

# Python for Data Science and Materials Science

---

Shyue Ping Ong

Aiiso Yufeng Li Family Department of Chemical and Nano Engineering  
University of California, San Diego

<http://materialsvirtuallab.org>

# Overview

Preliminaries

Python

Scientific Python

Materials analysis

# Preliminaries

---

# Preliminaries

- In this lecture, we will provide an overview of the Python programming language.
- This will form the basis for your first lab on materials data wrangling in Python.
- Very quick overview of chapters 1-4 of the Python Data Science Handbook[1] + materials science libraries.
- Content is targeted at giving a complete beginner to Python and the scientific python stack enough background to get started.
- This lecture is extremely heavy on practical demos and examples. Please bring your laptop so that you can follow along.
- *Coding is best learned by attempting to solve a practical problem.* The goal of this lecture is not to comprehensively cover all the libraries (which would be impossible in a one-quarter course).

Python

---

# Introduction to Python



- General-purpose, high-level programming language
- Design emphasizes code readability
- Supports multiple programming paradigms (OOP, imperative, functional, procedural)
- Dynamically typed, automatic memory management and large standard library
- Available on almost all platforms

# Python vs other languages

- Java

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

- Python

```
print("Hello world")
```

# Python Version

- We will be using Python 3.10+.
- Options for this course:
  - Install python on your laptop - faster, not-reliant on internet connection, most modern version of Python possible.
  - [Google Colab](#) - Online Python notebooks. No installation required. But Python version and installed libraries may lag the latest versions (sometimes by years).



# Installing Python

- Regardless of whether your OS comes with Python, it is recommended that you install your own version of python using a package manager (e.g., the Anaconda distribution, uv, poetry, etc.).
- For the Anaconda distribution, you have two options:
  - Miniconda** Minimal installation of Python interpreter itself + conda command-line tool. conda is essentially a package manager.  
[Link](#)
  - Anaconda** Miniconda + several scientific packages such as scipy, numpy, etc. Several GB in size. [Link](#)
- Recommendation: Use miniconda to keep bloat to a minimum.

# Basics of Python

- Dynamically typed, i.e., no need for variable declaration.

```
x = 1 # integer
y = 2.0 # float
my_string = "Hello world" # string
my_list = [1, 2, 3, "a"] # list
my_dict = {"a": 1, "b": 2, "c": 3} # dict
```

- Other than some basic operations and functions, most other functions are in packages and modules and must be *imported* prior to use.

```
import math
```

```
z = math.sin(3.14159) # Gives ~0
```

- Please go through [official Python tutorial](#).

# Virtual environments

- Isolated environment for Python projects, i.e., each env has its own Python and packages. Both conda and python3 itself comes with the ability to create virtual environments.

```
conda create --name nanox81 python=3.10  
conda activate nanox81
```

- To start over, you can remove the environment:

```
conda remove --name nanox81 --all
```

- You can also create reproducible environments from a file.

```
conda env create -f nanox81_env.yml
```

- Example nanox81\_env.yml file given in the [NANOx81RGithub repo](#).

# Installing packages using conda

- We will get into the details of the specific packages later.

```
# Scientific python stack
```

```
conda install numpy scipy matplotlib seaborn jupyter
```

```
# Data Science tools
```

```
conda install pandas scikit-learn tensorflow
```

```
# Materials science library
```

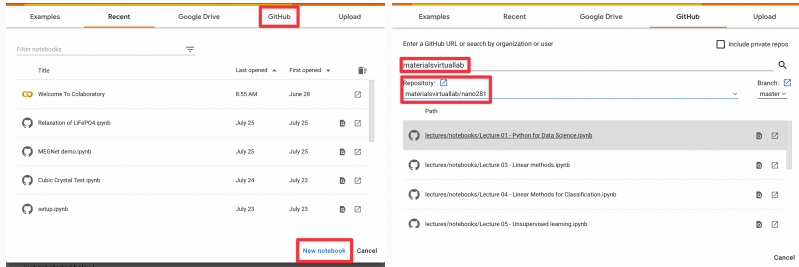
```
conda install --channel conda-forge pymatgen
```

- To update a package, use:

```
conda update numpy
```

# Google Colab Demo

- URL: <https://colab.research.google.com>
- Start a new blank notebook or download one from Github.



- Most of the data science packages we will be using are pre-installed.
- To install a missing package (e.g., pymatgen), run the following in the first cell:  

```
! pip install pymatgen
```

# Scientific Python

---

# Scientific Python Packages



**jupyter** Software for interactive computing

**Numpy** Fundamental package for numerical computation. Defines array and matrix types and basic operations on them.

**Scipy** Efficient numerical routines such as routines for numerical integration and optimization.

**pandas** High-performance, easy to use data structures.

**scikit-learn** Tools for predictive data analysis.

**TensorFlow** Library for developing and training ML models.

**PyTorch** Library for developing and training ML models.

**matplotlib** De facto plotting library

**seaborn** Statistical data visualization

## Brief note on coverage

- In the following slides, we will provide a brief background on all these packages, with a heavy focus on actual demos.
- It should be noted that it is not possible to comprehensively the extensive functionality of these packages in this course.
- Focus will be on modules and methods that pertain to the data science problem we are solving. You are expected to navigate the relevant documentation pages of each package (plus a generous amount of googling) to identify the appropriate module or method for what you need to do.
- numpy and scipy are the backbone for many of the other packages such as pandas and scikit-learn. You need to understand the basics of numpy arrays and how to perform various simple linear algebra operations. In most instances, the actual data science package (scikit-learn) will be the one you interact with most.



# Jupyter notebook

- We will primarily be using Jupyter for its notebook application, though it is capable of a lot more. (Google Colab is similar.)
- Jupyter Notebook is an open-source web application to create and share documents that contain code, equations, visualizations and narrative text, etc.
- All lecture examples will be done in Jupyter notebooks. You will also be submitting notebooks (ipynb files) as your lab reports.
- Document: [link](#)
- Running a jupyter notebook:  
`jupyter notebook`
- The notebook server will be running at `http://localhost:8888` (typically).

# Jupyter notebook, contd.

- Things you can do in a notebook:
  - Run code in a cell with 'shift+return' - helpful things include tab completion, magic functions (e.g., ?, ! and % are the key ones to know, especially ?).
  - Write text/narrative - set cell to markdown mode and write markdown.
  - Make plots.
- Very useful for quick code prototyping and data analysis.

# Numpy

- Implements the array and matrix objects.
- Underlying implementation is in C, and hence extremely efficient.
- Vectorization of operations, as opposed to loops, is key to efficiency.
- Recommend that you focus on the array object and ignore the matrix object.
- Note that standard operations such as `+`, `-`, `*` and `/` for `np.arrays` are element-wise. For matrix multiplications, use special numpy functions, e.g., `numpy.dot` or the new `@` operator available in Python 3.7 and above.
- For those of you who are familiar with Matlab, it helps to read the [Numpy for Matlab users doc](#).
- Documentation: [link](#)

# Numpy examples

```
import numpy as np

x = np.array([[1, 2, 3], [4, 7, 6], [9, 4, 2]])
y = np.array([1.5, 0.5, 3])
x * x
# array([[ 1,  4,  9],
#        [16, 49, 36],
#        [81, 16,  4]])
np.dot(x, x)
# array([[36, 28, 21],
#        [86, 81, 66],
#        [43, 54, 55]])
np.linalg.inv(x)
# array([[ 0.16949153, -0.13559322,  0.15254237],
#        [-0.77966102,  0.42372881, -0.10169492],
#        [ 0.79661017, -0.23728814,  0.01694915]])
```

# Scipy

- Efficient numerical routines for numerical integration, interpolation, optimization, linear algebra, and statistics, etc.
- `scipy.linalg` is a superset of `numpy.linalg`, and is always compiled with BLAS/LAPACK. So use `scipy.linalg`.
- Dependency for most other packages you will be using.

```
from scipy import stats # Statistics package
```

```
dist = stats.norm(0, 1) # Gaussian distribution
```

```
print(dist.cdf(1.96))
```

```
# Result: 0.9750021048517795
```

```
from scipy import constants # Physical constants
```

```
print(constants.e)
```

```
# Result: 1.6021766208e-19
```

- Documentation: [link](#)
- Scipy lectures (very useful resource): [link](#)

# Pandas

- Data analysis library
- Data structures designed for working with relational or labeled data
- Two main data types:
  - Series** 1D labeled homogeneously-typed array
  - DataFrame** 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column
- It is helpful to think of DataFrame as essentially a Python object representing the contents of a table, e.g. such as what you would have in Excel spreadsheet. This is similar to the dataframe in R.
- Please go through the [Pandas Guide](#).

# Pandas demo

- For this demo, we will utilize data that has been published by the MAterials Simulation Toolkit - Machine Learning (MAST-ML) on figshare.

```
import pandas as pd

url = "https://ndownloader.figshare.com/files/13007075"
# Reads in the data as a pandas DataFrame.
data = pd.read_csv(url)

# Extracting a column as a Series.
form_e = data["E_raw (eV)"]

# Summary statistics
data.describe()
```

# scikit-learn & TensorFlow & PyTorch

- Packages for machine learning.
- To some extent, scikit-learn, TensorFlow (TF) and PyTorch overlap in functionality.
- For the purposes of this course, we will use mainly scikit-learn for ML and will briefly touch on TF/PyTorch for neural networks.
- We will defer all specific code examples for scikit-learn until we go through specific types of ML models, e.g., linear regression, kernel regression, etc. Having the theory background of the ML methods will make it easier to understand the code usage and implementations.
- Instead, we will provide an overview of the high-level API design of scikit-learn.



# Scikit-learn Model API

- All ML models inherit from BaseEstimator and various mix-in classes with standardized method names.

```
model = Model(**params) # Init a model with parameters.  
model.get_params() # Get model parameters.  
model.set_params(**params) # Set model parameters  
model.fit(X, y) # Fit the model with a dataset  
model.score(X, y) # Provides the score for the dataset.  
model.predict(X) # Make a prediction using fitted model.
```

- Note that the params and score depends on the specifics of the ML model type.

# Matplotlib and Seaborn

- Matplotlib is the de facto library for generating all kinds of plots and many other plotting engines (including seaborn) are built on top of it
- seaborn is a extremely helpful package for statistical data visualization. Works very well with Pandas DataFrame.

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

```
# Plot two variables x and y with 'x' as markers and solid lines.
```

```
plt.plot(x, y, "x-")
```

```
# Use circles as markers and dashed lines instead.
```

```
plt.plot(x, y, "o--")
```

```
import seaborn as sns
```

```
# Plot the distribution of x.
```

```
sns.histplot(x)
```

```
# Scatter plot using columns from DataFrame
```

```
sns.scatterplot(x="Melting point", y="Modulus", data=materials)
```

- [Seaborn gallery](#) contains many visual examples with source code.

# Materials analysis

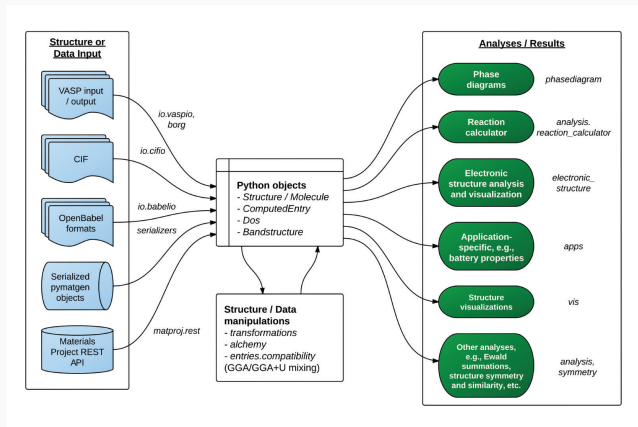
---

## Materials Analysis

# Python Materials Genomics (pymatgen)

- Core materials analysis powering the Materials Project.[2]
- Defines core extensible Python objects for materials data representation.
- Provides a robust and well-documented set of structure and thermodynamic analysis tools relevant to many applications.
- Establishes an open platform for researchers to collaboratively develop sophisticated analyses of materials data.
- High-level methods to access the Materials Project API.[3] (particularly useful for this course).

# Overview of capabilities



# Accessing the Materials API

- Accessed using the `pymatgen.ext.matproj.MPRester` class.
- Please review the <https://api.materialsproject.org>.

```
from pymatgen.ext.matproj import MPRester
```

```
# Change "<APIKEY>" to the API key obtained from MP.
```

```
with MPRester("<APIKEY>") as mpr:
```

```
    data = mpr.summary.search(
```

```
        formula=["Fe2O3"],
```

```
        fields=["formula_pretty", "formation_energy_per_atom", "band_gap"],
```



```
    )
```

```
print(data)
```


```
import pandas as pd
```

```
df = pd.DataFrame(data) # Convert to DataFrame
```

# Bibliography i

-  Jake VanderPlas.  
*Python Data Science Handbook: Essential Tools for Working with Data.*  
O'Reilly Media, Sebastopol, CA, 1 edition edition, December 2016.
-  Shyue Ping Ong, William Davidson Richards, Anubhav Jain, Geoffroy Hautier, Michael Kocher, Shreyas Cholia, Dan Gunter, Vincent L. Chevrier, Kristin A. Persson, and Gerbrand Ceder.  
**Python Materials Genomics (pymatgen): A robust, open-source python library for materials analysis.**  
*Computational Materials Science*, 68:314–319, February 2013.



 Shyue Ping Ong, Shreyas Cholia, Anubhav Jain, Miriam Brafman, Dan Gunter, Gerbrand Ceder, and Kristin a. Persson.

**The Materials Application Programming Interface (API): A simple, flexible and efficient API for materials data based on REpresentational State Transfer (REST) principles.**

*Computational Materials Science*, 97:209–215, February 2015.

The End