Week 1: Preliminaries

Chapter 1

Programming in Scientific Applications

- Embedded devices: check mileage, monitor emissions & speed, tracks position with GPS etc.
- Imaging: Medical scanning technology, flight simulators, computer gaming engines, CAD/CAM
- Information technology: web search engines, electronic commerce, algorithmic trading
- Routing & tracking: algorithms to synchronize information from sensors, transponders, satellites and ground control to manage aircraft position, speed, altitude, and trajectory
- Biology and Medicine: Genome analysis, protein folding, vaccine development
- Public Policy: contact tracing in pandemics, simulating disease spread and identifying smart mitigation strategies to prevent it etc.

Chapter 2

Agenda

- Goals
- Syllabus walk-through
- Software & Tools
- Python basics: Variables, Expressions, Data Types, Functions, Modules
- Session Theme: Variations on computing the value of π

Chapter 3

Course Goals

- Develop **computational thinking** skills: the ability to harness "thought processes involved in expressing solutions as computational steps or algorithms that can be carried out by a computer"
- Learn to apply **logic** and **precision** to problem solving
- Become fluent in the basic **syntax** and **semantics** of Python
- Understand best practices and develop skills for documenting, testing, debugging and refining Python code (baby steps to **software engineering**)

  Build a foundation for progress from a novice programmer to an expert

Chapter 4

Command-Line Shell

If you are not familiar with a Unix/Linux **command-line shell**, please take the time to learn about it! It comes pre-installed on Macs and Linux boxes as a `Terminal` application, and on Windows you can either install WSL 2 (Windows Subsystem for Linux) or Git Bash.

> **!** Important
>
> All data scientists should know how to skillfully use a command line shell to navigate through file systems, run commands and programs, and access cloud services.

Chapter 5

Software Installation

You can install Anaconda which comes bundled with Python and all the Python data science-related packages that will be used in this course. It also comes with its own package manager.

> **Tip**
>
> Consider installing Miniconda: this is a minimal installer that comes with a package management tool called `conda`.

Alternatively, if you want more control over what gets installed, you can install the needed software **individually**:

- Python: Version 3.11: check that `pip` or `pip3`, the Python package management utility has been installed. On WSL2, you will need to use the default package manager called `apt` (for Ubuntu Linux, the default distribution installed with WSL) - see the installation directions here.
- Once `pip3` installed, you can install **any** Python package from the PyPi repository. For now, just make sure that `jupyterlab`, `matplotlib`, `numpy`, `scipy` and `pandas` are installed using `pip`.

> **Warning**
>
> You should **either** use `conda` or use `pip`: **do not use both** as this can be a major source of confusion when it comes to locating exactly which version of which package is being used by your Python code!

Chapter 6

Editing Programs

There are basically two possibilities for **writing** Python code and **Markdown** annotations (for text explanations of the code):

- either a standalone or an integrated editor (called an **IDE**): e.g., **Spyder**, **VS Code**, **Vim** etc.
- a browser-based editing environment called a Jupyter notebook that is either local to your laptop (via Jupyter lab) or is hosted through Google's Colab service.

Chapter 7

Textbook code

The code demonstrated in the textbook is avaialble in a Github cloud repository.

This code can be downloaded. Please experiment with it while reading the corresponding chapters from the textbook.

Chapter 8

Python from 30,000 feet

- Invented by Guido Van Rossum and first released in 1991 (named after the popular **Monty Python's Flying Circus**)
- Easily the most popular **high-level** language with native support for different programming **paradigms** and excellent built-in libraries
- It is an **object-oriented** programming language: we understand programs as specifications of **interactions** among **computational objects** like numbers, strings, functions, files containing data and code, fixed-length and variable-length sequences, and so forth.
- It is an **interpreted** language: code is executed in interactive mode in a **REPL** (read-eval-print-loop) or is run as part of an **application**. This facilitates the rapid prototyping of code.
- It comes with **batteries included**, viz. a vast ecosystem of built-in packages.
- There are scores of really useful, third-party Python packages for data science!
- The Zen of Python: Clean, expressive and readable programs!
- It **encourages** computational thinking with intuitive and uniform *protocols* for iteration, handling data streams, and safely managing resources.

Chapter 9

Python Programs: Basic Ingredients

- Data (objects, types, *names*)
- Statements:
  - Assignments
  - Selection
  - Loops
- Functional abstraction

Chapter 10

Data Types

- Sets of **values** with associated operations on the values
- Some primitive types in Python:
  - Integer (`int`)
  - Float (`float`)
  - String (`str`)
  - Boolean (`bool`)
- Container type: `list`, `tuple`, `set`
- Mapping type: Dictionary (`dict`)

Chapter 11

Objects and (Named) Variables

- Values (corresponding to data types) are called **objects**
- We **reference** objects by giving them **names**; the names are called *variables* or *variable names*.

Chapter 12

Statements

A **program** in Python is a sequence of statements that are **evaluated step-by-step**

Typical statement types:

- Directive
- Assignment
- Selection
- Loop
- Function definition …

Chapter 13

Assignment

Consult the Python Tutor website to visualize the **binding** of values to names in a Python code snippet!

- Assignments bind names to values
- Binding is **dynamic**: the type of a value always remains unchanged, but the same name may refer to different values (and hence different types) at various times during the program execution.

Chapter 14

Basic Selection

Statement executes the selected block of code only if the boolean condition is `True`

```
if <cond>:
    <selected code block>
```

Branching: statement executes *either* the selected block (exclusively) *or* the alternative block depending on whether the condition is `True` or `False`.

```
if <cond>:
    <selected code block>
else:
    <alternative code block>
```

Chapter 15

Basic for Loop

Python has a **repetition** construct for **definite loops** that repeat a certain *number* of times:

```
for <var> in <iterable collection>:
    <code block>
```

The block of code is repeatedly executed, but prior to each iteration, the variable name (in the `for` statement) is assigned the **next** value in sequence from the iterable collection.

Iteration stops when the collections runs out of values!

Chapter 16

Basic `while` loop

Python also has a **repetition** construct for **indefinite loops**:

```
while <cond>:
    <code block>
```

Prior to each iteration, the boolean condition is evaluated. If it is `True`, only then is the code block executed. Otherwise, iteration stops.

Chapter 17

Functions

Little pieces of code that hold a program together!

- Abstraction that **encapsulates** a piece of computation whose outcome (**return value**) depends on the supplied inputs (**parameters** or **arguments**)
- Separation of code into functional units allows us to **compose** functions in a *modular* fashion (and helps with a host of other desirable software development activities).

Chapter 18

Batteries Included

Python comes pre-loaded with **library modules** that already define useful data-types (**classes**) and **functions**. For example:

- `math`: mathematical and trigonometric functions
- `random`: pseudo-random numbers and distributions for simulations and randomized computation

Chapter 19

Theme: How to approximate π

We will use the `random` and `math` builtin modules!

- **Archimedean** approximation using perimeters of regular polygons
- **Madhava-Gregory-Leibnitz** approximation:

  $tan^{-1}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - ...$
- **Nilakantha** approximation:

  $\frac{\pi}{4} = \frac{3}{4} + \left(\frac{1}{2 \cdot 3 \cdot 4} - \frac{1}{4 \cdot 5 \cdot 6} + \frac{1}{6 \cdot 7 \cdot 8} - ...\right)$
- **Viete** approximation:

  $\frac{2}{\pi} = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{2}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} ...$
- **Brouncker** approximation:

  $\frac{\pi}{4} = \frac{1}{1+\frac{1^2}{2+\frac{3^2}{2+\frac{5^2}{2+...}}}}$
- **Wallis** approximation:

  $\frac{\pi}{2} = \left(\frac{2}{1} \cdot \frac{2}{3}\right) \cdot \left(\frac{4}{3} \cdot \frac{4}{5}\right) \cdot \left(\frac{6}{5} \cdot \frac{6}{7}\right) ...$
- **Monte Carlo** simulation using random darts
- **Monte Carlo** simulation using needles crossing horizontal lines