# Introduction

This lesson introduces you to the world of scientific computation.

WE'LL COVER THE FOLLOWING



For many decades, Fortran has been the language of choice for scientific computing because of its speed. In the 1980s, when a programmer's time was becoming more valuable than computation time, there was a need for languages that were easier to learn and use. For the purpose of research, code-compile-execute workflow gave way to the interact-explore-visualize workflow. In this context, MATLAB, IDL, Mathematica, and Maple were born.

Modern scientific computing is not just about numerical computing. It needs to be versatile: deal with large datasets, offer richer data structures than just numerical arrays, make network calls, interface with databases, interwork with web apps, handle data in various formats, enable team collaboration, enable easy documentation, etc.

## MATLAB vs. Python #

MATLAB is proprietary, expensive, and hard to extend. Python is open, community-driven, portable, powerful, and extensible. Python is also better with strings, namespaces, classes, and GUIs. While MATLAB(along with Simulink) has vast libraries, Python is catching up, since many scientific projects are adopting it. MATLAB is said to be poor at scalability, complex data structures, memory handling, system tasks, and database programming.

**MATLAB** $vs.$ **Python**

However, there are also some criticisms of Python. Its syntax is not consistent since different packages are written by different people with different needs. For example, there are two ordinary differential equation (ODE) solvers in SciPy that use incompatible syntaxes. Duplicated functionality across packages may also result in confusion. MATLAB does better with data regression, boundary value problems, and partial differential equations (PDE).

In 2014, Konrad Hinsen commented that Python may not be suitable for small-scale projects where code is written once and rarely maintained thereafter. This becomes a problem when Python's scientific libraries are upgraded by deprecating older classes, functions, and methods.

But yes, Python is suitable, at least in certain scenarios.

## Why Python for scientific computing? #

- Python is not just suited for manipulating numbers. It offers a ***computational ecosystem*** that can fulfill the needs of a modern scientist.
- Python does well with **system integration**. Duck typing is one of the aspects that aid scientific work with things like data types, memory view, PyCapsule, and NumPy's array just to name a few.

- Python is **easy to learn and use**. It offers a natural syntax. This enables researchers to express and explore their ideas more directly rather than fight with low-level language syntax.
- With Python, performance **bottlenecks** can be optimized at a low-level without sacrificing high-level usability. A researcher needs to explore and visualize ideas in an incremental manner. Python allows this via IPython/Jupyter notebooks and matplotlib. A variety of Python tools can work together and share data within the same runtime environment without having to exchange data only via the filesystem.

---

In the next lesson, we will look at who would most benefit from this course and what to expect from it.