnr/Bnr 51/26
Flesåsveien 2, Gnr/Bnr 51/17
Flesåsveien 9, Gnr/Bnr 51/35
Flesåsveien 10, Gnr/Bnr 51/49
Flesåsveien 12, Gnr/Bnr 51/50
Flesåsveien 11C, Gnr/Bnr 51/38
Sognsveien 112B, Gnr/Bnr 51/39
Storkenebbveien 1, Gnr/Bnr 51/29
Storkenebbveien 4, Gnr/Bnr 51/34
Storkenebbveien 4B, Gnr/Bnr 51/74
Storkenebbveien 5, Gnr/Bnr 51/19
Flesåsveien 6A, Gnr/Bnr 51/21
Flesåsveien 6C, Gnr/Bnr 51/85
Storkenebbveien 3A, Gnr/Bnr 51/30
Storkenebbveien 3B, Gnr/Bnr 51/30
Flesåsveien 4C, Gnr/Bnr 51/18
Flesåsveien 4B, Gnr/Bnr 51/18
Flesåsveien 4A, Gnr/Bnr 51/84
Storkenebbveien 2B, Gnr/Bnr 51/20
Storkenebbveien 2A, Gnr/Bnr 51/20
Storkenebbveien 2C, Gnr/Bnr 51/20
Sognsveien 114B, Gnr/Bnr 51/25
```

Sognsveien 114A, Gnr/Bnr 51/25
Flesåsveien 6B, Gnr/Bnr 51/21
John Brandts vei 19B, Gnr/Bnr 51/101
John Brandts vei 19C, Gnr/Bnr 51/101
Flesåsveien 3D, Gnr/Bnr 51/26
Flesåsveien 3B, Gnr/Bnr 51/26
Flesåsveien 3A, Gnr/Bnr 51/26
Flesåsveien 3E, Gnr/Bnr 51/26
Flesåsveien 11A, Gnr/Bnr 51/102
Flesåsveien 11B, Gnr/Bnr 51/102

4 Automating PDF document writing

Since all Grethes neighbors should receive about the same document, with a few minor changes in the address fields, this is a great application for using Python!

First, we need a PDF handling library. After a small google search, we find that PyPDF2 is a decent library that can do what we wants. We also need ReportLab. Run **pip install PyPDF2** and **pip install ReportLab** in Windows PowerShell.

```
PS C:\Users\HansMartinEikero1> pip install PyPDF2
Collecting PyPDF2
  Downloading https://files.pythonhosted.org/packages/b4/01/68fcc0d43daf4c6bdb6b33cc3f77bda531c86b174cac56ef0ffdb96faab/PyPDF2-1.26.0.tar.gz (77kB)
    100% |#####| 81kB 1.1MB/s
Installing collected packages: PyPDF2
  Running setup.py install for PyPDF2 ... done
Successfully installed PyPDF2-1.26.0
You are using pip version 19.0.3, however version 19.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

We have downloaded the background PDF we need from the proper authorities:

Byggesaksblanketter

Her laster du ned søknadsskjemaer og blanketter til byggesaken

Sist endret 05.03.2019

Skal du fylle ut elektronisk?

Last ned skjemaet og husk å lagre det. Hvis du ikke lagrer, kan utregninger bli feil.

Er du privatperson?

Du kan bruke enklere skjemaer hvis du skal nabovarsle og bygge en frittliggende bygning mindre enn 70 m² eller et tilbygg mindre enn 50 m². Du kan også bruke et enklere skjema hvis du skal søke om bruksendring.

➤ Alle skjemaene for mindre byggeprosjekter på boligeiendom finner du her.

Nummer	Informasjonsblanketter	Bokmål	Nynorsk
5154	Nabovarsel	5154	5154N
<i>Privatpersoner kan bruke enklere nabovarslingsskjema for mindre byggeprosjekter som garasjer, bod, tilbygg og bruksendring:</i>			

Now, we will try to make use of the new libraries we have downloaded. When using pip for managing Python packages, then downloaded modules get places inside the “Lib/site-packages”-folder of your Python installation. You can also manually download python modules and place them directly inside the “Lib/site-packages”-folder.

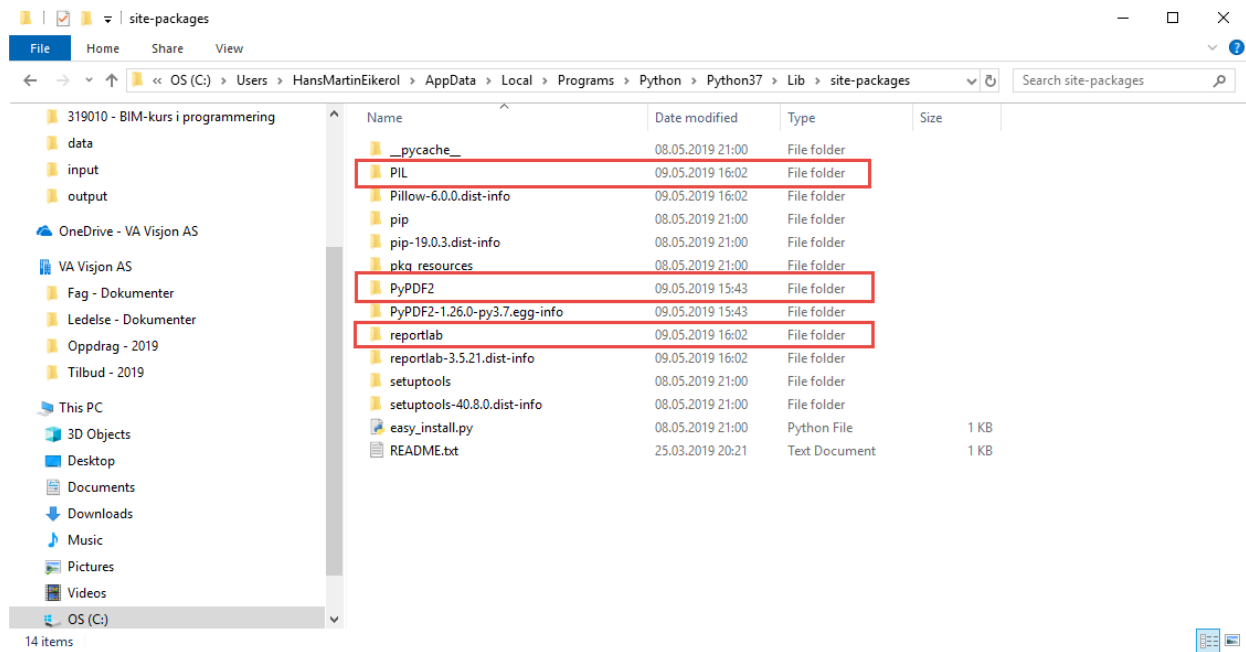
For 2.7 this will be folder:

C:\Python27\Lib\site-packages

For 3.7 this will be folder:

C:\Users\<Your username>\AppData\Local\Programs\Python\Python37\Lib\site-packages

Check that you have the required packages installed on your machine before continuing. You should have these:



(PIL is an image handling library used inside reportlab, and should get downloaded with reportlab. If it didn't, run **pip install PIL** or **pip install pillow** as well).

Now we will continue with the samples. Once you confirm you have the necessary libraries, create a new file called **Grethe3.py**.

File name: Grethe3.py

```
#Import required packages
from PyPDF2 import PdfFileWriter, PdfFileReader
import io
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4

# Define variables
debugGrid = False
imagepath = "input/Hans Martin Eikerol - signatur.png"
packet = io.BytesIO()

# Create a new PDF with Reportlab
can = canvas.Canvas(packet, pagesize=A4)
can.setFont("Times-Roman", 11)
print (can.getAvailableFonts())

can.drawString(65, 700, "Kjære nabo")
can.drawString(80, 637, "Gnr")
can.drawString(138, 637, "Brn")
can.drawString(80, 616, "Adresse")
can.drawImage(imagepath, 440, 45, width=60,
mask=[250,255,250,255,250,255], preserveAspectRatio=True,
anchor="sw")
```

```
# Print debug grid to help with text placement
if debugGrid:
    can.grid(range(0,800,20),range(0,1000,20))
can.save()

#####
# Start writing PDFs
#####

# Move to the beginning of the StringIO buffer
packet.seek(0)
new_pdf = PdfFileReader(packet)

# Read your existing PDF
existing_pdf = PdfFileReader(open("input/nabovarsel-utfylt.pdf",
"rb"))
output = PdfFileWriter()

# Add the "watermark" (which is the new pdf) on the existing page
page = existing_pdf.getPage(0)
page.mergePage(new_pdf.getPage(0))
output.addPage(page)

# Finally, write "output" to a real file
outputStream = open("output/destination.pdf", "wb")
output.write(outputStream)
outputStream.close() #Remember to close it
```

Examine the file. Try setting the variable `debugGrid` to `True`. Notice how we in this sample are leveraging PyPDF2 and reportlab to get the required code. Next up is the exercise.

4.1 Exercise 4 – converting code to functions

This exercise, we split in two. First, we need to modify the code from `Grethe3.py` and make it into a function or a class. Create a new file called **GretheExercise4.py**, based on the code of **Grethe3.py**. Convert the code into a function called `write_pdf`, which accepts the arguments as shown:

```
def write_pdf(destinationPath, signatureImage, neighbor, gnr, bnr,
adress):
```

Test the function by calling:

```
write_pdf("output/dest.pdf", "input/Hans Martin Eikerol -
signatur.png", "Kjære nabo", "50", "100", "Testveien 1")
```

You should now see a new PDF called "dest.pdf" in the output folder. Once you verify that the function works as expected, delete the function test call from the file. Also remove anything else that makes a print-statement inside the function.

4.2 Exercise 5 – using self-made functions

You now got functional code in GretheExercise2.py. Create a new file called **GretheExercise5.py**. In this file, you will import the function you made in the previous exercise. Since you already wrote a file containing data for all neighbors, read the file's data and generate separate signed neighborhood warnings for all Grethes neighbors. Base your script on the following sample:

```
from GretheExercise2 import write_pdf
import os

#First make sure the directory exists
os.mkdir("pdfs")

file = open("output/grethes_neighbors.txt", "r")
lines = file.readlines()
for line in lines:
    print(line, end="")
    # Write your code here

file.close()
```

Modify the script to make a separate output directory for the PDFs and write each one as separate PDFs for each neighbor. The results should look like this:

4.3 Exercise 6

Modify the script you made in Exercise 3 to become a function which takes input parameters `centerPoint` and `radius`. The new function should not write the neighbors to a file but send this data directly to the script for generating PDFs for all neighbors. Combine what you learned in the past three exercises to make some nice functionality. Name the file **GretheExercise6.py**.

5 Python with IfcOpenShell

As some may have noticed, at the time of writing there is no compiled version of IfcOpenShell available for download for Python 3.7.3. Thus, we have made it available to you. Unzip the file in the "lib"-folder of the project, and place it underneath the Lib-folder within your Python 3.7 installation:

C:\Users\<Your username>\AppData\Local\Programs\Python\Python37\Lib

Rename the extracted folder to only be named **ifcopenshell** – without the version numbering.

5.1 Using IfcOpenShell to modify and write IFC files

Next up, we will try some more advanced IFC handling. This time we will learn to modify an IFC-file, and write out a new file containing the modifications. Open the file **IfcSample5.py**.

Extras

6 How to build IfcOpenShell for Python 3.7

Note: The original script for dependencies and its download paths are pointing to Python library paths that no longer exist (no such MSI-file for python 3.7.3). Thus, you need to download the appropriate versions of Python separately.

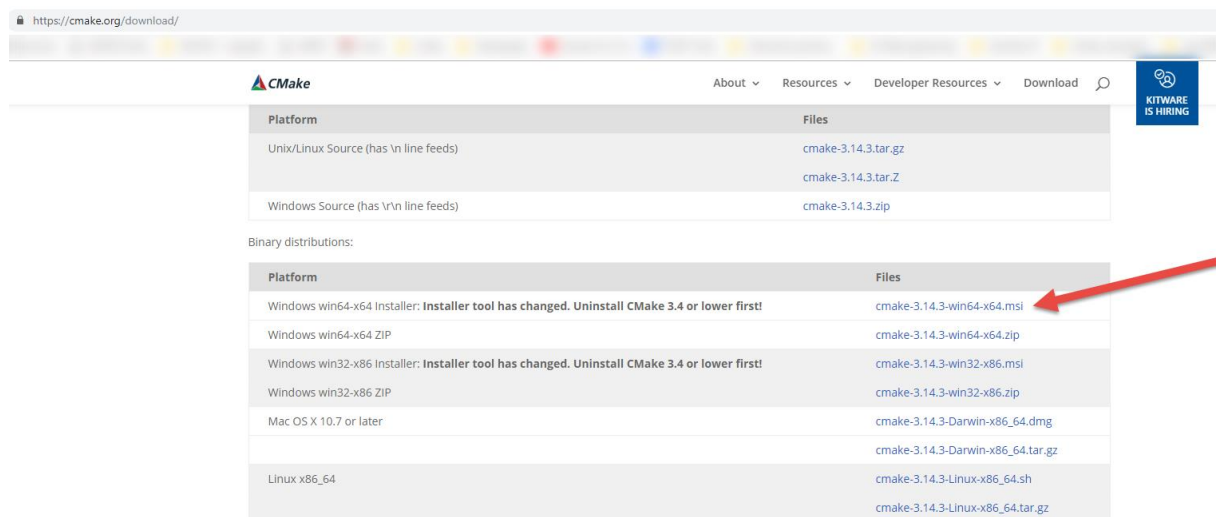
For original instructions, check this page:

<https://github.com/IfcOpenShell/IfcOpenShell/blob/master/win/readme.md>

Install git on your computer.

Git clone IfcOpenShell to your desktop

Install cmake on your computer.



Install 7-Zip on your computer. Add its root folder to PATH.

Start a new instance of Visual Studio Developer Console. Run the script:

```
C:\Program Files (x86)\Microsoft Visual
Studio\2017\Community\Common7\Tools\VsDevCmd.bat
```

Then open the /win directory inside the IfcOpenShell-directory.

```
cd "C:\git\IfcOpenShell\win"
```

Open the file build-deps.cmd in notepad or similar, and edit the following:

```
IF NOT DEFINED IFCOS_INSTALL_PYTHON set IFCOS_INSTALL_PYTHON=FALSE
(...)
set PYTHON_VERSION=3.7.3
```

Then start by running the following command in the developer console:

```
build-deps.cmd vs2017-x64 Release
```

Make sure you get no warnings. Change the build settings according to your needs. Once complete, make sure to set a few variables for the future scrips:

```
echo PY_VER_MAJOR_MINOR=37>> BuildDepsCache-x64.txt
echo PYTHONHOME=
C:\Users\HansMartinEikero1\AppData\Local\Programs\Python\Python37>>
BuildDepsCache-x64.txt
```

Open BuildDepsCache-x64.txt and delete whats marked:

```
1 GEN_SHORTHAND=vs2017-x64
2 OCC_INCLUDE_DIR=c:\git\IfcOpenShell\deps-vs2017-x64-installed\opencascade-7.3.0\inc
3 OCC_LIBRARY_DIR=c:\git\IfcOpenShell\deps-vs2017-x64-installed\opencascade-7.3.0\win64\lib
4 PY_VER_MAJOR_MINOR=37
5 PYTHONHOME=c:\git\IfcOpenShell\deps-vs2017-x64-installed\Python37
6 PY_VER_MAJOR_MINOR=37
7 PYTHONHOME=C:\Users\HansMartinEikero1\AppData\Local\Programs\Python\Python37
8
```

Then, you can are ready to run the next script:

```
run-cmake.bat
```

Then run:

```
build-ifcopenshell.bat
```

Then:

```
install-ifcopenshell.bat
```

The Python build upon successful build will be in this folder:

C:\Users\HansMartinEikero1\AppData\Local\Programs\Python\Python37\Lib\site-packages\ifcopenshell