Programming Basics for Bio(techno)logical Data

This series is not meant to replace a formal programming course, and thus it will not cover programming theory or fundamentals. Rather, it is aims to introduce useful tools and techniques that can be applied for the preprocessing, analysis, and visualization of data, with a focus on bio(techno)logical data. Example workflows and relevant links are provided for each concept.

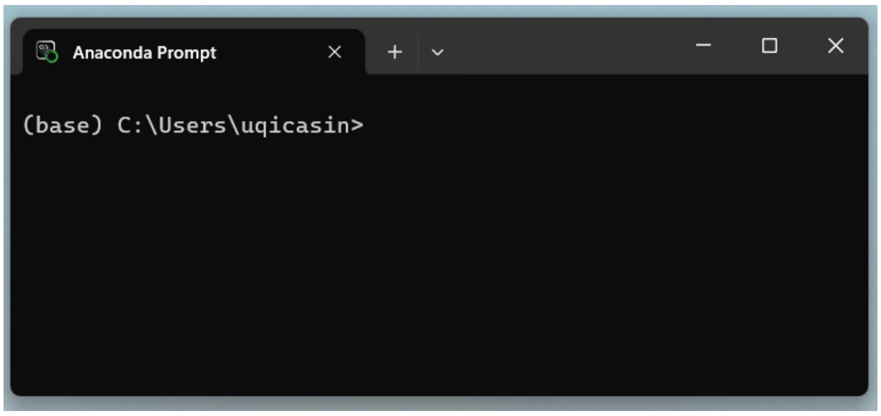
In this tutorial: You will learn how to: **1)** some basic data types in Python **2)** how to read in csv, txt, and Excel® files, and **3)** write out files.

Note: All code will be bolded and in the font **`Courier New`**.

1. Installation of additional packages (mini-review)

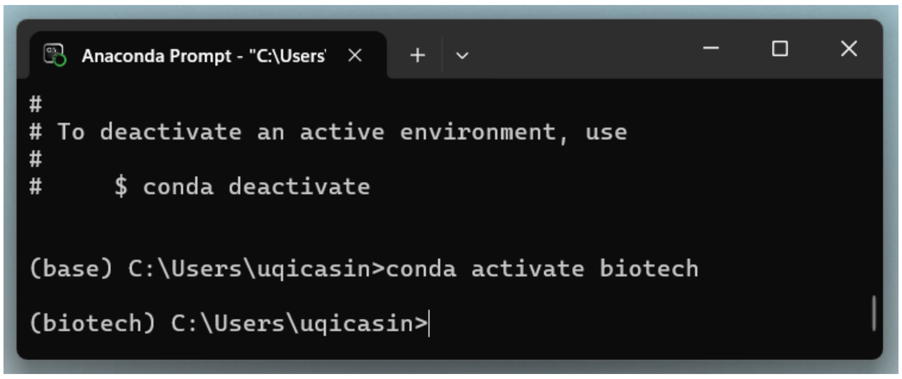
In the last session, we learned how to create our conda environment ("biotech") and how to install packages. In this session, we will install a couple more packages that allow us to interact with Excel® files.

1. Open the **Anaconda Prompt**. The icon looks like this:
   a. You should see the directory you're in (**C:\Users\uqicasin**) and your Python base or starting environment (**base**) to the left.



```
(base) C:\Users\uqicasin>
```

2. Activate your new environment: paste the following command and hit "**Enter**".
   a. **`conda activate biotech`**
   b. You should see that your environment has changed from "base" to "biotech".



```
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) C:\Users\uqicasin>conda activate biotech

(biotech) C:\Users\uqicasin>
```

3. Install packages of interest (openpyxl): paste the following commands and hit “**Enter**”.
    a. `conda install openpyxl`
    b. You will be asked the following question: “Proceed ([y]/n)?
        i. The default response is “y” or yes.
        ii. Type “`y`” and hit “**Enter**”
    c. Note: additional packages will be automatically installed for our packages of interest to function.
4. Check that your packages were installed: paste the following command and hit “Enter”.
    a. `conda list`

2. Python Data Types

In this tutorial, you will be walked through different data types that are typically used in Python. You will be introduced to the "print" function that can be used to display designated information. You will also be introduced to the "type" function that will help you determine what data type a variable/object is. Examples are given in the accompanying 002_DataTyles.ipynb. A summary of functions learned are provided below.

Important notes on variables:

1) You use variables to store objects or information
2) Python is case-sensitive. "bioreactor" is different from "Bioreactor"
3) You can only name variables with alpha-numeric characters and underscores. But they cannot **start** with a number.
4) There are some keywords in Python, that have fixed functions, you cannot use those either for variable names.
    a. https://www.w3schools.com/python/python_ref_keywords.asp

Important other notes:

1. Python "indexes" (starts counting) at 0 (not at 1)
2. In a Jupyter Notebook file, you have the option of putting different cell types, including “markdown” code. These are written in markdown not in Python and are great for writing text in between cells of code.
3. If you want a line not to be read (as code), you write a pound symbol (#) in front. This will "comment" out the entire line. It can also be used midline.
4. You can comment out a line(s) using Ctrl + / (Windows)

Table 1. Summary of Data Types in Python

Data Type	Description	Example	When to Use
Text (str)	A sequence of characters (letters, numbers, symbols)	"Bioreactor 1", "101", "abc123!"	When storing names, messages, or any readable text
Numeric (int, float)	Numbers. int = whole numbers, float = decimals	10, -3, 4.5, 2.0	For math, counting, measurements, prices, scores, etc.
Boolean (bool)	True/False values	True, False	For flags, conditions, yes/no decisions
List (list)	An ordered, changeable collection (can mix types)	[1, "apple", True]	When you need a flexible group of items
Dictionary (dict)	Key-value pairs - like a labelled list of data	{"color1": "red", "age": 25}	When you want to label data or store structured records
NumPy Array	Grid-like data (usually numeric), efficient for math and large datasets	array([[1 2 3], [4 5 6]])	For fast scientific computing, matrices, or vectorized math
Pandas DataFrame	Table of rows and columns (like a spreadsheet or database)	df = pd.DataFrame(...)	For data analysis, CSVs, time series, real-world structured data