

Python Tutorial – Part I: Introduction

Mark A. Austin

University of Maryland

austin@umd.edu

ENCE 688P, Spring Semester 2022

January 18, 2023

Overview

Part 1

1 What is Python?
• Origins, Features, Framework for Scientific Computing

2 Program Development with Python
• Working with the Terminal
• Integrated Development Environments

3 Data Types

4 First Program (Evaluate and Plot Sigmoid Function)

5 Builtin Collections (Lists, Dictionaries, and Sets)

6 Numerical Python (NumPy)

7 Data and Dataset Transformation (Pandas)

What is Python?

The Origins of Python

The Python programming language was initially written by Guido van Rossum in the late 1980s and first released in the early '90s. Its design borrows features from C, C++, Smalltalk, etc.

The name Python comes from Monty Python's Flying Circus.



Version 0.9 was released in February 1991. Fast forward to 2022, and we are up to Version 3.11.

What is Python?

Features:

- Designed for quick-and-dirty scripts, reusable modules, very large systems.
- Object-oriented. Very well-designed. Well documented.
- Large library of standard modules and third-party modules.
- Works on Unix, Mac OS X and Windows.
- Python is both a compiled and interpreted language. Python source code is **compiled** into a **bytecode format**.
- Integration with external C and Java code (Jython).

What is Python?

Strengths of Python:

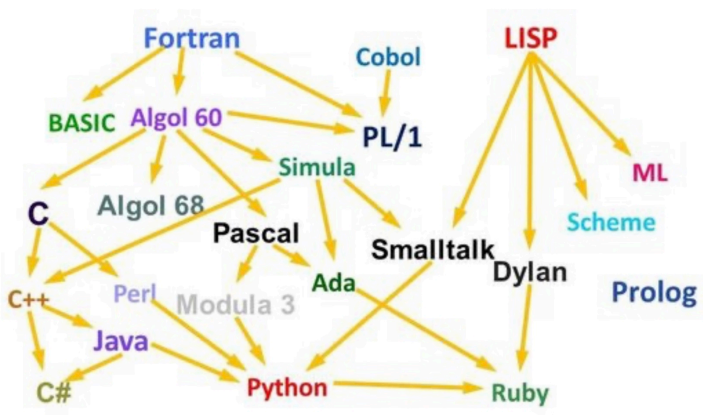
- Open source. Compared to C and Java, it's easy to learn.
- Provides an approximate **superset of MATLAB** functionality.
- Modern language with good support for object-oriented program development.

Third-Party Modules:

- NumPy is a language extension that **defines** the **numerical array** and **matrix type** and basic operations on them.
- SciPy uses numpy to do **advanced math**, signal processing, optimization, statistics, etc.
- Matplotlib provides easy-to-use **plotting Matlab-style**.

What is Python?

Graph of Feature Dependencies Among Computer Languages



Python Language: Borrows from C++, Java, Smalltalk, ..

Framework for Scientific Computing

Environments

— terminal window / console, Jupyter Notebook.

Python Language

— Python 2, Python 3

Python Packages

— numpy, pandas, matplotlib

System Libraries

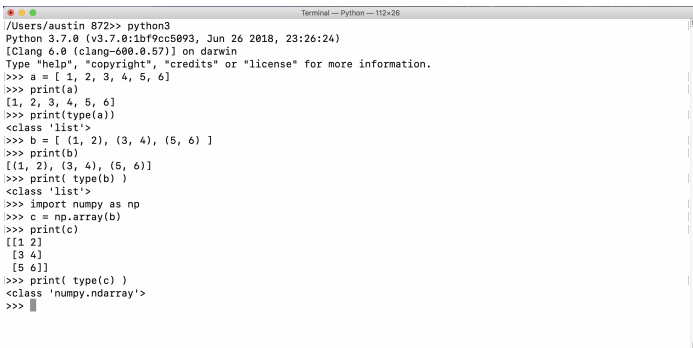
— BLAS, LAPACK (legacy numerical analysis).

Program Development with Python

First Steps: Working with the Terminal

Terminal Window (Console)

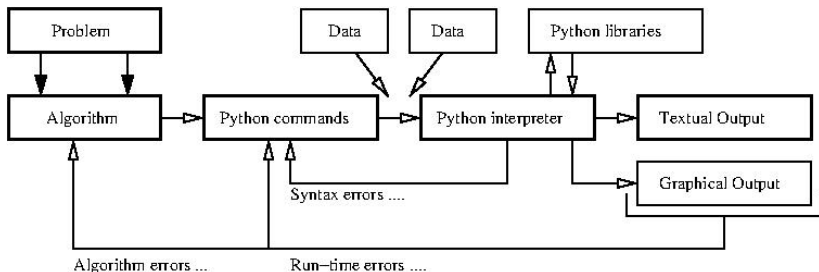
The **standard approach** runs a program directly through the **Python interpreter**.

A terminal window titled "Terminal — Python — 112x26" showing the execution of a Python script. The output shows the creation of a list, a tuple, and a NumPy array, along with their respective types.

```
~/Users/austin 872>> python3
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = [ 1, 2, 3, 4, 5, 6 ]
>>> print(a)
[1, 2, 3, 4, 5, 6]
>>> print(type(a))
<class 'list'>
>>> b = [ (1, 2), (3, 4), (5, 6) ]
>>> print(b)
[(1, 2), (3, 4), (5, 6)]
>>> print( type(b) )
<class 'list'>
>>> import numpy as np
>>> c = np.array(b)
>>> print(c)
[[1 2]
 [3 4]
 [5 6]]
>>> print( type(c) )
<class 'numpy.ndarray'>
>>> █
```

First Steps: Working with the Terminal

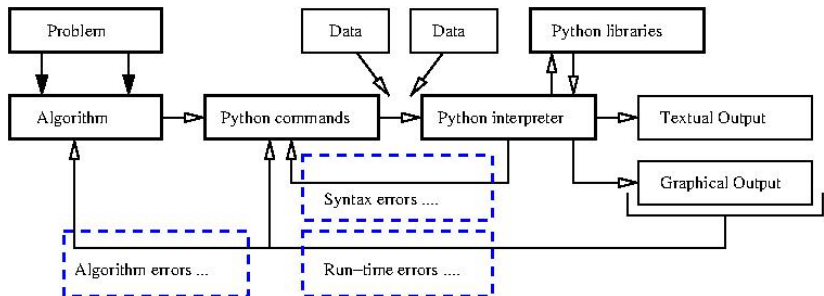
Program Development in the Terminal Window:



Step-by-Step Procedure:

- 1 Write, compile, fix, run, fix, run, validate → success!

First Steps: Fixing Mistakes



- 1 **Syntax Errors**: Check your typing ...
- 2 **Runtime Errors**: Program runs, but you have divide by zero and/or NaNs, etc.
- 3 **Algorithm Errors**: Does your program solve the right problem?

First Steps: Program Evaluation

Program Evaluation

- Robustness (does it work?)
- Accuracy and Efficiency (speed).
- Ease of Implementation (cost).

Things to Learn:

- How are numbers stored inside the computer?
- How do variables work?
- How do vectors and matrices work?
- How do list, dictionaries and sets work?
- What's in the Python Programming Language?
- How to apply Python to solution of numerical problems?
- Where can I go for help?

Integrated Development Environments

(Simplifying Program Development)

Integrated Development Environments

Integrated Development Environments

An **Integrated Development Environment (IDE)** is a **software application** that provides **comprehensive support** to computer programmers for **software development**.

State-of-the-art IDEs provide tools for:

- Syntax highlighting, editing source code, automation of program build, and code debugger.
- Program compilation (interpretation) and execution (run).

Two IDE's for Python:

- Visual Studio Code (for program development).
- Jupyter Notebook (web-based authoring of python documents).

Visual Studio Code

Visual Studio Code (vscode)

Visual Studio Code (vscode) is a source code editor for Windows, Linux and macOS. **Features** include **support** for **debugging**, **syntax highlighting**, **intelligent code completion** and code refactoring.

Standard Use Cases:

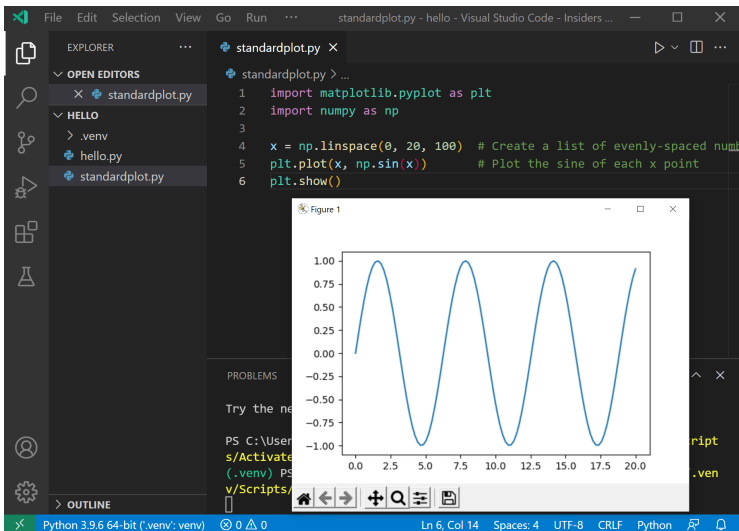
- Edit, debug, run, debug, run, test.
- Develop desktop apps.
- Numerical and scientific computing.

Advanced Use Cases:

- Deploy code to the cloud (Github).

Visual Studio Code

Graphical Interface



The screenshot displays the Visual Studio Code interface. The Explorer sidebar on the left shows a project structure with folders like ".venv" and "HELLO", and files "hello.py" and "standardplot.py". The main editor window shows the code for "standardplot.py":

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.linspace(0, 20, 100) # Create a list of evenly-spaced numbers
5 plt.plot(x, np.sin(x))     # Plot the sine of each x point
6 plt.show()
```

Below the code editor, a window titled "Figure 1" displays a plot of the sine function. The x-axis ranges from 0.0 to 20.0, and the y-axis ranges from -1.00 to 1.00. The plot shows a blue sine wave oscillating between -1 and 1.

The bottom status bar indicates the Python environment is "Python 3.9.6 64-bit (.venv: venv)" and the cursor is at "Ln 6, Col 14".

Jupyter Notebook

Jupyter Notebook (Web-based Application)

Web-based authoring of documents that combine live code with narrative text, equations and visualization.

To install Jupyter Notebook:

```
prompt >> pip3 install jupyter
```

To run Jupyter Notebook:

```
prompt >> jupyter notebook
```

Jupyter Notebook

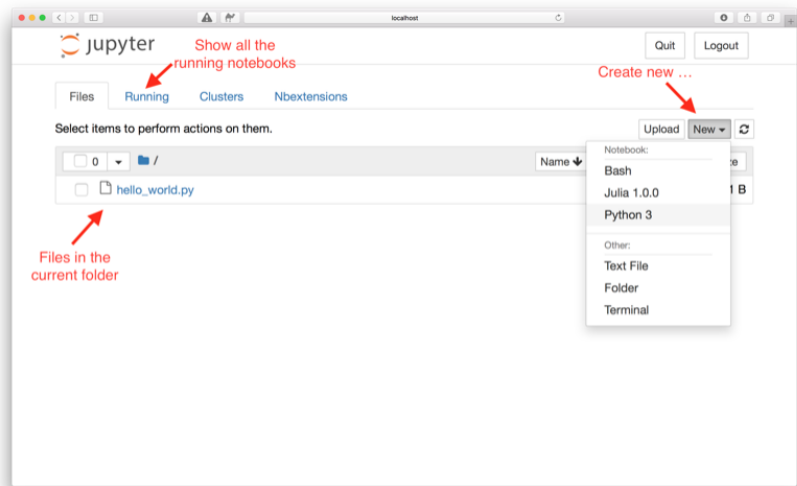
Use Cases:

- Data cleaning and transformation.
- Numerical simulation.
- Statistical modeling.
- Data visualization.
- Machine learning.

Jupyter Notebook File Format:

- File format is JSON-based with extension `.ipynb` (named after projects predecessor IPython).
- Supports documents containing text, source code, rich media data and metadata.

Jupyter Notebook User Interface



Jupyter Notebook User Interface

The screenshot displays the Jupyter Notebook interface in a web browser. The browser address bar shows the URL `localhost:8891/notebooks/hello_world.ipynb`. The notebook title is `hello_world`, and it indicates `Last Checkpoint: a minute ago (unsaved changes)`. The interface includes a **Header** with the notebook name and a **Menu** with options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A **Toolbar** is located below the menu, containing icons for running, saving, and other actions. The main content area shows a **Code cell** with the following code:

```
In [1]: 1 print('Hello World')
```

The code cell has been executed, resulting in the output `Hello World`. Below the code cell is a **Raw Markdown cell** containing the following text:

```
1 # This is a markdown blue (header level 1)
2
3 ## Header level 2
4
5 You can use bold text
6
7 You can use bullets list:
8
9 * bullet 1
10 * bullet 2
```

When this raw markdown cell is double-clicked, it is rendered into a **Rendered Markdown cell** with the following visual output:

This is a markdown cell (header level 1)

Header level 2

You can use **bold** text

You can use bullets list:

- bullet 1
- bullet 2

Jupyter Notebook Cells and Code Execution

Jupyter Notebook Cells:

- **Code Cells:** Allows for **development** and **editing** of **new code**, with **syntax highlighting** and tab completion.
- **Markdown Cells:** Document the computational process with the Markdown language (a simple way to perform text markup). Can also include mathematics with LaTeX notation.
- **Raw Cells:** Provide a place in which you can write output directly.

Code Execution:

- When a code cell is executed, the code is sent to the kernel associated with the code.
- Results are returned to the computation and then displayed.

Jupyter Notebook and Machine Learning

Jupyter Notebook (Machine Learning with TensorFlow)

untitled folder ▾ Google Yahoo Google Maps Facebook Apple iCloud Amazon YouTube Wolfram|Alpha Twitter

Cases — Jupyter Documentatio... Running the Notebook — Jupyter Documentation 4... ace · Editorial board | Hindawi Downloads/python-oreilly-hands-on-1-k

jupyter 01_the_machine_learning_landscape (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Run in Google Colab

Chapter 1 – The Machine Learning landscape

This is the code used to generate some of the figures in chapter 1.

[Run in Google Colab](#)

Code example 1-1

Although Python 2.x may work, it is deprecated so we strongly recommend you use Python 3 instead.

```
In [1]: # Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)
```

```
In [2]: # Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"
```

This function just merges the OECD's life satisfaction data and the IMF's GDP per capita data. It's a bit too long and boring and it's not specific to Machine Learning, which is why I left it out of the book.

O'REILLY™

**Hands-on
Machine Learning
with Scikit-Learn,
Keras & TensorFlow**

Concepts, Tools, and Techniques
to Build Intelligent Systems

Aurélien Géron

Jupyter Notebook and Machine Learning

Jupyter Notebook (Machine Learning with TensorFlow)

```
In [7]: def heaviside(z):
        return (z >= 0).astype(z.dtype)

        def mlp_xor(x1, x2, activation=heaviside):
            return activation(-activation(x1 + x2 - 1.5) + activation(x1 + x2 - 0.5) - 0.5)
```

```
In [8]: x1s = np.linspace(-0.2, 1.2, 100)
        x2s = np.linspace(-0.2, 1.2, 100)
        x1, x2 = np.meshgrid(x1s, x2s)

        z1 = mlp_xor(x1, x2, activation=heaviside)
        z2 = mlp_xor(x1, x2, activation=sigmoid)

        plt.figure(figsize=(10,4))

        plt.subplot(121)
        plt.contourf(x1, x2, z1)
        plt.plot([0, 1], [0, 1], "gs", markersize=20)
        plt.plot([0, 1], [1, 0], "ym", markersize=20)
        plt.title("Activation function: heaviside", fontsize=14)
        plt.grid(True)

        plt.subplot(122)
        plt.contourf(x1, x2, z2)
        plt.plot([0, 1], [0, 1], "gs", markersize=20)
        plt.plot([0, 1], [1, 0], "ym", markersize=20)
        plt.title("Activation function: sigmoid", fontsize=14)
        plt.grid(True)
```

