10 Running

10.1 Introduction

10.1.1 About the code

In this tutorial you will find four different types of codes. First, there is python code, possibly in a jupyter notebook. Python is a easily interpretable code and is both easily accessible to new users as well as quick to implement. Especially, when combined with linear algebra package 'numpy' and plotting package 'matplotlib' is a powerful tool for data science. However, since it is not compiled code, python is relatively slow. The second type of code python with 'numba' is a way to perform "Just-In-Time compilation" - or JIT. 'numba' reads python code and compiles it just before running, recovering a lot of the lost efficiency, often resulting in about 10 times higher efficiency than plain numpy. In some cases, like when running simulations, even 'numba' implementations can be limiting in its efficiency. That is why, in computational chemistry, we often work with low level codes as c, c++ and fortran. In this course we use c++ and therefore c++ libraries are made accessible to python as the third type of code you will find. Lastly, you will also find external code packages, where we don't expect you to make alterations to the code, but we only require you to alter input files and create analysis routines in the output files.

10.1.2 Youtube recommended videos

If you are unfamiliar or inexperienced with programming and/or python, we can recommend to view some of the following youtube video series before the molsim-school.

Harvard CS50's Introduction to Programming with Python	David J. Malan	link
Python tutorial for beginners	Corey Schafer	link
Intermediate Python Programming Course	freeCodeCamp.org	link
Linux/Mac Terminal Tutorial	Corey Schafer	link
The Shell	Missing Semester	link
Absolute BEGINNER Guide to the Mac OS Terminal	Percy Grunwald	link
Visual Studio Code Crash Course	James Q Quick	link
Visual Studio Code Tutorial for Beginners	Academind	link
Visual Studio Code Intro & Setup	Traversy Media	link

10.1.3 Jupyter notebooks

Jupyter notebooks are a very convenient way to work with computer ode. If you've used Mathematica or similar software, they take the same philosophy. You have "cells" where you can input code, and you run each of those cells. You can also have special cells to add text, so you can document what the notebook does. The big difference between Mathematica and Jupyter is that, whereas Mathematica only works with its own language, Jupyter works with many different languages. For all exercises in week one, you will use jupyter notebooks to run python, numba or c++ library code, and in the same notebook analyse and plot your results.

10.1.4 Numba

Numba translates Python functions to optimized machine code at runtime using the industry-standard LLVM compiler library. Numba-compiled numerical algorithms in Python can approach the speeds of C or FORTRAN. You do not need to replace the Python interpreter, run a separate compilation step, or even have a C/C++ compiler installed. In Numba you can just apply one of the Numba decorators to a Python function, and Numba does the rest. Numba is designed to be used with NumPy arrays and functions. Numba generates specialized code for different array data types and layouts to optimize performance.

However, 'numba' can sometimes be slightly more difficult to work with as the error messages are not very descriptive.

10.1.5 Conda

`Conda` is an open source package and environment manager. `Miniconda` is a Python distribution that only includes `Python`, `conda`, their dependencies, and a few other useful packages. `Mamba` is a re-implementation of the `conda` package manager in `C++`. As a package manager, you can use `conda` to install, update, and remove packages and their "dependencies" (the packages they depend upon). Package installation in `conda` is predictably easy because you're installing pre-compiled binaries. As an environment manager, you can use `conda` to manage virtual environments. Virtual environments allow you to maintain isolated environments with different packages and versions of those packages.

10.1.6 `molsim` environment

The `molsim` virtual environment is a named, isolated, working copy of Python that that maintains its own files, directories, and paths so that you can work with specific versions of libraries or Python itself without affecting other Python projects. Virtual environments make it easy to cleanly separate different projects and avoid problems with different dependencies and version requirements across components.

The `molsim` environment is based on:

- python=3.12
  Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages

such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

- ipykernel
  A powerful interactive Python shell and a Jupyter kernel to work with Python code in Jupyter notebooks and other interactive frontends. Each kernel can be associated with a separate virtual or Conda environment. By associating a Jupyter notebook with a specific kernel, you can ensure that the code runs in the exact same environment every time.

- ipywidgets >= 7.0
  ipywidgets, also known as jupyter-widgets or simply widgets, are interactive HTML widgets for Jupyter notebooks and the IPython kernel.

- jupyterlab
  JupyterLab is the next-generation user interface for Project Jupyter. It offers all the familiar building blocks of the classic Jupyter Notebook (notebook, terminal, text editor, file browser, rich outputs, etc.) in a flexible and powerful user inteface. Eventually, JupyterLab will replace the classic Jupyter Notebook.

- notebook >= 4.2
  The Jupyter notebook is a web-based notebook environment for interactive computing.

- matplotlib
  matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in Python scripts, the Python and IPython shell.

- numpy
  NumPy is the fundamental package needed for scientific computing with Python.

- scipy
  SciPy is a Python-based ecosystem of open-source software for mathematics, science, and engineering.

- pybind11
  pybind11 is a lightweight header-only library that exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code. Its goals and syntax are similar to the excellent Boost.Python library by David Abrahams: to minimize boilerplate code in traditional extension modules by inferring type information using compile-time introspection.

- clang
  Development headers and libraries for Clang

- numba
  Numba is an Open Source NumPy-aware optimizing compiler for Python sponsored by Anaconda, Inc. It uses the remarkable LLVM compiler infrastructure to compile Python syntax to machine code.

- pymatgen
  Pymatgen (Python Materials Genomics) is a robust, open-source Python library for materials analysis. Highly flexible classes for the representation of Element, Site, Molecule, Structure objects. Extensive input/output support, including support for VASP, ABINIT, CIF, Gaussian, XYZ, and many other file formats.

- nglview
  An IPython widget to interactively view molecular structures and trajectories. Utilizes the embeddable NGL Viewer for rendering.

- ase
  Atomic Simulation Environment (ASE) is a set of tools and Python modules for setting up, manipulating, running, visualizing and analyzing atomistic simulations.

- black
  Black is the uncompromising Python code formatter. By using it, you agree to cease control over minutiae of hand-formatting. In return, Black gives you speed, determinism, and freedom from pycodestyle nagging about formatting.

- mdanalysis
  MDAnalysis is a Python library to analyze trajectories from molecular dynamics (MD) simulations. It can read and write most popular formats, and provides a flexible and fast framework for writing custom analysis through making the underlying data easily available as NumPy arrays.

- llvm-openmp
  The OpenMP API supports multi-platform shared-memory parallel programming in C/C++ and Fortran.

10.1.7 `ops_env` environment

The `ops_env` environment (for the OpenPathSampling tutorial day) is based on:

- python=3.9
  A powerful interactive Python shell and a Jupyter kernel to work with Python code in Jupyter notebooks and other interactive frontends. Each kernel can be associated with a separate virtual or Conda environment. By associating a Jupyter notebook with a specific kernel, you can ensure that the code runs in the exact same environment every time.

- openmm=8.0.0
  OpenMM is a toolkit for molecular simulation. It can be used either as a stand-alone application for running simulations, or as a library you call from your own code. It provides a combination of extreme flexibility (through custom forces and integrators), openness, and high performance (especially on recent GPUs) that make it truly unique among simulation codes. OpenMM is MIT licensed with some LGPL portions (CUDA and OpenCL platforms).

- openpathsampling=1.5.2
  OpenPathSampling is a library for trajectory-based rare events simulations, including transition path sampling, transition interface sampling, and committor simulations. Note that for

real-world usage, you must also install one of the powerful molecular dynamics engines that OPS interfaces with, such as OpenMM or Gromacs. If you install those seprately, OPS will automatically support your installation of them.

- openmmtools
  openmmtools is a Python library layer that sits on top of OpenMM to provide access to a variety of useful tools for building full-featured molecular simulation packages.

- mdtraj
  MDTraj is a python library that allows users to manipulate molecular dynamics (MD) trajectories and perform a variety of analyses, including fast RMSD, solvent accessible surface area, hydrogen bonding, etc. A highlight of MDTraj is the wide variety of molecular dynamics trajectory file formats which are supported, including RCSB pdb, GROMACS xtc, tng, and trr, CHARMM / NAMD dcd, AMBER binpos, AMBER NetCDF, AMBER mdcrd, TINKER arc and MDTraj HDF5.

- nglview
  An IPython widget to interactively view molecular structures and trajectories. Utilizes the embeddable NGL Viewer for rendering.

- jupyter
  Jupyter metapackage. Install all the Jupyter components in one go. Includes ipykernel, ipywidgets, ipython, jupyterlab, and notebook.

- tqdm
  A Fast, Extensible Progress Bar for Python and CLI.

10.2 Running locally on macOS

10.2.1 The Homebrew package manager

Homebrew is the easiest and most flexible way to install the UNIX tools Apple didn't include with macOS. It can also install software not packaged for your Unix distribution without requiring sudo. The Homebrew website can be found at https://brew.sh. Install homebrew using

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Install some packages that we will need

```
brew install gnuplot gcc wget pybind11
```

10.2.2 Install Visual Studio Code

1. Download Visual Studio Code for macOS.
2. Open the browser's download list and locate the downloaded app or archive.
3. If archive, extract the archive contents. Use double-click for some browsers or select the 'magnifying glass' icon with Safari.
4. Drag Visual Studio Code.app to the Applications folder, making it available in the macOS Launchpad.
5. Open VS Code from the Applications folder, by double clicking the icon.
6. Add VS Code to your Dock by right-clicking on the icon, located in the Dock, to bring up the context menu and choosing Options, Keep in Dock.

If you want to run VS Code from the terminal by simply typing 'code', VS Code has a command, Shell Command: Install 'code' command in PATH, to add 'code' to your $PATH variable list. After installation, launch VS Code. Now open the Command Palette (⌘ + Shift ⇧ + P) and type shell command to find the Shell Command: Install 'code' command in PATH command. After executing the command, restart the terminal for the new $PATH value to take effect. You'll be able to simply type 'code .' in any folder to start editing files in that folder.
Note: Install the VS Code extensions listed in Table 11.1.

10.2.3 Installing conda

On the mac, install and use the homebrew package manager to install mamba

```
  brew install micromamba
  micromamba shell init --shell zsh --root-prefix=~/.micromamba
```

You should see

```
  Preparing transaction: done
  Verifying transaction: done
  Executing transaction: done
```

10.2.4 Creating the `molsim` virtual environment

```
cd Molsim-source-exercises
micromamba env create --file=env.yml
micromamba activate molsim
```

The `molsim` environment sets the python version to 3.12 (3.13 does not work yet).

10.2.5 Install the `molsim` package

```
cd Molsim-source-exercises
micromamba activate molsim
pip install .
```

The output should be

```
Successfully built molsim
Installing collected packages: molsim
Successfully installed molsim-1.0.0
```

10.2.6 Running the notebooks

Now that we have created the environment we can test this by running it on notebooks. Since we run the notebooks from the `carbon` server, we should first make sure that we connect to the notebook correctly. A jupyter notebook is very similar to an html page and has an address a browser can reach. There are two ways to connect to this jupyter notebook.

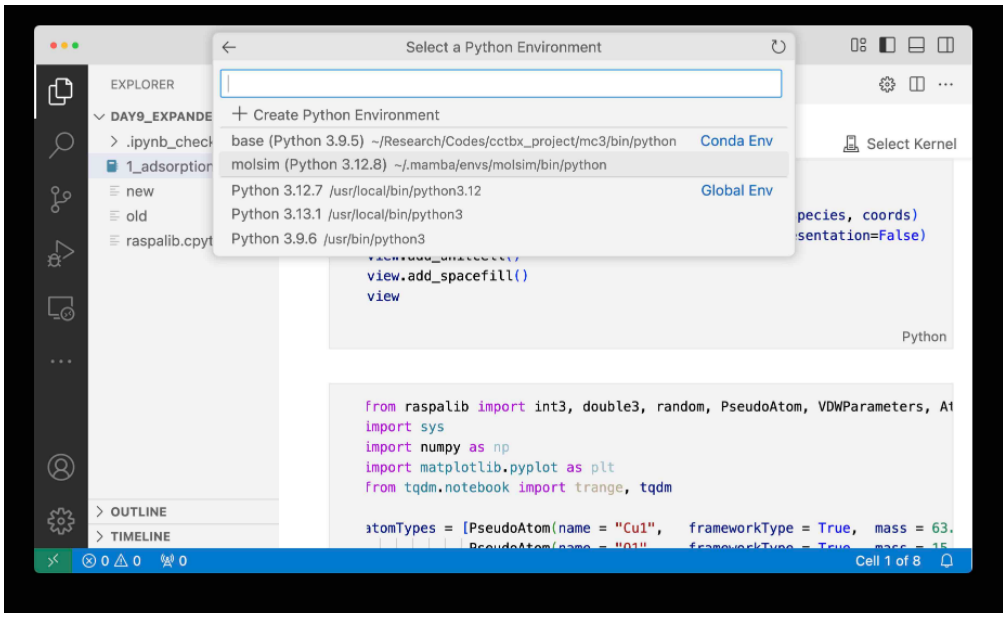
There are two ways of running the notebooks:

1. Using `Visual Code`

   ```
   micromamba activate molsim
   cd Molsim-source-exercises
   code .
   ```

2. Using `Jupyter Lab`

   ```
   micromamba activate molsim
   cd Molsim-source-exercises
   jupyter lab
   ```



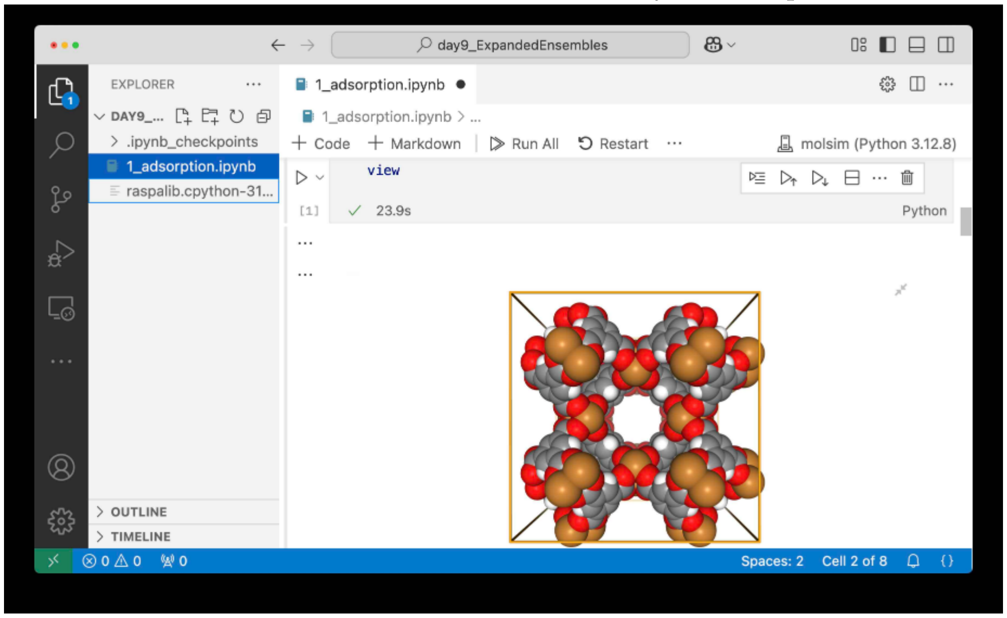
Create Python Environment
base (Python 3.9.5) ~/Research/Codes/cctbx_project/mc3/bin/python Conda Env
molsim (Python 3.12.8) ~/.mamba/envs/molsim/bin/python
Python 3.12.7 /usr/local/bin/python3.12 Global Env
Python 3.13.1 /usr/local/bin/python3
Python 3.9.6 /usr/bin/python3

```
pecies, coords)
sentation=False)
view.add_spacefill()
view
```

```
from raspalib import int3, double3, random, PseudoAtom, VDWParameters, At
import sys
import numpy as np
import matplotlib.pyplot as plt
from tqdm.notebook import trange, tqdm

atomTypes = [PseudoAtom(name = "Cu1",   frameworkType = True,  mass = 63.
```

(a) Select the 'molsim' environment as the Python Interpreter.



```
view
```

[1] ✓ 23.9s

(b) Example of NGLview to view atomic sytructures.

Figure 10.1: Running VC Code in macOS

10.3 Running locally on Linux

Instructions are for Debian-based linux systems like Ubuntu.

10.3.1 Install required and useful linux packages

Install packages that we will need:

```
  sudo apt install bzip2 wget gpg curl git build-essential
  sudo apt install gnuplot-qt pybind11-dev
```

Install micromamba using the following procedure:

```
wget https://github.com/mamba-org/micromamba-releases/releases/latest/
   ↪ download/micromamba-linux-64.tar.bz2
bunzip2 micromamba-linux-64.tar.bz2
tar -xvf micromamba-linux-64.tar
./bin/micromamba shell init -s bash --root-prefix=~/.micromamba
source ~/.bashrc
```

10.3.2 Install Visual Studio Code

```
sudo apt-get install wget gpg
wget -qO- https://packages.microsoft.com/keys/microsoft.asc | gpg
   ↪ --dearmor > packages.microsoft.gpg
sudo install -D -o root -g root -m 644 packages.microsoft.gpg /etc/apt/
   ↪ keyrings/packages.microsoft.gpg
echo "deb [arch=amd64,arm64,armhf signed-by=/etc/apt/keyrings/packages.
   ↪ microsoft.gpg] https://packages.microsoft.com/repos/code stable
   ↪ main" |sudo tee /etc/apt/sources.list.d/vscode.list > /dev/null
rm -f packages.microsoft.gpg
sudo apt install apt-transport-https
sudo apt update
sudo apt install code
```

Note: Install the VS Code extensions listed in Table 11.1.

10.3.3 Creating the `molsim` virtual environment

```
    cd Molsim-source-exercises
    micromamba env create --file=env.yml
    micromamba activate molsim
```

The `molsim` environment sets the python version to 3.12 (3.13 does not work yet), add some package channels to locate the packages and install de dependencies: `ipykernel`, `jupyterlab`, `matplotlib`, `numpy`, `scipy`, `pybind11`, `clang`, `numba`, `pymatgen`, `notebook`, `ipywidgets`, `nglview`, and ase.

10.3.4 Install the `molsim` package

```
cd Molsim-source-exercises
micromamba activate molsim
pip install .
```

The output should be

```
Successfully built molsim
Installing collected packages: molsim
Successfully installed molsim-1.0.0
```

10.3.5 Running the notebooks

Now that we have created the environment we can test this by running it on notebooks. Since we run the notebooks from the `carbon` server, we should first make sure that we connect to the notebook correctly. A jupyter notebook is very similar to an html page and has an address a browser can reach. There are two ways to connect to this jupyter notebook.

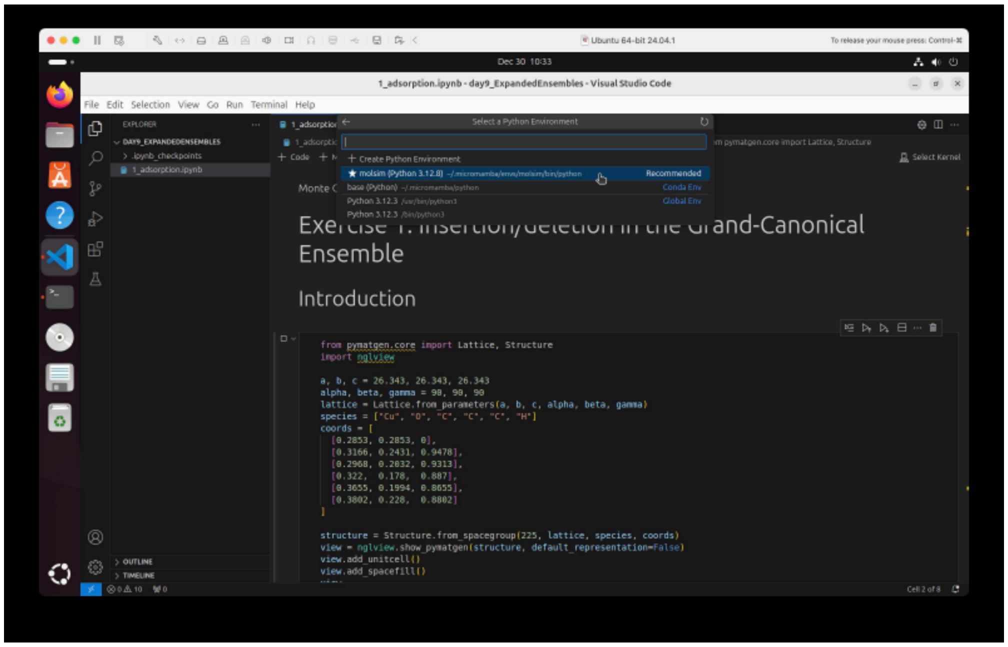
There are two ways of running the notebooks:

1. Using `Visual Code`

   ```
   micromamba activate molsim
   cd Molsim-source-exercises
   code .
   ```

2. Using `Jupyter Lab`

   ```
   micromamba activate molsim
   cd Molsim-source-exercises
   jupyter lab
   ```


Ubuntu 64-bit 24.04.1
To release your mouse press: Control-⌘
Dec 30 10:33
1_adsorption.ipynb - day9_ExpandedEnsembles - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
DAY9_EXPANDEDENSEMBLES
.ipynb_checkpoints
1_adsorption.ipynb
Select a Python Environment
+ Create Python Environment
★ molsim (Python 3.12.8) ~/.micromamba/envs/molsim/bin/python Recommended
base (Python) ~/.micromamba/python Conda Env
Python 3.12.3 /usr/bin/python3 Global Env
Python 3.12.3 /bin/python3
Select Kernel
OUTLINE
TIMELINE
Cell 2 of 8

Exercise 1: Insertion/deletion in the Grand-Canonical Ensemble

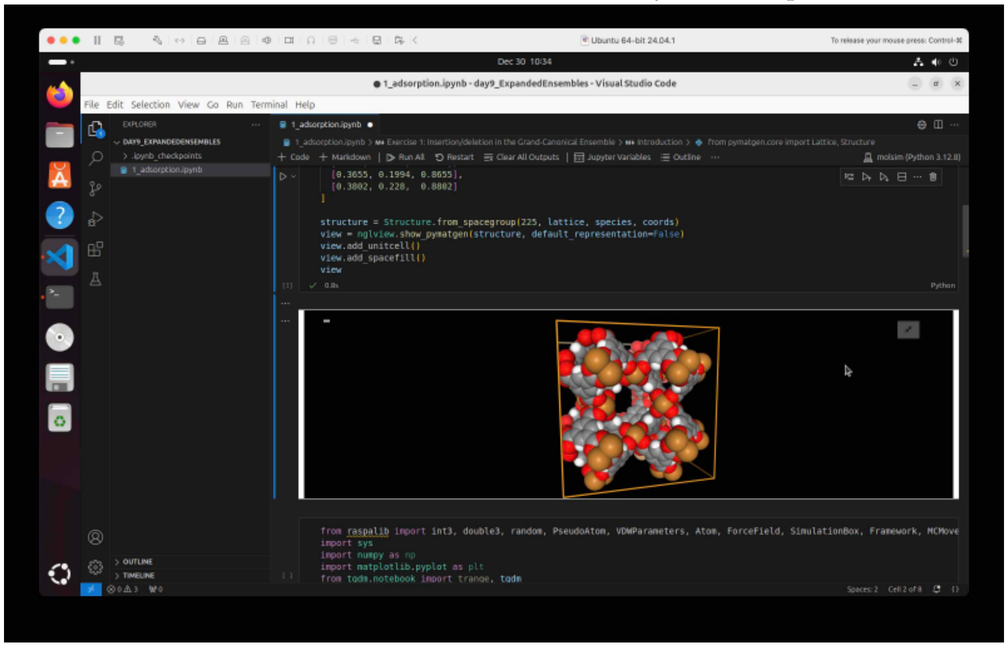
Introduction

```
from pymatgen.core import Lattice, Structure
import nglview

a, b, c = 26.343, 26.343, 26.343
alpha, beta, gamma = 90, 90, 90
lattice = Lattice.from_parameters(a, b, c, alpha, beta, gamma)
species = ["Cu", "O", "C", "C", "C", "H"]
coords = [
    [0.2853, 0.2853, 0],
    [0.3166, 0.2431, 0.9478],
    [0.2968, 0.2032, 0.9313],
    [0.322,  0.178,  0.887],
    [0.3655, 0.1994, 0.8655],
    [0.3802, 0.228,  0.8802]
]

structure = Structure.from_spacegroup(225, lattice, species, coords)
view = nglview.show_pymatgen(structure, default_representation=False)
view.add_unitcell()
view.add_spacefill()
```

(a) Select the 'molsim' environment as the Python Interpreter.


Ubuntu 64-bit 24.04.1
To release your mouse press: Control-⌘
Dec 30 10:34
● 1_adsorption.ipynb - day9_ExpandedEnsembles - Visual Studio Code
File Edit Selection View Go Run Terminal Help
1_adsorption.ipynb > Exercise 1: Insertion/deletion in the Grand-Canonical Ensemble > Introduction > from pymatgen.core import Lattice, Structure
+ Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline
molsim (Python 3.12.8)
Python
Spaces: 2 Cell 2 of 8

```
    [0.3655, 0.1994, 0.8655],
    [0.3802, 0.228,  0.8802]
]

structure = Structure.from_spacegroup(225, lattice, species, coords)
view = nglview.show_pymatgen(structure, default_representation=False)
view.add_unitcell()
view.add_spacefill()
view
```

```
from raspalib import int3, double3, random, PseudoAtom, VDWParameters, Atom, ForceField, SimulationBox, Framework, MCMove
import sys
import numpy as np
import matplotlib.pyplot as plt
from tqdm.notebook import trange, tqdm
```

(b) Example of NGLview to view atomic structures.

Figure 10.2: Running VC Code in linux

10.4 Running locally on Windows

10.4.1 Requirements

- Minimum windows 10 version is the Anniversary Update (Version 1607),
- 64-bit version of Windows 10

10.4.2 Turn on Developer Mode (not needed on recent versions)

- Go to "Start menu"
- Goto "Settings"
- Search for "developer settings"
- Turn on "Developer Mode"

10.4.3 Install Windows Subsystem for Linux (WSL)

The procedure of installing WSL depends on whether the computer has a CPU with Hyper-V Virtualization support or not:

- Computers that do *not* have a CPU with Hyper-V Virtualization support are only capable of running WSL1.
  - Go to "Control Panel"
  - Goto "Programs" and "Program and Features"
  - Goto "Turn Windows features on or off"
  - Select "Windows Subsystem for Linux"
  - Reboot if required

  Open a powershell and run

  ```
  wsl --set-default-version 1
  ```

- Computers that have a CPU with Hyper-V Virtualization support are capable of running WSL2. WSL 2 is a major overhaul of the underlying architecture and uses virtualization technology and a Linux kernel to enable new features. WSL 2 is faster, more versatile, and uses a real Linux kernel. The primary goals of this update are to increase file system performance and add full system call compatibility. WSL 2 is only available in Windows 10, Version 1903, Build 18362 or higher.
  - Go to "Control Panel"
  - Goto "Programs" and "Program and Features"
  - Goto "Turn Windows features on or off"
  - Select "Windows Subsystem for Linux"

  - Select “Virtual Machine Platform”
    "Enables platform support for virtual machines" and is required for WSL2.
  - Select “Hyper-V”
    A computer with Hyper-V Virtualization support is required for WSL2.
  - Reboot if required

Open a powershell and run

```
wsl --set-default-version 2
```

10.4.4 Install Ubuntu 24 in WSL

Install the Ubuntu linux distribution in WSL by entering in a windows console/powershell:

```
wsl --install -d ubuntu
```

Once installation is complete, you will be prompted to create a new user account (and its password).
Creating your Linux user is the first step in setting up a new Linux distribution on WSL. The first user account you create is automatically configured with a few special attributes:

1. It is your `default user` – it signs-in automatically on launch.
2. It is Linux administrator (a member of the `sudo` group) by default.

Each Linux distribution running on the Windows Subsystem for Linux has its own Linux user accounts and passwords. You will have to configure a Linux user account any time you add a distribution, reinstall, or reset. Linux user accounts are not only independent per distribution, they are also independent from your Windows user account.
The linux account does not have to be the same as your Windows account.

- Enter a username in the required field and press Enter (you can not use the username “admin”)
- Enter a password (twice)

Update the distribution and install some required packages

```
sudo apt-get update
sudo apt-get upgrade
sudo apt install bzip2 wget gpg curl git build-essential
sudo apt install gnuplot-qt pybind11-dev
```

```
wget https://github.com/mamba-org/micromamba-releases/releases/latest/
   ↪ download/micromamba-linux-64.tar.bz2
bunzip2 micromamba-linux-64.tar.bz2
tar -xvf micromamba-linux-64.tar
./bin/micromamba shell init -s bash --root-prefix=~/.micromamba
source ~/.bashrc
```

10.4.5 Install Visual Studio Code

- Download the Visual Studio Code installer for Windows.
- Once it is downloaded, run the installer (VSCodeUserSetup-version.exe). This will only take a minute.
- By default, VS Code is installed under `C:\Users\{Username}\AppData\Local\Programs\Microsoft VS Code`.

Setup will add Visual Studio Code to your `%PATH%`, so from the console you can type '`code .`' to open VS Code on that folder. You will need to restart your console after the installation for the change to the `%PATH%` environmental variable to take effect.
Note: Install the VS Code extensions listed in Table 11.1.

10.4.6 Creating the `molsim` virtual environment

Open the `Ubuntu` app and execute the following commands in the linux terminal

```
cd Molsim-source-exercises
micromamba env create --file=env.yml
micromamba activate molsim
```

The `molsim` environment sets the python version to 3.12 (3.13 does not work yet), add some package channels to locate the packages and install de dependencies: `ipykernel`, `jupyterlab`, `matplotlib`, `numpy`, `scipy`, `pybind11`, `clang`, `numba`, `pymatgen`, `notebook`, `ipywidgets`, `nglview`, and `ase`.

10.4.7 Install the `molsim` package

Open the `Ubuntu` app and execute the following commands in the linux terminal

```
cd Molsim-source-exercises
micromamba activate molsim
pip install .
```

The output should be

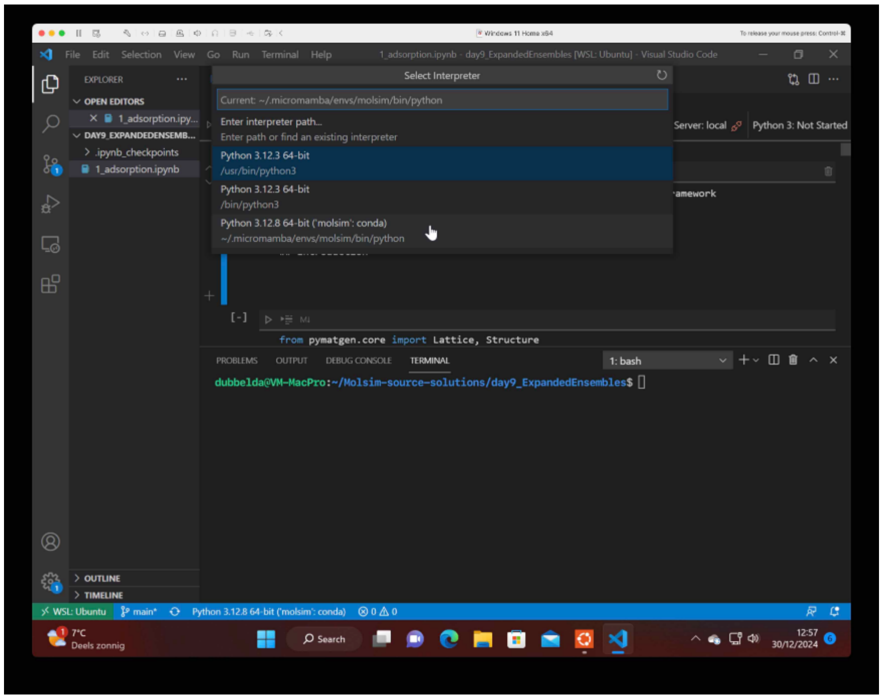
```
Successfully built molsim
Installing collected packages: molsim
Successfully installed molsim-1.0.0
```

10.4.8 How to access your files?

The windows files are accessible from WSL at

```
/mnt/c/Users/<username>
```

Any change made from the linux side will immediately show up in windows. Note that from the terminal you can open a file explorer using the command (note the back-ticks around the `wslpath` command):


Select Interpreter

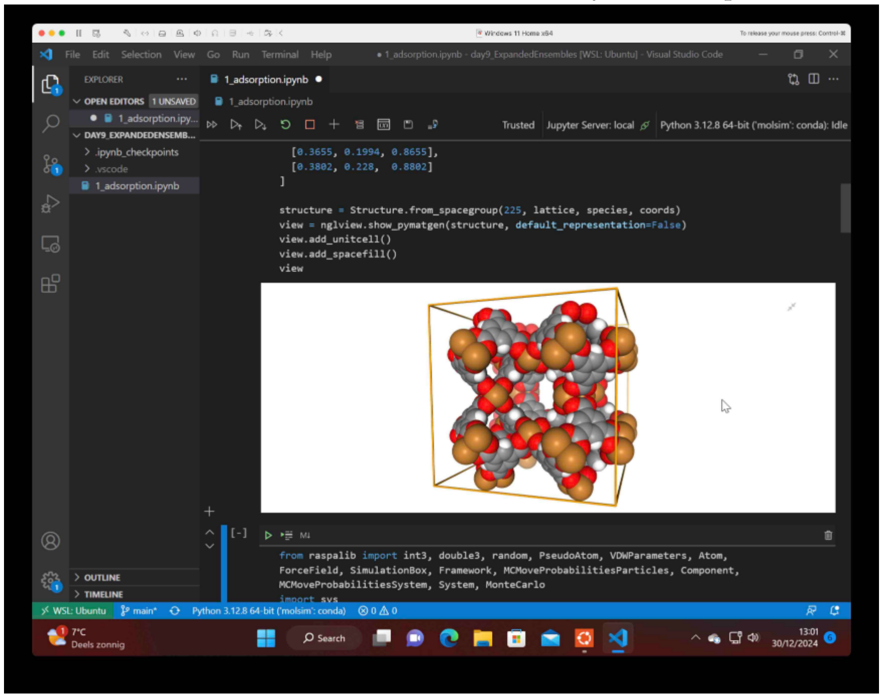
Current: ~/.micromamba/envs/molsim/bin/python

Enter interpreter path...
Enter path or find an existing interpreter
Python 3.12.3 64-bit
/usr/bin/python3
Python 3.12.3 64-bit
/bin/python3
Python 3.12.8 64-bit ('molsim': conda)
~/.micromamba/envs/molsim/bin/python

```
from pymatgen.core import Lattice, Structure
```

```
dubbelda@VM-MacPro:~/Molsim-source-solutions/day9_ExpandedEnsembles$
```

(a) Select the 'molsim' environment as the Python Interpreter.


Trusted | Jupyter Server: local | Python 3.12.8 64-bit ('molsim': conda): Idle

```
    [0.3655, 0.1994, 0.8655],
    [0.3802, 0.228,  0.8802]
  ]

structure = Structure.from_spacegroup(225, lattice, species, coords)
view = nglview.show_pymatgen(structure, default_representation=False)
view.add_unitcell()
view.add_spacefill()
view
```

```
from raspalib import int3, double3, random, PseudoAtom, VDWParameters, Atom,
ForceField, SimulationBox, Framework, MCMoveProbabilitiesParticles, Component,
MCMoveProbabilitiesSystem, System, MonteCarlo
import sys
```

(b) Example of NGLview to view atomic sytructures.

Figure 10.3: Running VC Code in Windows

```
explorer.exe ‘wslpath -w "$PWD"‘
```

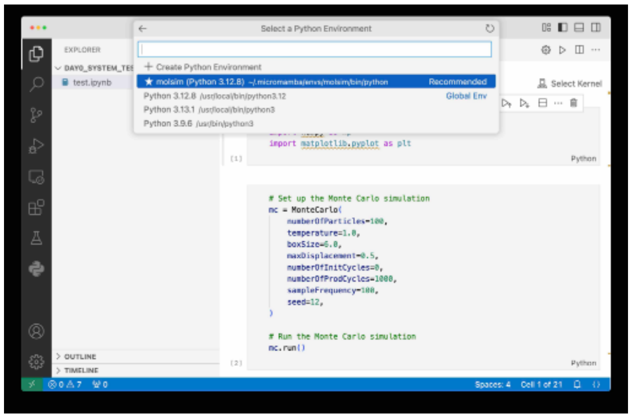
When you access files on your Windows file-system from within Bash, it will honor the NT file-system behaviors (e.g. case-insensitivity), permissions, etc. so you can easily access the same files using both Windows tools and Bash tools without having to copy files back and forth between file-systems.

10.5 Test notebook

In directory 'day0_system_test' you will find a notebook 'test.ipynb' to test the correct installation of the software. The following commands in the unix terminal

```
    micromamba activate molsim
    cd day0_system_test
    code .
```

should open Visual Studio Code. Click on `Select Kernel` and select the `molsim` kernel.



```
import matplotlib.pyplot as plt
```

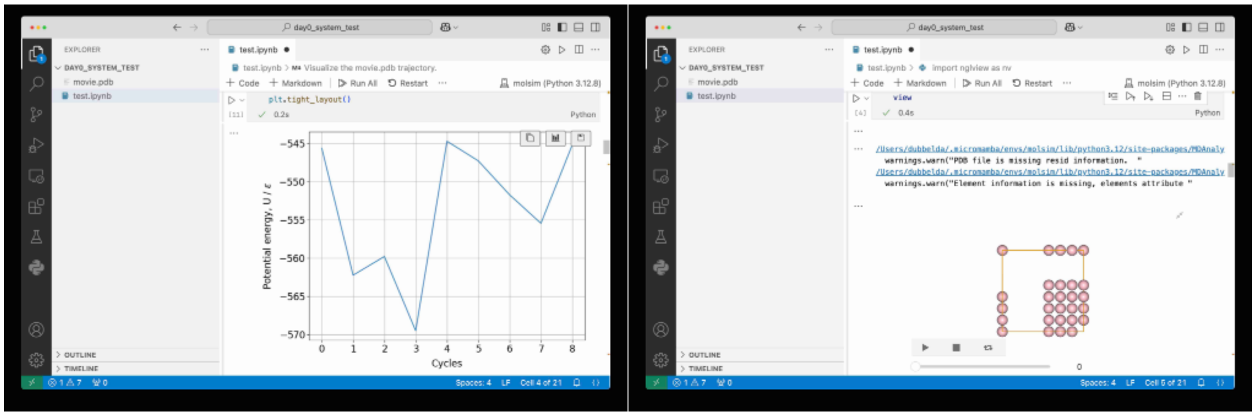
```
# Set up the Monte Carlo simulation
mc = MonteCarlo(
    numberOfParticles=100,
    temperature=1.0,
    boxSize=6.0,
    maxDisplacement=0.5,
    numberOfInitCycles=0,
    numberOfProdCycles=1000,
    sampleFrequency=100,
    seed=12,
)

# Run the Monte Carlo simulation
mc.run()
```

Select a Python Environment
Create Python Environment
★ molsim (Python 3.12.8) ~/.micromamba/envs/molsim/bin/python Recommended
Python 3.12.8 /usr/local/bin/python3.12 Global Env
Python 3.13.1 /usr/local/bin/python3
Python 3.9.6 /usr/bin/python3
Select Kernel

Figure 10.4: Testing `test.ipynb` notebook in Visual Studio Code.

Next, click on `Run All`. There should not be any errors. The output should look like:



```
plt.tight_layout()
```

```
view
```

```
/Users/dubbelda/.micromamba/envs/molsim/lib/python3.12/site-packages/MDAnaly
  warnings.warn("PDB file is missing resid information. "
/Users/dubbelda/.micromamba/envs/molsim/lib/python3.12/site-packages/MDAnaly
  warnings.warn("Element information is missing, elements attribute "
```

(a) Matplotlib (b) NGLView

Figure 10.5: Testing `matplotlib` and `NGLView` used in the `test.ipynb` notebook in Visual Studio Code.

Any errors on the `from molsim import MonteCarlo` line, means you forgot the install step

```
    cd Molsim-source-exercises
    pip install .
```

10.6 FAQ

10.6.1 macOS: `brew install micromamba` gives `'do not provide support for this old version'`

Solution: use 'brew install miniconda'.
Note: instead of 'micromamba', use the command 'conda'.

Does VS Code run on Apple silicon machines?

Yes, VS Code supports macOS Arm64 builds that can run on Macs with the Apple silicon chipsets. You can install the Universal build, which includes both Intel and Apple silicon builds, or one of the platform specific builds.

WSL Ubuntu stuck on "Installing, this may take a few minutes"

After a few minutes, the ubuntu app can be opened. However, WSL did not create a default user, and the terminal opens with user `root`.

```
adduser <username>
adduser <username> sudo
ubuntu.exe config --default-user <username>
```

and restart the `Ubuntu` app.

WSL 'code .' command returning error ".../bin/code not found"

- Open VS Code on Windows
- Open Extensions and then search on WSL

It should say the extension needs to be reloaded - go ahead and reload it Close VS code on Windows, and type `code .` in the linux terminal and this time it should launch.

Removing the `molsim` environment

You can remove the `molsim` conda environment with

```
    micromamba env remove --name molsim
```

Removing the `conda` environment

On the mac

```
    brew remove micromamba
```

"The kernel 'molsim (Python 3.12.8)' died"

Can be caused by running out of memory.

VSCode does not properly load env vars for environment created with micromamba executable

- In VSCode open your command palette — `Ctrl+Shift+P` for linux and windows, `Ctrl+Option+P` for mac by default
- Look for `Python: Select Interpreter` In `Select Interpreter` choose `Enter interpreter path...`
- Enter the path `~/.micromamba`.
- In the virtual environment folder choose `molsim (Python 3.12.8) ~/.micromamba/envs/molsim/bin/python`

Visual Studio Code: `Could not render content for 'application/vnd.jupyter.widget-view+json'`

Update the `Jupyter Notebook Renderer extension` to `Switch to Pre-Release Version`.

Visual Studio Code truncates the output cells

Click at the bottom of the output cell on `View as scrollable element`.