



University of  
Stavanger

Faculty of Science  
and Technology

Stavanger, January 3, 2025

## ELE610 Applied Robot Technology, V-2025

### Fragments of Python stuff

My hope is mainly that this document will be helpful to students in ELE610, and perhaps also others who may be interested in the examples presented here. One reason for this document is to summarize my experience learning, and teaching, Python. Python has all the features wanted and is well suited for most of the tasks I do. This document is still “work in progress”, which it will probably always be.

Per July 2023 Python 3.12 is latest version and this, or version 3.11 which I use now, should be a good choice to use. However, I have installed several Python version, as Anaconda allows one to easily switch between different environments, the differences between different Python versions are mostly not important for ELE610. Thus, if you have **Python 3.x** ( $x \geq 9$ ) and is happy with this there should be no particular reason to update. The important issue is that the needed additional packages will work in your computer environment. The many packages used in ELE610 are mentioned in this document.

The organization of this document is made by a first section with some general links to useful resources on the web. Section 2 has stuff on installation and use of Python, mostly based on Anaconda and a simple command line interface, but some integrated developer environments (IDE) are also mentioned. The rest of this document describes some commonly used packages. In particular, section 3 briefly describes the general packages for numerical calculations and visualization: **numpy** and **matplotlib**. Section 4 describes packages for image processing; the huge and general **OpenCV** package and the special **pyeye** package for image acquisition using IDS cameras. Section 5 is on the huge **Qt** package for developing programs with graphical user interface (GUI). This section also presents some simple examples. Finally, section 6 briefly describes other packages that may, or may not, be relevant for your Python program.

# 1 Resources

- The (official) [Python documentation ↗](#).
- The (official) [Python reference ↗](#).
- The web page from [DataCamp ↗](#) includes many cheat sheets. Cheat sheet for `numpy`, `SciPy`, linear algebra, and more are available here.
- A good Python [Cheat sheet ↗](#) with much information compressed to two pages.
- Bernd Klein [Python course ↗](#).
- [OpenCV tutorial ↗](#).
- The official [Qt documentation ↗](#).

# 2 Installation and use

Installation of Python is not as easy as we could want, but it is getting better. The main challenges are that it is a large system, with many packages, and that it should be possible to install it on most system (which it is). The development of Python packages is done by many independent groups and individuals.

## 2.1 History

Python 2 was introduced in 2000, and the backward *incompatible* Python 3 was introduced in 2008, see history in [Wikipedia ↗](#). Python 2 (2.7) was until some years ago still used by many, and version 2.7 was updated until 2014 when it was frozen, and it will not be supported after 2019. We abandoned this version when the NAO-robots “retired” in 2020.

Python 3 has been gradually developed for many years now and a new “version” extending the previous version by adding some new and improved features have been released every (second) year. Also, the number of packages, which add functionality for particular areas, has become quite large, many packages have become obsolete and others have developed a great deal through many different versions. **The different Python packages can be difficult to get to work together.** To keep track of which packages, and versions of them, goes with which Python version and which operating system and which packages may be used at the same time is both difficult and frustrating.

## 2.2 Python installation

If you have already installed Python 3.x, ( $x \geq 9$ ), and your favorite IDE (integrated development environment) and this works well, there is no reason to reinstall Python. You can scroll forward to section 2.3 as you have to check that the packages needed in ELE610 are installed on your computer.

If you know that you will use only one version of Python and one quite standard setup for the packages you may install Python from the official [Python downloads page](#) ↗. I recommend that you install the newest (stable) version, or version 3.11. You will need administrator access to the computer.

Packages should then be installed by `pip`. PIP is an acronym that stands for “PIP Installs Packages” or “Preferred Installer Program”. PIP is now included in the Python installation. PIP is a command-line utility that allows you to install, reinstall, or uninstall Python packages with a simple and straightforward command: `pip`. Almost all packages can be installed by `pip`, including the UiS package for Robot Web Services: `rwsuis`.

Many persons, included me, prefer to use a package manager program, and of course several exists. I think [Anaconda](#) ↗, or the core part **Miniconda**, is one of the better ones. The large Anaconda uses 3GB disk space, (Mini)conda uses 400 MB. Anaconda handle incompatible versions and packages combinations by setting up different *environments*, you may have installed one environment with Python 3.9 and some packages and another environment with Python 3.11 and some other packages and easily add more packages into an environment and easily switch between different environments.

You may also consider an Integrated Developer Environment (IDE). The most used IDE now is [Visual Studio](#) ↗, a general programming IDE from Microsoft. This will work well. [PyCharm](#) ↗ has been used in basic programming course at UiS and is a good alternative in ELE610. Other students have used [Spider](#) ↗, and this has also worked well. The most important thing is that each student uses an IDE she is familiar with and where she is able to solve the “package installation challenges” that may occur.

## 2.3 Packages used in ELE610

Many UiS students are familiar with Visual Studio. They can continue to use this in ELE610, but should be aware that Karl don't know Visual Studio. However, our laboratory engineer Asbjørn knows Visual Studio and should be able to help with most common questions you may have. Asbjørn has prepared some files to make package installation as simple as possible for Visual Studio users, these files are a docx-file explaining the Visual Studio installation and a file requirements.txt. These two files should be in [ELE610py3files.zip ↗](#). The files are also on Canvas Files. The rest of this section is mainly for those of you who still uses command line interface (CLI) or Anaconda, the rest of you may jump directly to section 3.

Anaconda should be installed on the computers in room E462 and E464. You may use Anaconda, or a similar IDE, on your laptop computer or on the laboratory computer on the desk assigned to your ELE610 group. To start Anaconda on the laboratory computer select<sup>1</sup> *Anaconda Prompt (Anaconda3)* that may be in the *Anaconda 3 (64 bit)* group. You should then get a window with the command line. The path, here root `C:\>`, may be another catalog than root.

```
(base) C:\>
```

To list all packages installed in the active environment (here `base`) type:

```
(base) C:\> conda list
```

You should change to ELE610 catalog and perhaps create and then activate the correct environment. If you use your own computer you may want to create a new environment just to install, or upgrade, the ELE610 packages, this to avoid any problems in the environments you already have. The following installation guide is by no means complete, you should look at the installation guidelines on the web-site for each package to be sure installation is done correctly on your computer. The preferred version of Python is 3.12, but if you have a version equal to or newer than version 3.9 there should be no reason to upgrade. The `[ ]` brackets indicate an optional term, as the `[conda]` term 3 lines below.

```
(base) C:\ELE610\> conda create --name py12 python=3.12
```

```
(base) C:\ELE610\> conda env list          (list environments)
```

```
(base) C:\ELE610\> [conda] activate py12
```

```
(py12) C:\ELE610\> conda list             (list installed packages)
```

You now have to install (or upgrade) the wanted packages. I suppose `pip install <package>` can be used for most packages, it is the recommended installer and assumed below. But `conda install <package>` may be needed for some packages on Anaconda, i.e. `pyqt`.

---

<sup>1</sup>Use the window start button in the lower left corner of the screen, or search for “Ana” in the search field.

To further complicate (or perhaps solve some issues) another alternative for package installation exists, as some packages should perhaps rather be installed from within python using `python -m pip install <package>`. To know why or when search web for *difference between python -m pip install package and pip install package*.

Unfortunately, OpenCV is one of the packages that may cause problems. Quite recently I experienced that the OpenCV installation on my laptop did not work, importing `cv2` the error message was `ImportError: DLL load failed while importing cv2: ....` Searching web I found that this problem was not uncommon, and the suggested solutions were many. Trying many of these I finally made OpenCV work, but I am not able to recall what actually solve the problem.

Below the prompt (py12) `C:\ELE610\>` is simply written `>`. More information on most of these packages can be found in later this document. The packages are:

```
> pip install numpy
> pip install matplotlib
> pip install opencv-python
> pip install requests
> pip install pyzbar
> pip install cmd2
> pip install bleak
```

The `conda` installation for Qt may be necessary as `pip` may not install Qt correctly from within `anaconda`. We encountered a problem some years ago, and some cryptic error messages indicated that a DLL was missing. Perhaps this problem is fixed now, but it was very frustrating until one clever student, Kestutis, worked on this problem and after hours of searching and testing March 2021 found that `conda install pyqt` usually worked. The problem is explained on the web: [Anaconda issues ↗](#). PyQtChart is not needed for ELE610, but if you want to install it you may need to ensure the same versions of Qt and QtChart, using `python -m pip install`. The examples we use in this course should work in both Qt5 and Qt6. Hopefully the line below will do the installation job.

```
> pip install pyqt6
```

Some of the Python packages needed in ELE610 are not commonly used so even if you have used Python extensively it is unlikely that these packages are already installed on your computer.

```
> pip install pyeye
> pip install qimage2ndarray
> pip install rwsuis
```

Some useful packages are probably not needed in ELE610 but may still be

installed.

```
> pip install scipy
> pip install pandas
> pip install alive-progress
```

To prepare for using Python in ELE610 you will probably want to have a catalog for the Python files and perhaps also keep images in the same catalog. I don't recommend to have different catalogs for each assignment. Using Anaconda you can freely change the current catalog, all needed catalogs for Anaconda commands are stored in the `path`. You may change to your ELE610 catalog and do:

```
...\ELE610> mkdir py
...\ELE610> cd py
...\ELE610\py> dir
```

Now I *strongly* recommend that you copy some of the files linked to from this document, or the assignments, into this catalog. To make this easy I have collected (most of) the Python files used for the ELE610 course into a zip-file: [ELE610py3files.zip ↗](#).

What follows is mainly for Anaconda, if you use an IDE you should only assert that you can run some of the downloaded Python programs. You may want to update one or more of the downloaded files to better fit your preferences, for example the contents of `startup.py` and `myTools.py` is actually adjusted to *my* preferences. You may want to delete (large) parts of these files, but there may still be something you may want to keep. Feel free to change any or all of these files.

To test Python in a simple Qt application you can copy [appAnalogClock.pyw ↗](#) to the current catalog, see section 5.1, and then do:

```
(py12) ... \ELE610\py> pythonw appAnalogClock.pyw
```

To start Python in interactive mode you can now use:

```
(py12) ... \ELE610\py> python -i startup.py
```

Hopefully you will see that the packages are imported as they were supposed to be. When you have explored the Command Line Interface (CLI) of Python a little bit, you return to the command window (of the operative system) by invoking the exit function:









```
>>> exit()
```

To test if OpenCV works you can run the `appImageViewer2` program, it should be in the zip-file mentioned above, and open a color image and transform this into a gray scale image. If this last operation works OpenCV works.

```
(py12) ... \ELE610\py> python appImageViewer2.py
```



## 2.4 Editors and IDE

The Python programmer is free to use his or her favorite text editor or to choose among several Integrated Development Environments (IDE), some listed below. I prefer a simple way to develop small Python programs and examples and use Notepad++ on Windows platforms and Notepadqq on Ubuntu.

- [Notepad++](#)  is a source code editor.
- [Notepadqq](#)  is a Notepad++ like editor for the Linux desktop.
- [Sublime](#)  is a sophisticated text editor for code, markup and prose.
- Some experienced users still prefer [Emacs](#) .
- [Spyder](#)  is a powerful IDE for the Python language.
- [PyCharm](#)  is another popular IDE.
- [Jupyter](#)  can be tried without installing anything.
- [Visual Studio Code](#) .

## 2.5 Python command line interface (CLI)

This section is mainly for those who uses Anaconda and CLI (Command Line Interface/Interpreter), also PyCharm and Spyder have the option to open a CLI. Having successfully installed Python and your favorite editor or IDE you will find that the Python interface is a very simple CLI. Compared to the user-friendly interface of MATLAB, Python CLI is rather primitive. Python is much more like other programming languages whose user interface is through the command line of the operating system or through an IDE.

At least one more advanced CLI for Python has been developed: [IPython](#)  is a powerful interactive shell. It provides a rich toolkit to help you make the most out of using Python interactively. This toolkit is commonly used together with [Jupyter](#) .

As I mainly use Python to make small programs and examples I prefer to keep it as simple as possible. In Windows, starting the command line and the Anaconda3 bat-file made by the conda installation, Anaconda displays a modified prompt in the command line window.

```
(base) C:\Users\...\Dropbox\ELE610\py3>
```

I now activate the wanted environment,

```
..\ELE610\py3> activate py12
```

and get a new prompt. I may start the Python interface (CLI)

```

..\ELE610\py3>python
or a startup-file and the Python interface
..\ELE610\py3>python -i startup.py
or a Python program without any CLI in the background
..\ELE610\py> pythonw appAnalogClock.pyw

```

In the standard Python interface most Python expressions can be executed, for example a variable may be created and assigned a value

```
>>> a = 42
```

Now, to list the variables defined the `vars()` function can be used. It returns a `dict` object, and if this object is not assigned to a variable the CLI displays the returned object:

```

>>> vars()
{'__name__': '__main__', '__spec__': None,
 '__loader__': <class '_frozen_importlib.BuiltinImporter'>,
 '__builtins__': <module 'builtins' (built-in)>,
 '__package__': None, 'a': 42, '__doc__': None}
>>>

```

Note that above new-line-character is added in the list (in natural places), in Python the `dict` object is just displayed as a string (without line-feeds, but wrapped to next line when window width is reached). For me this was a great step backwards from the `whos` command in MATLAB. To fill my most basic needs I made a small startup file that loads (import) packages (variables, functions, and classes), including some functions made for my particular needs, and display some system information. I made this file mainly for my own use, but I think most users that use Python CLI will benefit from using a (similar) startup file. The file is [startup.py ↗](#) and it is supposed to be started in interactive mode, meaning that it does not immediately return to the (Anaconda/OS) prompt.

```
(py12) C:\Users\...\Dropbox\ELE610\py3>python -i startup.py
```

My startup file starts with something like this

```

import sys
import os
from importlib import reload
# from math import *
from math import sqrt, pi, cos, sin, tan, log, floor, ceil
from myTools import DBpath, ls, whos
import numpy as np
# import matplotlib as mpl
import matplotlib.pyplot as plt
try:
    cv2Exists = True    # evt. imp.find_module('cv2')
    import cv2

```



```

except ImportError:
    cv2Exists = False

print( 'Package   sys ver. ' + sys.version )
print( 'Package numpy ver. ' + np.__version__ )
if cv2Exists:
    print( 'Package   cv2 ver. ' + cv2.__version__ )

```

Notice the functions imported from my own file, [myTools.py ↗](#). For example the `whos` function has the first argument as a `dict` or `list` object, ex. the output from `vars()` (default), and displays it in a more reader friendly format than simply printing it. The results is now displayed as:

```

>>> whos( vars() )
cv2                module
np                 module
os                 module
plt                module
sys                module
ceil               builtin_function_or_method
cos                builtin_function_or_method
floor              builtin_function_or_method
log                builtin_function_or_method
sin                builtin_function_or_method
sqrt               builtin_function_or_method
tan                builtin_function_or_method
DBpath             function
ls                 function
reload             function
whos               function

a                  int      : 42
cv2Exists          bool      : True
img                ndarray of uint8, size: 1088 2048 3
pi                 float: 3.141592653589793

```

As for many Python functions, and objects, `>>> help(whos)` display some hopefully useful information. If you want to see what is inside the `whos` function you may look at [myTools.py ↗](#), and perhaps also copy (parts of) it to your own “tools”.

## 3 NumPy, Matplotlib and SciPy

Some examples may be included here, but generally I think that the examples and documentation found on web is quite extensive and good. For `numpy` you may look at [the W3 Numpy Tutorial ↗](#). The web page from [DataCamp ↗](#) includes many cheat sheets, also for `numpy`. For `matplotlib` you may start at [matplotlib tutorials page ↗](#) and first make sure that you are familiar with the concepts presented in Usage Guide on this page. Then you may continue with more of the tutorials.

In my opinion `numpy` and `scipy` makes Python a good alternative to MATLAB for some tasks like (general) calculations and perhaps even for image processing when OpenCV is used. But for some tasks I still think MATLAB is superior, especially when it comes to installation, user friendliness, export and import of signals (files), in particular sound files, and visualization (plotting). The benefit of Python is that it is widely available and free, and it is a general programming language.

`matplotlib` and in particular `pyplot` makes visualization easier in Python, but it is still, in my opinion, not as user friendly as MATLAB. Nevertheless, `matplotlib` makes it possible to display results graphically in Python.

[SciPy ↗](#) extends `numpy` with some fundamental algorithms for scientific computing, i.e. for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics and many other classes of problems. You will probably not need SciPy in ELE610, nevertheless it may be a good thing to install it as you are likely to need SciPy's capabilities in other subjects.

## 4 OpenCV and pyueye

### 4.1 OpenCV

[OpenCV ↗](#) is a library of functions for computer vision tasks. It is mainly written in C++ and there are *bindings* for many other languages. The Python binding, [opencv-python ↗](#), builds on `numpy` and the images are stored in memory as `numpy`-arrays. The package has many functions for image processing. OpenCV should be quite easy to install using Anaconda, but there could be some trouble. I skip these issues here, but if you experience trouble installing OpenCV on Anaconda I may have some tips. Anyway, `appImageViewer2.py` can be used to test if OpenCV is actually working together with Qt, as described a little bit above, in the end of section 2.3.

You should note that for OpenCV, like Qt and `pyueye`, most code is written in

and for C++. *Bindings generators* create a bridge between C++ and Python, or Java or another platform, which enables users to call the C++ functions from Python. These bindings generators should hide C++ features and show them as Python features, ex: C++ `Mat` class should be shown as a numpy `array` in Python. This usually works fine for packages like OpenCV and Qt.

Having successfully installed the OpenCV package, the comprehensive [OpenCV documentation](#) will be useful as a reference. As an introduction the [OpenCV tutorial](#) will be even more useful.

#### `qimage2ndarray`

To convert OpenCV images, i.e. numpy arrays, to Qt `QImages` is not straightforward. To do this conversion, in either direction, the following small package is useful [qimage2ndarray](#). The package can be installed in Anaconda by: `pip install qimage2ndarray`.

## 4.2 pyueye

Imaging Development Systems GmbH (IDS) make industrial cameras. The cameras can communicate with computer software using an Application Program Interface (API). To get the correct software drivers installed (i.e. DLL-files) you need to install the uEyeCockpit program on your Windows PC<sup>2</sup>, it should be included in IDS Software Suite 4.94 and 4.95 and 4.96(?).

Registered users may get the installation file from IDS web pages<sup>3</sup>. When an IDS camera is attached to USB port the uEyeCockpit program can be used to capture and display an image. This is a good test to do to check that the drivers are properly installed. The help available from help menu in IDS Cockpit program explains the interface of the API functions that also can be called from Python. When IDS Cockpit program is installed (and working as intended), the [pyueye package](#) can be installed in the active Python environment as an ordinary Python (pip) package and the `appImageViewer2` program should be able to capture an image using the IDS camera. The program should have an IDS camera connected to the PC to work properly and the uEyeCockpit program can not simultaneously access the same IDS camera.

As the IDS camera driver are somehow difficult to use some previous students have made a Python class that wraps around the IDS driver and thus may make it easier to use. This file is: [clsCamera.py](#).

I find the IDS web-pages somehow confusing and difficult to use, thus an anxious reader **should skip the following paragraphs** and go directly to next section.

---

<sup>2</sup>On Mac PC it is generally not possible to install the IDS software

<sup>3</sup>UiS students in ELE610 can also find a link on Canvas or ask me

The map of IDS-pages and the access rules have changed over the years, thus some of the links below may be obsolete. I also think that IDS give all web user general access to some general web pages, but only registered users will get access to specific resources, and perhaps most of the IDS links below are to specific resources, I am not sure so you just have to try for yourself. IDS now seems to recommend the [IDS peak tool ↗](#) for Python usage, I have not used it myself but the most daring of you may want to look into this. Some [PyuEye release notes ↗](#) may be available, and some [IDS Software Suite 4.96 release notes ↗](#) (27/4/2022) may also be available.

The API documentation for `pyueye` can be found somewhere on IDS web pages, you may try [this link ↗](#), the manual for IDS industrial cameras IDS Software Suite 4.95. The API is explained using C++ syntax, but the corresponding Python functions should be similar. The manual also explains the general functionality of an IDS camera, and how to access this.

The binding generator should hide C++ features and show them as Python features, but for this low-level package `pyueye` the binding from C++ to Python is done by creating lots of new Python classes, each corresponding directly to a C++ class and the Python objects (variables) are basically used as a C++ objects. This way an `int` passed as argument in a function should be a `ueye.INT` Python object and its value can be changed (it is called by reference, as can be done in C, and not by value, as `int` is done in Python). IDS describes the binding generator precisely as:

... a lean wrapper implementation of Python function objects that represent uEye API functions. As there is *no intelligence* in the PyuEye interface, it is as near as possible to the uEye C API and always up-to-date.

The emphasize of *no intelligence* is done by myself here, as all C++ features are moved almost unaltered to Python. This is features like memory handling, argument passing, data types and more. To work with the `pyueye` package is, for most students, one of the more demanding tasks for the image acquisition assignments in ELE610.

The IDS documentation includes an embedded vision kit example where an image is captured by IDS software and processed by OpenCV. If you want to try this example follow the link to [IDS example ↗](#). This IDS TechTip shows in a few simple steps how to implement a simple embedded vision application with a uEye camera and a Raspberry Pi 3. A disadvantage with this example is that it uses Python 2.7 and Qt4. I also had to struggle a bit to understand parts of it. I have slightly modified this (or rather a previous version of this or a similar) example. Qt was also used in this example, and I changed it from Qt4 to Qt5 and made it work but probably not as well as in the original implementation. Links to the slightly modified Python-files are:

- [pyueye\\_example\\_utils.py ↗](#),
- [pyueye\\_example\\_camera.py ↗](#),
- [pyueye\\_example\\_gui.py ↗](#), and
- [pyueye\\_example\\_main.py ↗](#).

## 5 Python Qt

[Qt ↗](#) is a large library for writing GUI programs, i.e. Windows-like programs. Qt is developed in C++, and most documentation and examples are for C++. Searching the web for examples and solutions in Qt you often find examples in C++, and if you find something for Python it may be in the older version Qt4, which has some important differences to Qt5 and Qt6. The newer version PyQt6 is recommended instead of Qt5, these two versions are quite similar, see [GUIS web page explaining PyQt5 and PyQt6 ↗](#). To further increase the confusion there are two Python bindings of Qt *PySide* and *PyQt*.

For ELE610 you may use PyQt5 or PyQt6 (and the PyQt binding). The `appImageViewer` programs should now work in both Qt5 and Qt6, the two versions are quite similar to each other.

The official [Qt documentation ↗](#) is essential, but to use it in Python you should be aware of the differences between C++ and Python and how Qt is ported to Python. `PyQt6` is a Python binding of the `Qt` GUI toolkit, see [wikipedia ↗](#). A good way to learn the differences between C++ Qt and PyQt is to implement some of the [many Qt5 examples ↗](#) or [many Qt6 examples ↗](#) or just read the good (but big) Qt documentation, in particular the [Qt Widgets ↗](#) part.

Usually Python programs have the extension `.py` but for Windows GUI applications the alternative extension `.pyw` may also be used. This application may then be started using `pythonw`, which suppresses the terminal window on startup. I now prefer to use only the `.py` extension, as I even for GUI programs often use functions that print to terminal window, i.e. `print()`. The example programs in sections 5.1 to 5.3 below show some simple GUI examples.

### Python Qt and OpenCV

Combining several huge packages, that may have partly overlapping functions, like `numpy`, `OpenCV` and `Qt`, in a Python program is not always simple. It can be done in many ways, and no clear answers to how to do it best is available. The examples for the programs in section 5.3 below presents one way to do this. Other alternatives to learn about the `Qt` GUI (in Python) are:

- Michael Herrmann, Omaha Consulting GmbH, Austria, has written a textbook on Python and Qt, and also made a simple example or tutorial

available on the web: The [PyQt5 tutorial web page ↗](#) gives a quick overview and an example.

- Guru99, Steve Campbell, has a [web page ↗](#) named *PyQt5 Tutorial: Design GUI using PyQt in Python with Examples*.
- Aquiles Carattino has a [web page ↗](#) named *Using PyQt to build a GUI for your webcam*.
- A tutorial (an example) is on this [web page ↗](#). The title is *How to make OpenCV and PyQt5 based GUI for image processing applications*, so I expect it is relevant.

### 5.1 `appAnalogClock.pyw`

- The C++ example is [here ↗](#).
- A version for Qt6 is [here ↗](#).
- The corresponding Python file I made is [appAnalogClock.pyw ↗](#).

### 5.2 `appCalculator.pyw`

- The C++ example is [here ↗](#).
- The corresponding Python file is [appCalculator.pyw ↗](#).

### 5.3 `appImageViewer.py`

I have made some image viewer examples, from a quite simple one to some slightly more advanced, all intended as examples for students. An image is represented in many ways in these programs, see figure 1. For UiS ELE610 students there are some video lectures available on *canvas*, in Modules on the course page. I strongly recommend ELE610 students to **watch these videos** because they are very relevant.

- The very simple image viewer is available on the web page: [appSimpleImageViewer.py ↗](#). This Python example is loosely based on the C++ example [here ↗](#).
- Another simple image viewer is `appImageViewer` included in zip-file [ELE610py3files.zip ↗](#). The program displays an image using Qt, and it has some few options: On File menu the following options are included: Open File, Clear Image, and Quit. And on Scale menu: Scale 1, Scale

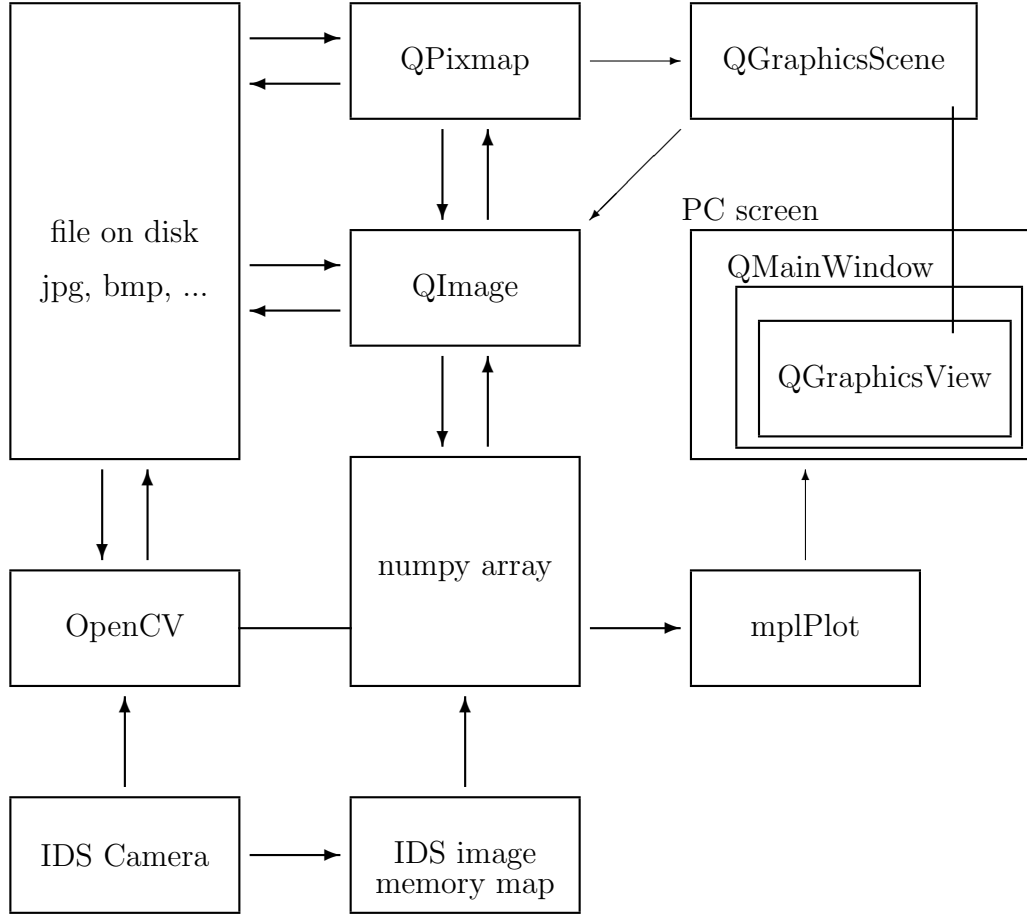


Figure 1: Illustration of connection between different representations of images as used in the appImageViewer programs. The boxes in center column are different formats of the image for storing image in PC memory. A `QPixmap` can be included in a `QGraphicsScene` as a `QGraphicsItem`. The scene, or a part of it, can be shown in a `QGraphicsView` which is a *widget* and can be displayed on the PC screen. A `QGraphicsScene` can also be *rendered* on a `QPaintDevice` which can be a `QImage`. An image captured by the IDS camera is stored in (camera) memory in a raw IDS format, this can be converted to a numpy array. OpenCV works mainly with images stored as numpy arrays but it can also read from and store to image files of many formats and it can read image directly from a camera, including IDS camera. The `mplPlot` package, and similar Python packages can be used to display images (normally in numpy format) on screen without using Qt.

Up, and Scale down. In the bottom of the GUI the value for pixel that mouse points on is shown. This program does not use `numpy`, `OpenCV` or `qimage2ndarray` or `ueye` (IDS camera).

- The next, and also the following viewers, is included in zip-file [ELE610py3files.zip ↗](#). The `appImageViewer1` program does not inherit, but is similar to, `appImageViewer.py`. Some simple image processing functions are included in this program. It uses packages `OpenCV` and `numpy` and will also work better if `qimage2ndarray` can be used to copy images between `numpy` representations and `Qt` representations.
- The `appImageViewer2` inherits `appImageViewer1.py` above, and some image capture functions are added. For the Camera functions to work it needs `pyueye` and the example files described in the end of section 4 in this document. Thus, to use this an IDS camera should be available.
- The `appImageViewer3` inherits `appImageViewer2.py` above, by adding functions for colors and image processing of **dice images**. It is a framework for image acquisition assignments 2 and 3 in ELE610.
- Finally, the `appImageViewer4` also inherits `appImageViewer2.py` above, by adding functions for **disk images** (here, not a computer file 'disk image', but an ordinary image/photo of the disk on our camera rig). It is a framework for image acquisition assignments 4 in ELE610.

## 6 More relevant packages

### Pandas

`pandas` is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. `pandas` should make it easy to work with structured data, both 1D as (Series) and 2D tables (DataFrame). I find `pandas` well suited to import and process csv-files. You may see [the user guide ↗](#). We may use `pandas` a little bit in ELE610, in RS4 we may use the graphical methods and `pandas` to present content of JSON-report file.

### Requests and rwsuis

`Requests` is a much used package that allows you to send HTTP 1.1 requests easily. More information on [the web page ↗](#). In ELE610 the `UiS` package `rwsuis` [↗](#) is mainly used for communication between Python and ABB robots. It uses `requests` together with the ABB package Robot Web Services `RWS` [↗](#). `rwsuis` is mainly used in RS5, but also the `appTATEM.py` program in RS4 uses `rwsuis` for reading RAPID variables. Documentation for the `rwsuis` package is available on web: [ABB Robot with Machine Vision ↗](#).



### **pyzbar**

Our preferred choice of QR scanner for Python is [pyzbar](#). We use this to locate pucks in assignment RS5.

### **bleak**

Bleak is an acronym for Bluetooth Low Energy platform Agnostic Klient. It is a GATT client software, capable of connecting to BLE devices acting as GATT servers. It is designed to provide a asynchronous, cross-platform Python API to connect and communicate with e.g. sensors. See [bleak for Python](#).

[GATT](#) is an acronym for the **Generic ATtribute** Profile, and it defines the way that two Bluetooth Low Energy devices transfer data back and forth using concepts called Services and Characteristics. This is used in RS4 in ELE610 as the communication platform in the `tatemCom*.py` program for data transfer from the TATEM tool.

### **cmd2**

[cmd2](#) is a powerful python library for building command-line interpreter (CLI) applications. It extends the `cmd` package in Python. In ELE610 `cmd2` is used as an interface in the Python program `tatemCom*` for communication between Python and the TATEM tool and ABB robot Rudolf in RS4.

### **dash** and **plotly**

Dash is the original low-code framework for rapidly building data apps in Python, i.e. when the program runs (on a server) a web-page is generated and can be accessed by a web-browser. Dash can be regarded as a blend of Flask (a lightweight WSGI, Web Server Gateway Interface, web application framework), [plotly](#) (a Graphing Library), and React.js (a JavaScript library for building user interfaces). See [dash on web](#). It is recommended to also install `pandas`. In ELE610 `dash`, together with `pandas` and `plotly`, may be used in RS4 in the dash-based application `appTATEM.py` to display graphical information from the JSON-file created by the TATEM tool.

### **pytest**

The `pytest` framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries. Can be used to do [unittest-based tests](#). See [pytest for Python](#). We will probably not use `pytest` in ELE610.

### **alive-progress**

A new kind of Progress Bar, with real-time throughput, ETA (expected time of arrival), and very cool animations according to [the installation web-page](#). We will probably not use `alive-progress` in ELE610.

### **Machine Learning (ML)**

[scikit-learn](#) (formerly `scikits.learn` and also known as `sklearn`) is a free comprehensive software **machine learning library** for the Python programming language. It is built on (uses and depends on) NumPy, SciPy, and

`matplotlib`. More specific information on features, installation, documentation and examples are on [scikit-learn web page ↗](#). It contains simple and efficient tools for classification, regression, clustering, dimensionality reduction, model selection, and data preprocessing. There are tutorials, user guides and many examples available.

You should note that also OpenCV contains a more limited machine learning module for statistical classification, regression and clustering of data. If you are new to ML you should consider to read the OpenCV tutorials on [Principal Component Analysis \(PCA\) ↗](#) and [Support Vector Machines \(SVM\) ↗](#). We don't use machine learning in ELE610.

### **TensorFlow and Keras**

[TensorFlow ↗](#) is particularly aimed at Neural Networks (NN). It is a popular and much used end-to-end open source platform for machine learning, and much used for learning of (deep) neural network like [convolutional neural network ↗](#) (CNN). The TensorFlow web pages are extensive and well designed and can be used both for beginners and experts.

[Keras ↗](#) is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation and providing a delightful developer experience. The purpose of Keras is to give an unfair advantage to any developer looking to ship ML-powered apps. We don't use TensorFlow or Keras in ELE610.

### **Numba**

[Numba ↗](#) is an open source JIT (Just In Time) compiler that translates a subset of Python and NumPy code into fast machine code. We don't use Numba in ELE610.