# R&D coding test

Estimated time: **3 hours**

DICOM handling and manipulation is paramount in order to develop proper computer vision or AI methodologies for medical image analysis. Python is currently the most extended programming language for AI, especially in medical imaging. Docker is a container technology that allows to run isolated instances of code with much lower resource consumption than a Virtual Machine.

The following exercise aims to test basic knowledge regarding DICOM, Python and Docker (optional). The code should be uploaded to a git repository (GitHub, Bitbucket, GitLab…) following GitFlow recommendations. A Python 3.X (recommended >= 3.6) module (dicomhandling) should be developed. The following tasks have to be considered:

1. The module should be able to be imported:

   ```
   import dicomhandling as dh
   ```

2. The module should implement a **DcmFilter** class (including the needed methods) that is able to:
   a. Read and store a DICOM file's pixel data as NumPy array.
   b. Use a gaussian 2D filter to smooth the image with a default sigma of 3, storing the resulting NumPy array.
   c. Read the ImagePositionPatient DICOM tag and store it as a 3 item list.

   Execution example:

   ```
   from dicomhandling import DcmFilter

   path = …
   sigma = 5

   dcm_filter = DcmFilter(path, sigma)

   # dcm_filter.original should contain the NumPy array with the original image
   # dcm_filter.filtered should contain the NumPy array with the filtered image
   # dcm_filter.ipp should store the 3 item list containing the
   ImagePositionPatient
   ```

3. The module should implement a **DcmRotate** class (including the needed methods) that is able to:
   a. Read and store a DICOM file's pixel data as NumPy array.
   b. Rotate the image (multiples of 90º with a default angle of 180º), storing the resulting NumPy array.
   c. Read the ImagePositionPatient DICOM tag and store it as a 3 item list.

   Execution example:

   ```
   from dicomhandling import DcmRotate

   path = …
   angle = 270
   ```

```
dcm_rotate = DcmRotate (path, angle)

# dcm_rotate.original should contain the NumPy array with the original image
# dcm_rotate.rotated should contain the NumPy array with the rotated image
# dcm_rotate.ipp should store the 3 item list containing the
ImagePositionPatient
```

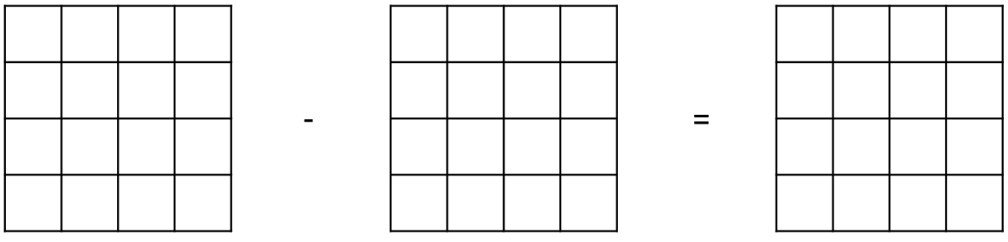
4. Efficient implementation of these classes will be positively evaluated.

5. The module should implement a **check_ipp** method that, given 2 objects of the classes defined before, is able to check if both have the same ImagePositionPatient (ipp) attribute value and store the result in a boolean variable (true or false).

   Execution example:

```
from dicomhandling import DcmFilter, DcmRotate, check_ipp

path_1 = …
path_2 = …
sigma = 5
angle = 270

dcm_filter = DcmFilter(path, sigma)
dcm_rotate = DcmRotate(path, angle)

same_ipp = check_ipp(dcm_filter, dcm_rotate)

# The method should accept any combination of two objects of these classes.
```

6. The main purpose of the module is to obtain the residue of a **voxelwise subtraction operation** applied to two input images, unfiltered and filtered. For this, the module should read the DICOM files present in a specific folder. The number of images in the folder should be exactly 2. If any other number of DICOM files is present, a custom exception **IncorrectNumberOfImages** should be raised and the execution aborted, printing the following: “Incorrect number of images. Aborting.”.
   If two DICOM files are present in the input folder, the module should read them and check if they have the same ImagePositionPatient. If that is the case, a custom exception **SameImagePositionPatient** should be raised, aborting and printing the following: “The DICOM files appear to be the same. Aborting.”. If the files don’t have the same ImagePositionPatient, the images have to be filtered using a 2D gaussian filter with a sigma of 3. Then, a voxelwise subtraction operation has to be applied to obtain the residue image. The subtraction operation has to be applied to both the unfiltered and filtered images, obtaining 2 residue images:



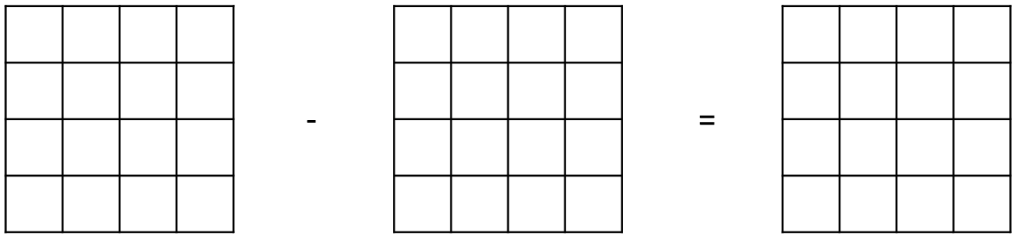



Image 01 filtered | Image 02 filtered | Filtered residue

**These residue images have to be stored as jpg images** showing the full dynamic range of the residue in a "residues" folder located inside the input folder. The module should be run as follows:

```
python3 -m dicomhandling input_folder
```

Where input folder is the path to the folder containing the two DICOM files.
**Tip:** since the module has to be run as if it were a script, dicomhandling.py should implement a "main" method with the needed operations to achieve the desired result.

7. **OPTIONAL:** create a DOCKERFILE with a recipe that allows to run the code inside a Docker container using an Ubuntu 20.04 base image.