Pandas Lab, Part 2 — October 16, 2025

1. Libraries and Modules in Python

In Python, libraries and modules help you organize and reuse code:

- A module is any .py file that contains Python code. You can import its functions, classes, or variables into other scripts.
- A library is a collection of modules packaged together. Libraries can be standard (like math, os) or external (like NumPy, Pandas).

1.1 Example: Creating a Simple Local Library

You can create a local library (or package) by creating a folder containing Python files and an **init**.py file. Example:

```
my_lib/
├── __init__.py
└── utils.py
```

- The **init**.py file makes the folder a *package*.
- It can be empty — its mere presence tells Python that this is a module.

1.2 Different Ways to Import and Use Local Modules

Given a structure like:

```
project/
├── main.py
└── my_lib/
    ├── __init__.py
    └── utils.py
```

If utils.py contains a function:

```
# utils.py
def greet(name: str) -> str:
    return f"Hello, {name}!"
```

You can load and use it in different ways:

```
# Option 1
from my_lib.utils import greet
print(greet("Alice"))

# Option 2
import my_lib.utils as u
print(u.greet("Bob"))

# Option 3
import my_lib.utils
print(my_lib.utils.greet("Carol"))
```

2. What is Pandas?

Pandas is a powerful Python library for data manipulation and analysis. It provides two main data structures:

- Series: one-dimensional labeled arrays
- DataFrame: two-dimensional labeled tables (like a spreadsheet or SQL table)

It is widely used in data science, machine learning, and bioinformatics.

2.1 Useful Links

- Pandas Documentation
- 10 Minutes to Pandas
- Pandas Tutorials on W3Schools

2.2 Basic Examples

```
import pandas as pd

# Create a DataFrame
data = {'name': ['Alice', 'Bob'], 'age': [25, 30]}
df = pd.DataFrame(data)
print(df)

# Read a CSV file
df = pd.read_csv('example.csv')

# Describe summary statistics
print(df.describe())
```

```
# Filter rows
young = df[df['age'] < 28]
print(young)

# Save to CSV
df.to_csv('out.csv', index=False)
```

3. Today's Task: Mutation Rate Calculator

You will use your previous NumPy project (Needleman–Wunsch alignment) to create a new script using Pandas. Your goal is:

> Given a FASTA alignment file, calculate the mutation rate of each sequence in relation to a reference sequence (selected by the user).

3.1 Project Checklist

Your repository must contain:

- README.md — explain what the project does and how to use it
- LICENSE — include a license (MIT, GPL, etc.)
- main.py — the main script that runs the analysis
- A local module folder (e.g., mut_calc/) with:
    - **init**.py
    - Supporting Python scripts (e.g., for parsing FASTA, computing mutation rates)

3.2 Code Requirements Checklist

- Use docstrings and comments to document your functions.
- Use type hints throughout your code.
- Use at least one class in main.py.
- Use argparse to handle user input from the command line.
- Use sys.stdout and sys.stderr for appropriate output and errors.

3.3 Testing and Examples Checklist

- Provide an example FASTA file.
- Include a Bash script (e.g., run_tests.sh) that runs your project on the test file.

3.4 Demonstration Requirement

You must demonstrate to the TA before 6:45 pm today that your code:

- Runs successfully in an Anaconda environment.
- Produces a mutation rate output.
- Handles errors and missing input gracefully.

4. Why These Instructions Are Open-Ended

You may have noticed that this lab gives fewer step-by-step instructions. That's intentional.

In the real world, you'll often face vague requirements, unexpected errors, and the need to make technical decisions on your own. By navigating this ambiguity:

- You practice problem-solving and exploration.
- You improve your ability to design modular code.
- You build confidence working with real data and tools.

Take this as an opportunity to be creative, take ownership, and prepare for professional coding scenarios.

> Show your work to the TA before 6:45 pm. Good luck!