Exercises Instructions

Contents

[Introduction 1](#_Toc434270669)

[Start of exercises 2](#_Toc434270670)

[Exercise 1: Data Simulation Brain 2](#_Toc434270671)

[Exercise 2: Data Simulation Thorax with PET 2](#_Toc434270672)

[Exercise 3: Data Simulation Thorax with SPECT 2](#_Toc434270673)

[Exercise 4: Preparation for Image reconstruction exercise 3](#_Toc434270674)

[Exercise 5: Image reconstruction part 1: EMML and OSEM 3](#_Toc434270675)

[Exercise 6: Image reconstruction part 2: adding Poisson noise 4](#_Toc434270676)

[Exercise 7: Image reconstruction part 3: MAP 4](#_Toc434270677)

[Exercise 8: Random estimation 5](#_Toc434270678)

[Exercise 9: Scatter Correction 5](#_Toc434270679)

[Exercise 10: Motion Correction 6](#_Toc434270680)

[Exercise 11: Image reconstruction part 4: PSF and MAP 7](#_Toc434270681)

[Appendices 7](#_Toc434270682)

[File extensions 8](#_Toc434270683)

[Python 8](#_Toc434270684)

[Image display outside of Python 9](#_Toc434270685)

[STIR commands for evaluation 9](#_Toc434270686)

# Introduction

This material is intended for practical demonstration using STIR on PET and SPECT Image Reconstruction.

Simulated data will be prepared during the exercises. These are based on 2 sets of images:

* Thorax phantom data are obtained from the recent open access article: *Tsoumpas et al 2013 Phys Med Biol*.   
  We have two respiratory gated positions of a thorax FDG PET phantom along with the corresponding CTAC image.
* Brain data are obtained from BrainWeb.  
  We have a segmented brain-map.

You will probably only want to run either the brain or the thorax data (except for the motion correction exercise which is currently only for the thorax).

The input data are stored in the folders called EX**\_\*, but you will need to run the scripts from the “main” STIR-exercises folders** (open a terminal, cd to where you extracted the exercises, and always cd back after every exercise).

We are using Python for some of the exercises. Python is an open-source interactive language, a bit like MATLAB. We provide Python scripts for the evaluation, so you should be fine. It would be best to read a Python tutorial before the course. We will use Spyder as our Python environment.

As an alternative to Python, we also provide instructions (in the sections “command line evaluation”) for loading sinograms and images in a display program. Note that in the text below we’re using AMIDE for display. ImageJ would work as well (see the end of the document).

See the appendices at the end of this document for some information to get started. ***Please read this before the course.***

Authors:

Elisabetta Grecchi, Irene Polycarpou, Charalampos Tsoumpas, Kris Thielemans

# Start of exercises

Open a terminal and type

cd ~/STIR-exercises

spyder&

You will then alternate between the terminal (to run shell scripts that execute STIR commands) and the spyder window (to run Python scripts that read in data and make plots).

**Note:**

The shell scripts (\*.sh) should all write “DONE” or “Done” at the end. If they didn’t, something went wrong. Check the log files (\*.log) in the output directory for the script that failed.

# Exercise 1: Data Simulation Brain

(Always run scripts from the STIR-exercises directory)

This is a simple simulation of a brain phantom. PSF is incorporated. Scatter is set to zero. Randoms are constant.

The aim of the exercise is get familiar with the scripts and evaluation environment, and to look at projection data in different ways (sinograms, by view etc)

Read and run script:

./run\_simulation\_brain.sh

## Python evaluation

Open the file /home/stir/STIR-exercises/evaluate\_simulation\_brain.py in Spyder (obviously adjust the path if you installed the exercises elsewhere). You can do this via its menus, or by typing in the terminal:

spyder evaluate\_simulation\_brain.py&Command line evaluation

You will need to extract the sinograms in an “image” Interfile to be able to load them in AMIDE

cd working\_folder/brain

extract\_segments my\_prompts.hs

Extract as SegmentByView (0) or BySinogram (1)?[0,1 D:0]: 1

Display the extracted sinogram (e.g. my\_promptsseg0\_by\_sino.hv) using **AMIDE**. On Linux, you can do

amide my\_promptsseg0\_by\_sino.hv&

Check the central sinogram plane to see if it looks as expected.

Go back to main directory

cd ../..

We can also extract profiles through the sinogram to display these in Excel or GNUmeric or similar. An example of this is given in the script

./evaluate\_simulation\_brain.sh

which will extract the segments and create profile text files for you. Output files will be

working\_folder/brain/profile\_prompts.txt

working\_folder/brain/profile\_randoms.txt

# Exercise 2: Data Simulation Thorax with PET

(Always run scripts from the STIR-exercises directory)

This is a simple simulation of a thorax phantom (2 gates). PSF is not incorporated. Scatter is simulated using STIR. Randoms are constant.

The aim of the exercise is first to do the same things as for the brain, to see differences in sinograms etc, get a first view on scatter, but then to move on to seeing the effect of motion.

Read and run script:

./run\_simulations\_thorax.sh

## Python evaluation

Start spyder with the evaluation script

spyder evaluate\_simulation\_thorax.py&

or if spyder is running, just open the file.

# Exercise 3: Data Simulation Thorax with SPECT

(Always run scripts from the STIR-exercises directory)

This is a simple simulation of single slice of a thorax phantom for SPECT. PSF is incorporated. Scatter is set to zero.

The aim of the exercise is first to see how PECT sinograms differ from PET.

Read and run script:

./run\_simulations\_SPECT.sh

Output is in working\_folder/single\_slice\_SPECT.

## Python evaluation

Start spyder with the evaluation script

spyder evaluate\_simulation\_SPECT.py&

or if spyder is running, just open the file.

## Command line evaluation

You will need to extract the sinograms in an “image” Interfile to be able to load them in AMIDE

cd working\_folder/GATE1/

extract\_segments my\_prompts\_g1.hs

TODO ENTER 1

Extract as SegmentByView (0) or BySinogram (1)?[0,1 D:0]: 1

cd ../GATE2/

extract\_segments my\_prompts\_g2.hs

Extract as SegmentByView (0) or BySinogram (1)?[0,1 D:0]: 1

cd ..

Import the extracted sinogram (e.g. my\_prompts\_g1seg0\_by\_sino.hv) using **AMIDE**.

Select and export the central sinogram plane and upload them online.

Subtract the two sinograms. This can be done in AMIDE or on the command line

stir\_subtract -s diff.hs GATE1/my\_prompts\_g1.hs GATE2/my\_prompts\_g2.hs

extract\_segments diff.hs

Extract as SegmentByView (0) or BySinogram (1)?[0,1 D:0]: 1

(Note: when using copy-paste of the above lines, please make sure that the “-s” option of stir\_subtract uses a minus sign. On some systems it copies as a character which looks the same but isn’t…)

How does the difference look like in sinogram space?

Go back to main directory

cd ..

We can also extract profiles through the sinogram to display these in Excel or similar. You could run

./evaluate\_simulation\_thorax.sh

to extract the segments and create profiles for you.

# Exercise 4: Preparation for Image reconstruction exercise

For the thorax reconstruction exercise, we first need to generate a new simulation data set. This time just a single slice to speed things up. Scatter is also set to zero for simplicity here.

./run\_simulation\_single\_slice.sh

Output is in working\_folder/single\_slice. There is no real need to look at the generated results as they are a single sinogram of the thorax simulation (but without scatter and with PSF).

# Exercise 5: Image reconstruction part 1: EMML and OSEM

(Always run scripts from the STIR-exercises directory)

You will need to have run the corresponding simulation script from the previous section. Output is in working\_folder/single\_slice or working\_folder/brain or working\_folder/single\_slice\_SPECT

We will now look at EMML and OSEM. A sample script is provided to generate results

./run\_reconstruction\_thorax.sh

Or

./run\_reconstruction\_brain.sh

Or

./run\_reconstruction\_SPECT.sh

This will first run FBP, EMML (also known as MLEM) for 240 iterations, then OSEM (with 8 subsets) for 240 sub-iterations. For both of these, the images obtained after every 24 updates will be saved. It will also continue EMML and OSEM from 240 sub-iterations and continue for 8 more, writing images at every subiterations. In addition, the EMML and OSEM images at 240 iterations are also post-filtered with a Gaussian filter.

Output is in working\_folder/single\_slice or working\_folder/brain or working\_folder/single\_slice\_SPECT *resp*.

The script runs the reconstruction on the noiseless simulations. See the next exercise for adding noise.

Sample questions to address:

* Is it worth running EMML? Why not simply use OSEM?
* At late iterations, is there a difference between OSEM and MLEM convergence behaviour?

## Python evaluation

Start spyder with the evaluation script

spyder evaluate\_reconstruction\_brain.py&

or if spyder is running, just open the file. If you have run the thorax or SPECT simulation, just adjust the path in Line 23 (or there abouts)

os.chdir('working\_folder/brain')

to the appropriate location.

## Command line evaluation

Run

./evaluate\_reconstruction\_brain.sh

(or choose evaluate\_reconstruction\_thorax.sh or evaluate\_reconstruction\_SPECT.sh).

The script generates some differences images and launches AMIDE. You can of course modify the script for yourself or type commands on the command line.

# Exercise 6: Image reconstruction part 2: adding Poisson noise

(You will have to run the simulation and reconstruction scripts first.)

We can make the simulation more realistic by adding noise to the data. An example would be (please *adjust the directory name to your case, e.g.* working\_folder*/brain* orworking\_folder/single\_slice\_SPECT)

cd working\_folder/single\_slice

# save noiseless data results in a subfolder

mkdir noiseless

cp \*.\* noiseless

poisson\_noise -p my\_noisy\_data.hs my\_prompts.hs 0.1 1

# overwrite the noiseless projection data with the noisy ones

cp my\_noisy\_data.hs my\_prompts.hs

cd ../..

For the SPECT exercise, replace my\_prompts.hs in the text above with my\_sim.hs.

Run poisson\_noise to understand what these arguments mean.

As we overwrite the data[[1]](#footnote-1), we can just use the same reconstruction and evaluation scripts as before (Exercise 5). An alternative would be to adjust the reconstruction parameter files to use your new noisy data (input) and change the filename used for the output, and change your evaluation scripts.

If you want to “reset” to the noiseless case, you can of course copy the data in noiseless back:

cd working\_folder/single\_slice

# save noisy data

mkdir noisy

cp \*.\* noisy

cp noiseless/\* .

cd ../..

Sample questions to address:

* Are the answers to the questions in Exercise 5 the same now that we added noise?

# Exercise 7: Image reconstruction part 3: MAP

This exercise needs results from the previous step (as the MAP reconstruction starts from an OSEM image in this exercise). Output is in working\_folder/single\_slice or working\_folder/brain or working\_folder/single\_slice\_SPECT *resp.*

We will now look at OSL and OSSPS with a Quadratic Prior. A sample script is provided to generate results

./run\_reconstruction\_thorax\_MAP.sh

or

./run\_reconstruction\_brain\_MAP.sh

or

./run\_reconstruction\_SPECT\_MAP.sh

This will run OSL and OSSPS (continuing from a previous OSEM image after 24 subiterations).

**Warning**:

If you added noise to the data (exercise 6), the MAP reconstructions will also use the noisy data of course. However, because of the initialization with OSEM, you want to run the normal reconstruction script (Exercise 5) first whenever you change the noise level.

You could for instance first run this exercise with the noisy data, then move the noiseless data back in place, then re-run the reconstruction script for the current exercise.

Sample questions to address:

* Do OSL and OSSPS generate the same results?
* Does this depend on the penalty factor? Noise level? Iteration number? Initialisation (try to remove the initial estimate for instance).

Start spyder with the evaluation script

spyder evaluate\_reconstruction\_brain\_MAP.py&

or if spyder is running, just open the file. If you have run the thorax or SPECT simulation, just adjust the path in Line 23 (or there abouts)

os.chdir('working\_folder/brain')

to the appropriate location.

# Exercise 8: Random estimation

This exercise uses Maximum Likelihood estimation to find singles from a sinogram of ‘delayed coincidences’. These estimated singles are then used to construct the randoms estimate.

Output is in working\_folder/randoms*.*

First run

./run\_randoms.sh

This will construct a noiseless randoms sinogram (by using ground-truth singles), add noise, and run the Maximum Likelihood estimation.

Sample questions to address:

* How do the estimated singles differ from the fansums? In which situations does this matter?
* Is there still noise in the estimated random sinogram? (You could do this by changing the seed for the Poisson noise generator and re-running the script).

Start spyder with the evaluation script to help you

spyder evaluate\_randoms.py&

or if spyder is running, just open the file.

# Exercise 9: Scatter Correction

(Always run scripts from the exercises directory)

There are 4 example scatter estimation scripts which can be used to investigate different questions about scatter.

**Run 0**

Ideal (i.e. correct attenuation map, scatter simulation matches with how the data was generated) scatter correction (using 3 scatter correction loops)

**Run 1**

Calculate scatter by using a smaller energy window than that simulated. This will demonstrate if the scaling technique works. We have selected 425keV for the lower energy window (original is 350keV).

**~~Run 2~~**(this is currently not available)

~~Scatter correction using the scatter estimation from the first gate for both gates. This will demonstrate how sensitive is scatter in choosing different but adjacent gates.~~

**Run 3**

Perform reconstruction & attenuation correction by using wrong attenuation map. In the particular exercise we have assigned bone attenuation value to lung attenuation value for the first gate. Then we use this wrong attenuation map located at the first gate to correct for attenuation and scatter for each gates.

You should run the scripts as folllows (you can try reading it but these scripts are relatively complicated):

./run\_scatter\_0.sh

The scripts make the following files in working\_folder/GATE1 (and similar in working\_folder/GATE2)

* input\_g1.hs: “measured” sinogram after randoms correction
* my\_scatter.hs: sinogram output of the simulation (i.e. ground truth)
* scatter\_estimate\_run0.hs etc: sinogram output of the iterative scatter estimation
* FDG\_g1.hv: input of the simulation (i.e. ground truth image)
* FBP\_recon\_with\_scatter\_correction\_run0.hv: FBP reconstruction of the scatter corrected data

Example questions to answer:

* How close is the scatter estimate in the ideal case of a simulation and how does this affect the image reconstruction? (run0)
* How different is the scatter (and its estimate) between gate 1 and gate 2? (run0)
* How different is the scatter (and its estimate) if you have a wrong estimate of the energy dependence of the detection efficiency? (run1)
* What happens to the scatter estimate and the reconstructed image if you use the wrong attenuation image? (run3)

## Python evaluation

2 Python scripts are provided as a starting point for investigating the results.

* evaluate\_scatter\_run0.py which reads results from run\_scatter0.sh and displays them comparing with the truth (i.e. simulation input and simulation scatter output) for GATE1
* evaluate\_scatter\_run3.py reads results from run0 and run3 and displays them (also for GATE1). You should be able to use this script with small modification to look at results from run 2. (make a copy, don’t overwrite the existing one).

## Evaluation using AMIDE

Use **AMIDE** to visualize your FBP\_recon\_with\_scatter\_correction\_run0.hv for each gate and the original simulated image. Use maximum display value 25.

Can you display the subtraction (e.g. using stir\_subtract) between the two gates?

How much motion do you see in the reconstructed images?

Extract sinograms for display with AMIDE for each of the two gates, e.g.:

cd working\_folder

extract\_segments GATE2/scatter\_estimate\_run0.hs

extract\_segments GATE2/my\_scatter\_g2.hs

Can you display the difference of the two sinograms e.g. using stir\_subtract –s? Can you see motion in the original sinograms? Scatter sinograms?

# Exercise 10: Motion Correction

(Always run scripts from the exercises directory. This exercise depends on the output of run\_simulations\_thorax.sh)

There are 2 scripts for Part I (MCIR vs noMC):

run\_MCIR\_0.sh

Correct for motion using valid motion vectors and the previously calculated scatter background  
 output folder: working\_folder/MCIR

run\_MCIR\_1.sh

Do not correct for motion   
 output folder: working\_folder/noMC

and 2 scripts for Part II (mismatched AC):

run\_MCIR\_2.sh

Correct for motion using valid motion vectors for emission and the previously calculated   
scatter background but use the same average attenuation map for both gates

output folder: working\_folder/MCIR/avAC

run\_MCIR\_3.sh

Correct for motion using valid motion vectors for emission and the previously calculated   
scatter background but use the same attenuation map   
(based on CTAC\_g1) for both gates  
output folder: working\_folder/MCIR/g1AC

run scripts, e.g.:

./run\_MCIR\_0.sh

./run\_MCIR\_1.sh

It takes about 30seconds to complete each reconstruction. If there are problems, you can confirm the scripts ran OK:

less working\_folder/MCIR/MCIR.log

less working\_folder/noMC/noMC.log

(quit less by pressing q)

To save some time, you can run Part II n the terminal while already evaluating Part I.

2 Python scripts are provided as a starting point for investigating the results: evaluate\_MCIR\_Part\_I.py and evaluate\_MCIR\_Part\_II.py.

Part I compares motion correction with no motion correction result and shows the forward motion & backward motion vector images.

Part II compares motion correction using the three different attenuation correction files (one for each gate, the one obtained from the average, the one obtained from gate 1)

Example questions to answer:

* How close is motion correction in the ideal case of a simulation and how does this affect

image reconstruction? (Part I)

- What type of motion is simulated? (Part I)

- How important is the use of the correct attenuation map? (Part II)

- Any comment on scatter correction? (Part II)

# Exercise 11: Image reconstruction part 4: PSF and MAP

This exercise needs results from exercises 5 and 7. So, you should already have done the following steps:

./run\_simulation\_SPECT.sh

# oaptionally add noise in Exercise 6

./run\_reconstruction\_SPECT.sh

./run\_reconstruction\_SPECT\_MAP.sh

We will now look at OSEM and OSSPS (with a Quadratic Prior) when PSF modelling is included in the reconstruction. We will only do this for SPECT as at present, STIR PSF modelling is PET is hard to modify. A script is provided to generate results

./run\_reconstruction\_SPECT\_PSF.sh

This will run OSEM and OSSPS (continuing from a previous OSEM image after 24 subiterations) with PSF model (check e.g. OSEMPSF.par).

Output is in working\_folder/single\_slice\_SPECT.

**Warning**:

If you added noise to the data (exercise 6), the MAP reconstructions will also use the noisy data of course. See the MAP exercise for more info.

Sample questions to address:

* Does PSF-modelling increase resolution?
* Do you see edge-effects or overshoots in the reconstructed images? For both OSEM and OSSPS or only OSEM? Why?
* What happens to the noise “texture” of the images if you reconstruct with noisy data. Are overshoots worse?
* Extension: you could try to underestimate the PSF (edit OSEMPSF.par and change the collimator modelling and re-run the reconstruction script).

Start spyder with the evaluation script

spyder evaluate\_reconstruction\_SPECT\_PSF.py&

or if spyder is running, just open the file.

# Appendices

## Installing STIR

You will need the updated source made available via the web for this course (see the STIR website). You also need a display program, we recommend AMIDE. You will then need to unpack the zip file and add the directory to your path such that they can be found when typing a command.

If you have Linux (or vGate), use the installation script. Otherwise you need to build STIR yourself, install it, and add it to your path

Open a terminal and type something like this all on one line (adjust to where your files are):

* Linux/MacOS:  
  PATH=~/STIRShortCourse/bin/:$PATH
* Cygwin  
  PATH=~/STIRShortCourse/bin/:/cygdrive/c/Program\ Files\ \(x86\)/amide/bin/:$PATH
* Windows command terminal (if Cygwin not available)  
  PATH %HOMEDRIVE%%HOMEPATH%\Documents\STIRShortCourse\bin;c:\Program Files x86\amide\bin;%PATH%

After installing, try to type in your terminal

forward\_project

You should see a usage message. If you get an error, you probably didn’t set-up your path correctly.

## File extensions

.hv: Interfile header for image (volume)

.ahv: (ignore) old-style Interfile header for image

.v: raw data of image (in floats)

.hs: Interfile header projection data (sinograms)

.s: raw data of projection data (in floats)

.par: STIR parameter file

.sh: Shell script (sequence of commands)

.bat: Windows batch file

.log: log file (used to record output of command)

.py: Python file

## Linux Terminal

If you have never used a Linux/Unix terminal before, have a look at a tutorial at <https://help.ubuntu.com/community/UsingTheTerminal>.

You can use UPARROW to go to previous commands, and use copy-paste shortcuts Left-CTRL-SHIFT-C and Left-CTRL-SHIFT-V.[[2]](#footnote-2)

## Python

We use Spyder as a nice Integrated Development Environment (IDE) for Python (actually iPython which is a slightly friendlier version of Python). You need only minimal knowledge of Python for this course, but it would be good to read-up a bit (see below).

You will normally work by loading an example script in Spyder in the editor, executing it bit by bit, and then editing it to do some more work. Useful shortcuts for in the editor (these are in Windows-style, including the usual copy-paste shortcuts Left-CTRL-C and Left-CTRL-V).[[3]](#footnote-3)

* F9 executes the currently highlighted code.
* LEFT-CTRL + <RETURN> executes the current cell (menu entry Run -> Run cell). A cell is defined as the code between two lines which start with the agreed tag #%%.
* SHIFT + <RETURN> executes the current cell and advances the cursor to the next cell (menu entry Run -> Run cell and advance).
* TAB tries to complete the word/command you have just typed

And here are some useful ipython "magic" commands that you can use in the ipython console on the right (but not in the scripts). Most of these are identical to what you would use in the terminal.

* change to a new directory  
  cd some\_dir/another\_subdir
* change back 2 levels up  
  cd ../..
* print current working directory  
  pwd
* edit a file  
  edit FBP.par
* list files in current directory  
  ls \*.hs
* Running system commands from the ipython prompt can be done via an exclamation mark  
  !FBP2D FBP.par
* Get rid of everything in memory  
  %reset

One thing which might surprise you that in Python *indentation is important[[4]](#footnote-4).* You would write for instance

for z in range(0,image.shape[0]):

plt.figure()

plt.imshow(image[z,:,:])

# now do something else

The Spyder Integrated Development Environment (IDE) has of course lots of parameters which you can tune to your liking. The main setting that you might want to change is if the graphics are generated “inline” in the iPython console, or as separate windows. Go to Tools > Preferences > iPython console > Graphics > Graphics backend. Change from “inline” to “automatic” if you prefer the separate windows.

Here is some suggested material on Python (ordered from easy to quite time-consuming).

* The official Python tutorial. Just read Section 1, 3, a bit of 4 and a tiny bit of 6.  
  <https://docs.python.org/2/tutorial/>
* Examples for matplotlib, the python module that allows you to make plots almost like in MATLAB  
  <https://github.com/patvarilly/dihub-python-for-data-scientists-2015/blob/master/notebooks/02_Matplotlib.ipynb>
* You could read bits and pieces of Python the Hard Way  
  <http://learnpythonthehardway.org/book/index.html>
* Google has an online class on Python for those who know some programming. This goes quite in depth and covers 2 days.  
  <https://developers.google.com/edu/python/?csw=1>

## Image display outside of Python

Several display programs can be used. AMIDE reads the interfile volumes directly. ImageJ and others can use import of raw floats (i.e. the .v file).Settings are for instance.

Image type: 32-bit Real

Width ?

Height: ?

Offset: 0

Number of images ?

Gap between images: 0

White is 0: Ticked

Little endian: Ticked

You will have to find the data sizes from the header (the .hv file), or by using list\_image\_info.

## STIR commands for evaluation

These are a few STIR commands that can be used on the command prompt (not Python). You normally don’t need to if you use the Python scripts though.

For nearly all of these, you can just type the command name without arguments for a usage message.

### Basic information about geometry

list\_projdata\_info projdata.hs

list\_image\_info image.hv

### Image reconstruction

OSMAPOSL someParameterFile.par

FBP2D somePparameterFile.par

OSSPS someParameterFile.par

These are the STIR executables used for reconstruction. In the text above, they are called by the shells scripts.

### Profile extraction

list\_image\_values prof.txt input\_image \

min\_plane max\_plane min\_row max\_row min\_col max\_col

(note: the backslash “\” is used in shell scripts for “line continuation”, i.e. when everything does not fit on one line)

list\_image\_values writes values to a text file (for import in Excel et al).

Indices need to be in the STIR convention (plane starts from 0, col,row are

centred around 0). Use list\_image\_info to find ranges.

Note: there is currently a bug in list\_image\_values that row (x) and column (y)

have to be given in that order (i.e. it's z,x,y while should have been z,y,x)

### Conversion of projection data to an image for display

extract\_segments projdata.hs

convert projection data into an (Interfile) image e.g. for display via AMIDE (as no standard display program reads in sinogram data).

1. Note that we only copy the header (.hs) file as this header contains a field that “points” to the raw float data (.s). [↑](#footnote-ref-1)
2. On Windows and Linux, VirtualBox sets the “host-key” by default to Right-CTRL, so unless you change this, you have to use Left-CTRL to “send” the CTRL-keystroke to the Virtual Machine. [↑](#footnote-ref-2)
3. Text from <http://www.southampton.ac.uk/~fangohr/blog/spyder-the-python-ide.html> [↑](#footnote-ref-3)
4. The amount of indentation is not important, as long as you are consistent (i.e. it doesn’t matter if you use 2 or 4 spaces, but you cannot mix them). [↑](#footnote-ref-4)