Python Tutorial – Part I: Introduction

Mark A. Austin

University of Maryland

austin@umd.edu
ENCE 201, Fall Semester 2025

September 2, 2025

Overview

1. What is Python?
   - Origins, Features, Framework for Scientific Computing
2. Program Development with Python
   - Working with the Terminal
   - Integrated Development Environments
3. Elements of Python Programming
   - Data Types, Variables, Arithmetic Expressions, Strings, Program Control, and Functions
4. First Program (Evaluate and Plot Sigmoid Function)
5. Builtin Collections (Lists, Dictionaries, and Sets)
6. Numerical Python (NumPy)

Introduction

What is Python?

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 2025 and we are up to Version 3.13.

What is Python?

Features: Advertising ...

- 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: (**easy to get started**)

- Provides an approximate superset of MATLAB functionality.
- Modern language with good support for object-oriented program development.
- But, Python doesn't force users to think in term of objects from the very beginning ...
- Open source. Licenses are free.

Weaknesses of Python: (**throw away code**)

- Behind the scenes, everything is an object. The language design is not as clean (logical) as Java.
- Python provides users with considerable freedom to mix-and-match data types. Code might not scale well, and could become very difficult to debug/maintain.
- Language versions are not backwards compatible. Ugh !!!!

What is Python?

Third-Party Tools:

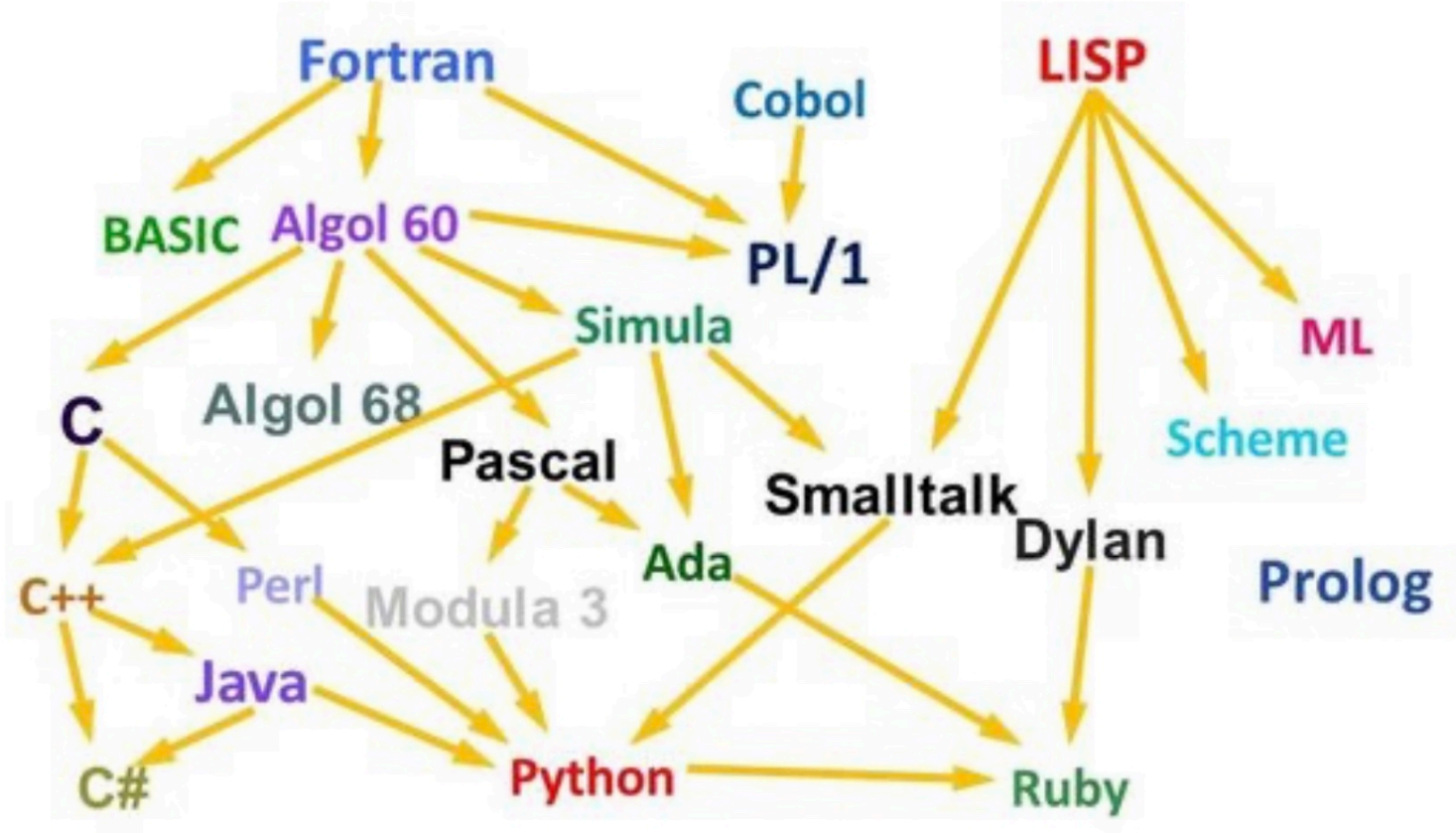
- pip3 is a command-line tool for installing Python modules.
- csv reads/writes comma-separated data files.

Many 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.
- Tensorflow is an open source machine learning platform developed by Google.

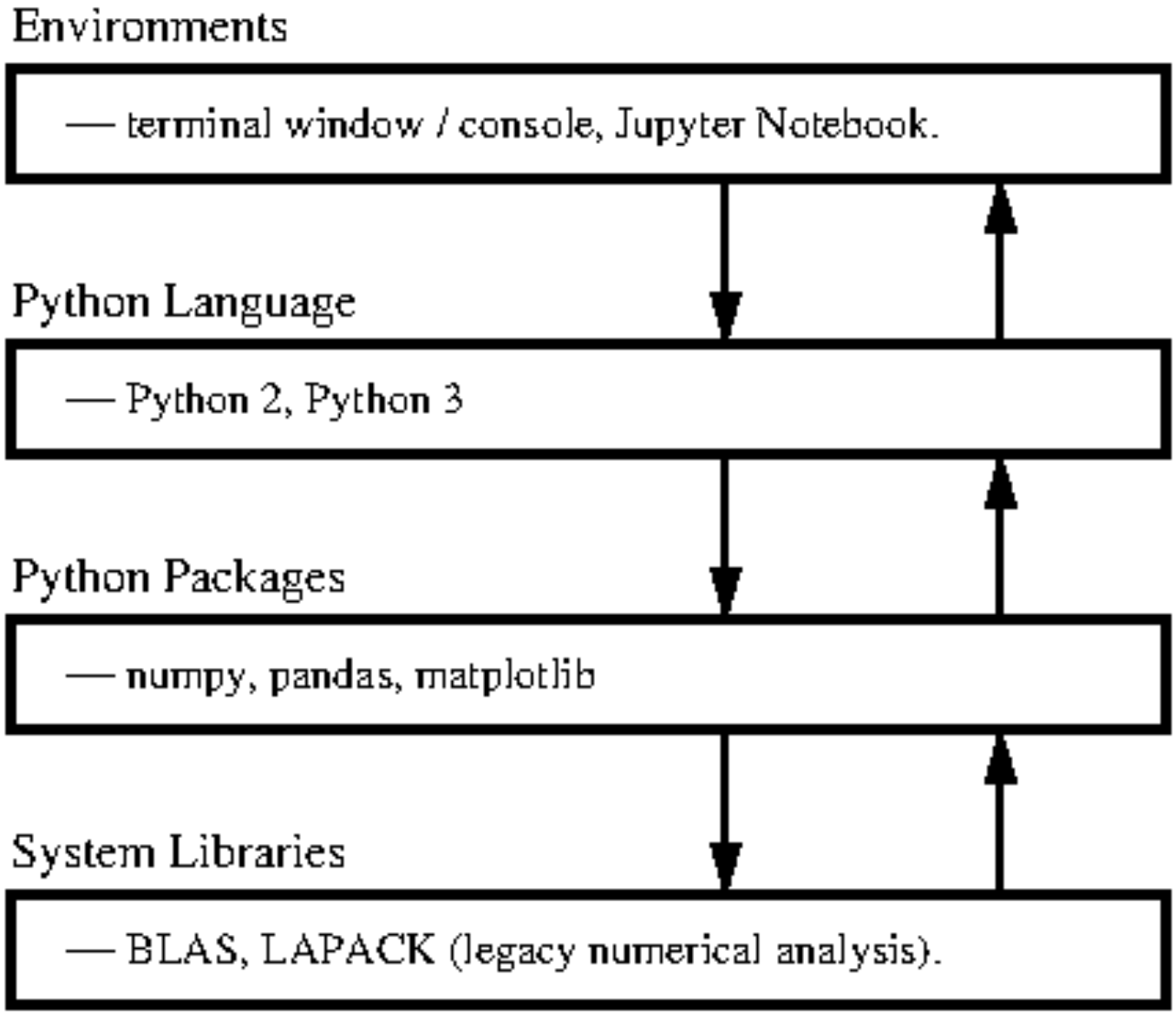
What is Python?

Graph of Feature Dependencies Among Computer Languages



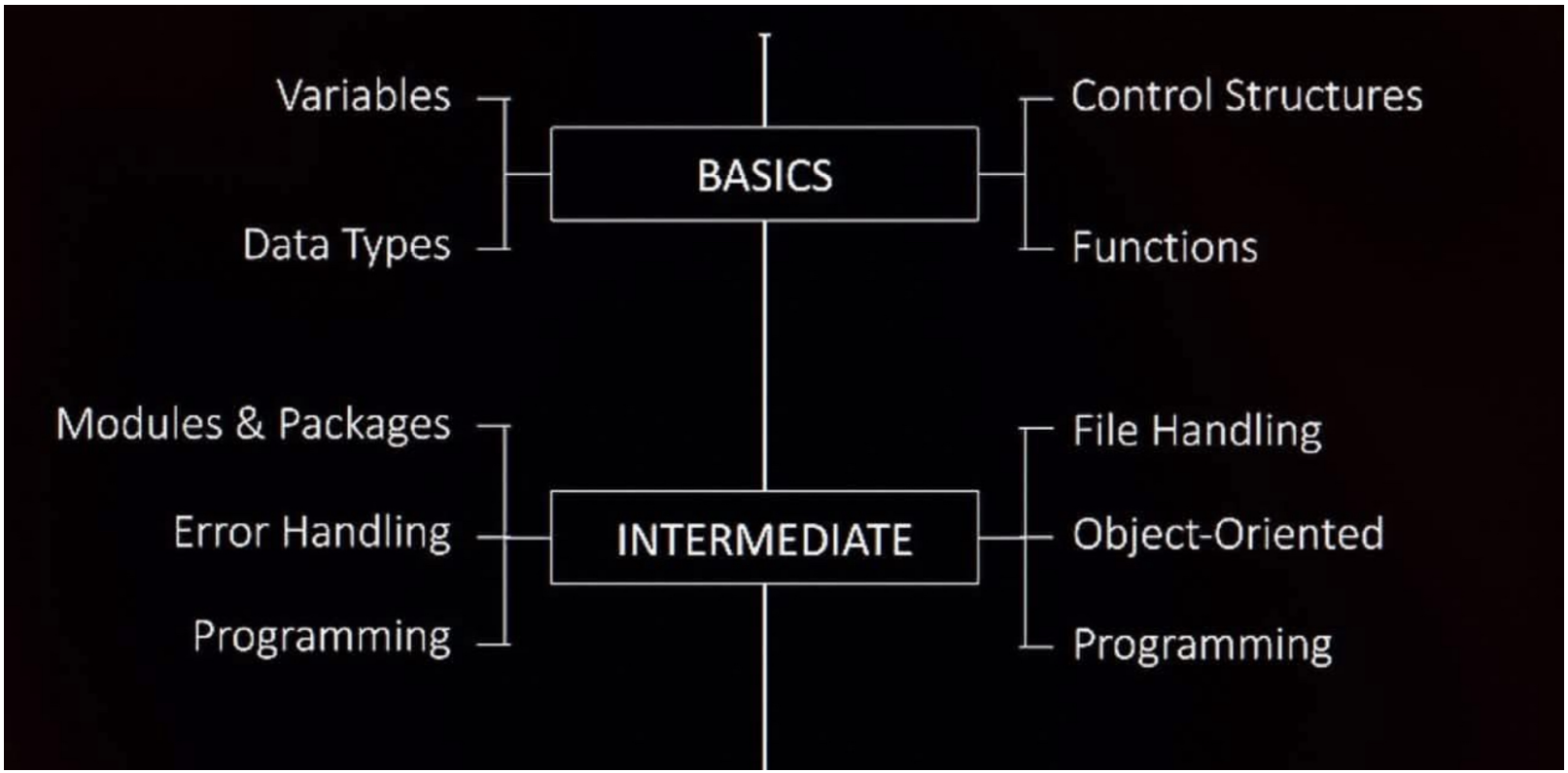
Python Language: Borrows from C++, Java, Smalltalk, ..

Framework for Scientific Computing



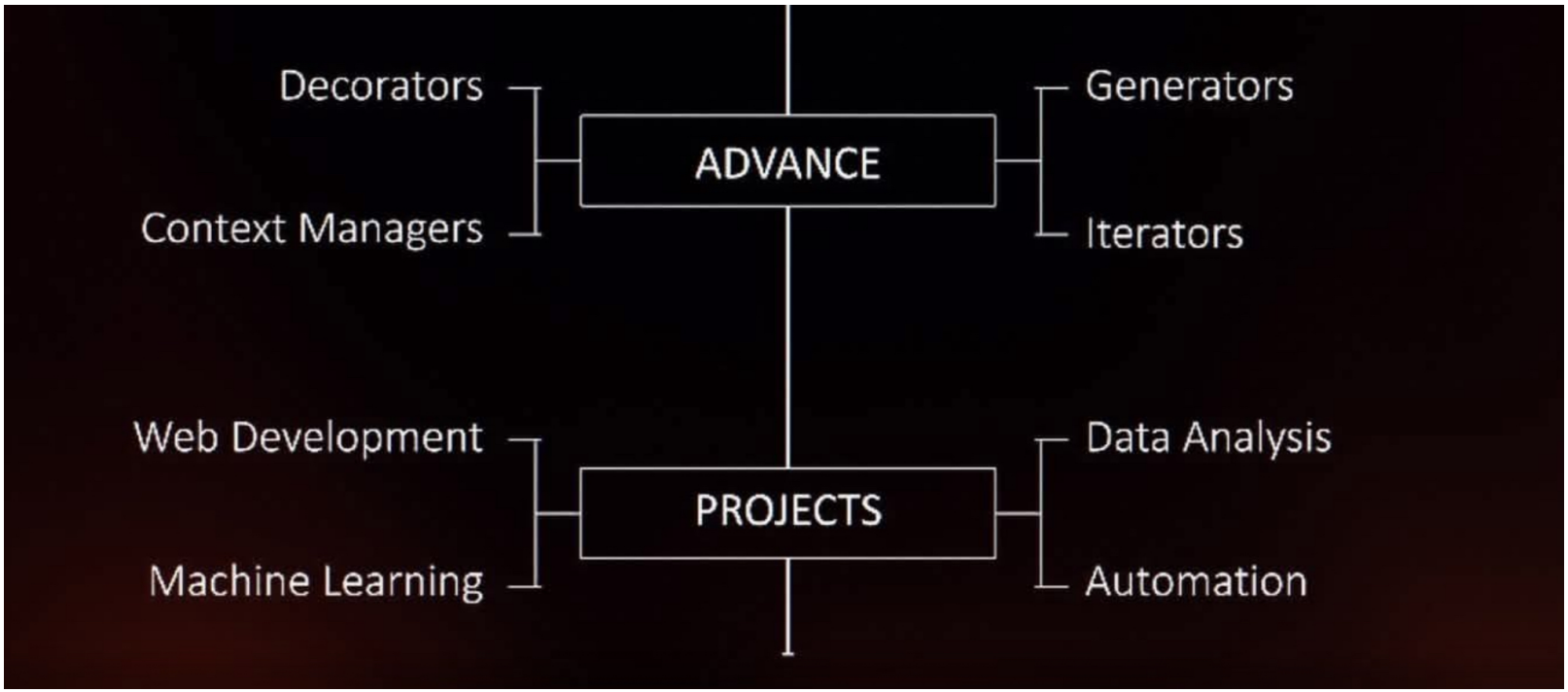
Learning Python: A Roadmap ...

Getting Started: Scope and Objectives for ENCE 201 ....



Learning Python: A Roadmap ...

More Advanced Topics: Beyond ENCE 201 ....

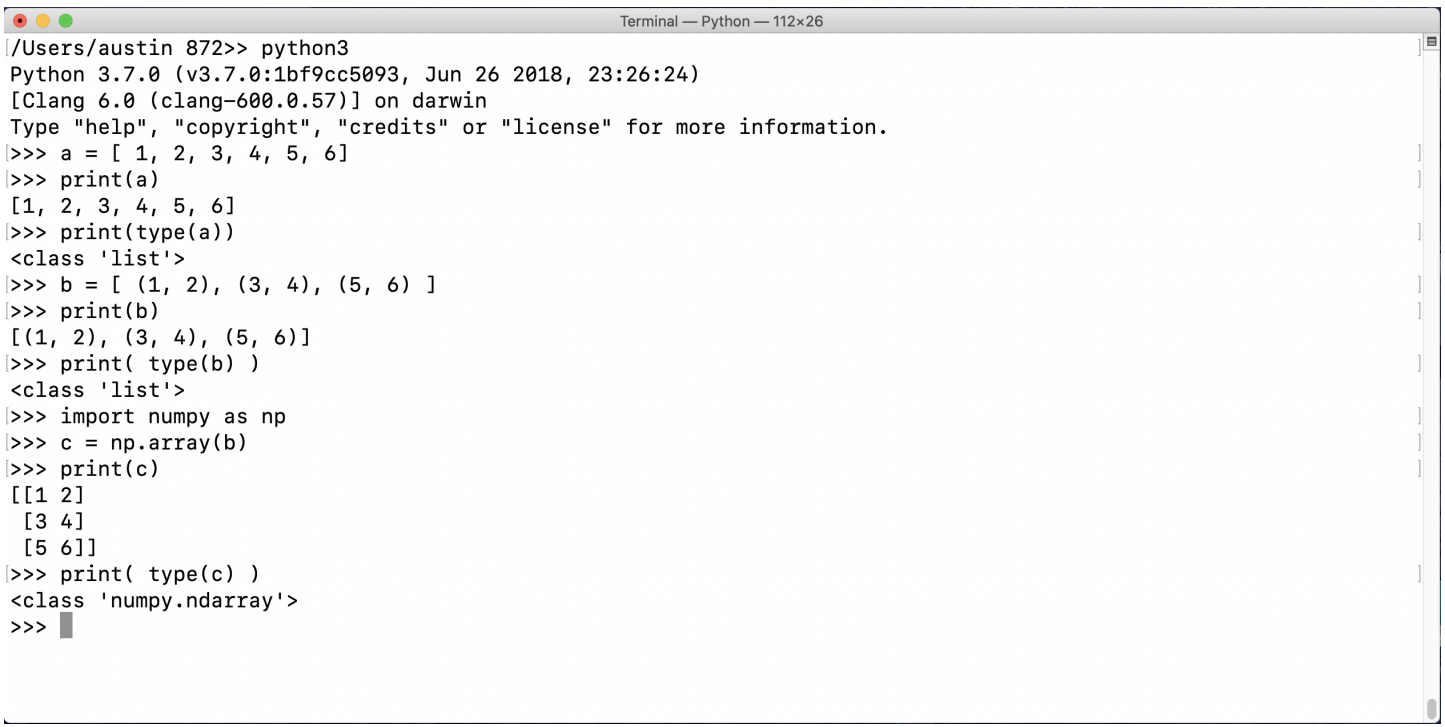


Program Development with Python

First Steps: Working with the Terminal

Terminal Window (Console)

The standard approach runs a program directly through the Python intepreter.



First Steps: Using Python as a Calculator

You can type expressions in the command window, e.g.,

```
>>> 2 + 3/4*5
5.75
>>>
```

Expressions are evaluated according to predefined priorities:

- Evaluate quantities in brackets,
- Evaluate powers $2 + 3^2 \rightarrow 2 + 9 \rightarrow 11$.
- Evaluate * /, working left to right (i.e., 3*4/5 $\rightarrow$ 12/5),
- Evaluate + -, working left to right (3+4-5 $\rightarrow$ 7-5),

Program Development

Step-by-Step Procedure:

1. Write, compile, fix, run, fix, run, validate $\rightarrow$ success!
2. Interpreted and compiled languages.

Program Control Structures:

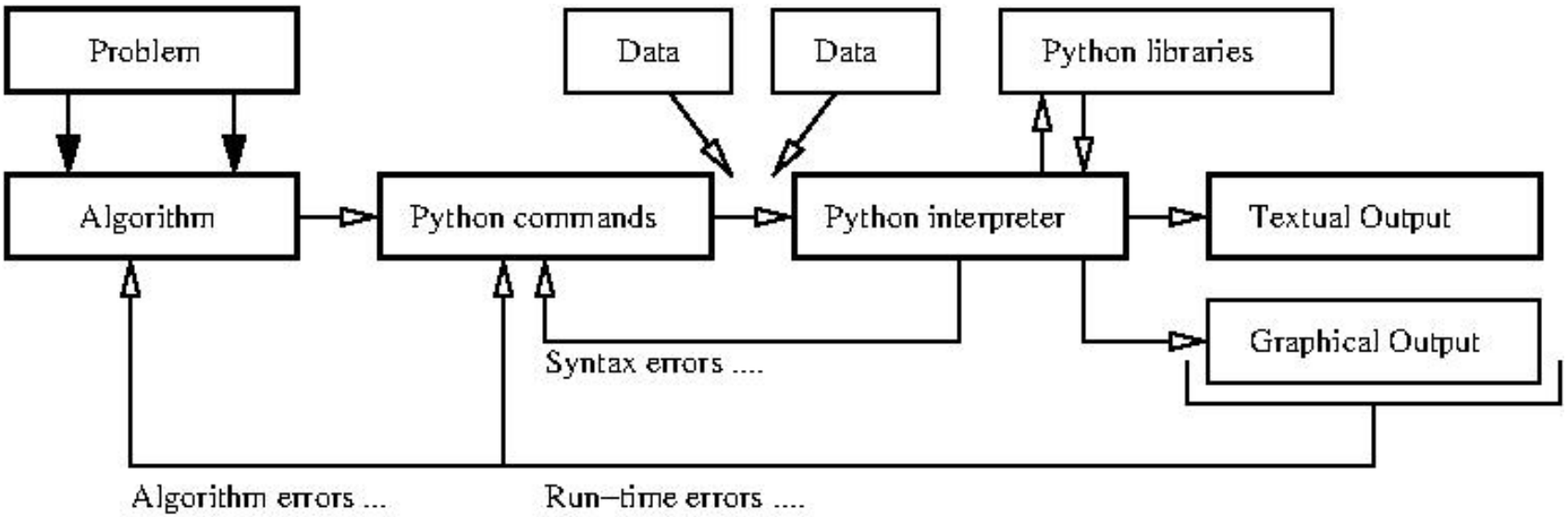
1. Logical and relational expressions
2. Selection constructs
3. Looping constructs

Program Input and Output:

1. Reading variables from the keyboard and files.
2. Formatted output of variables

First Steps: Working with the Terminal

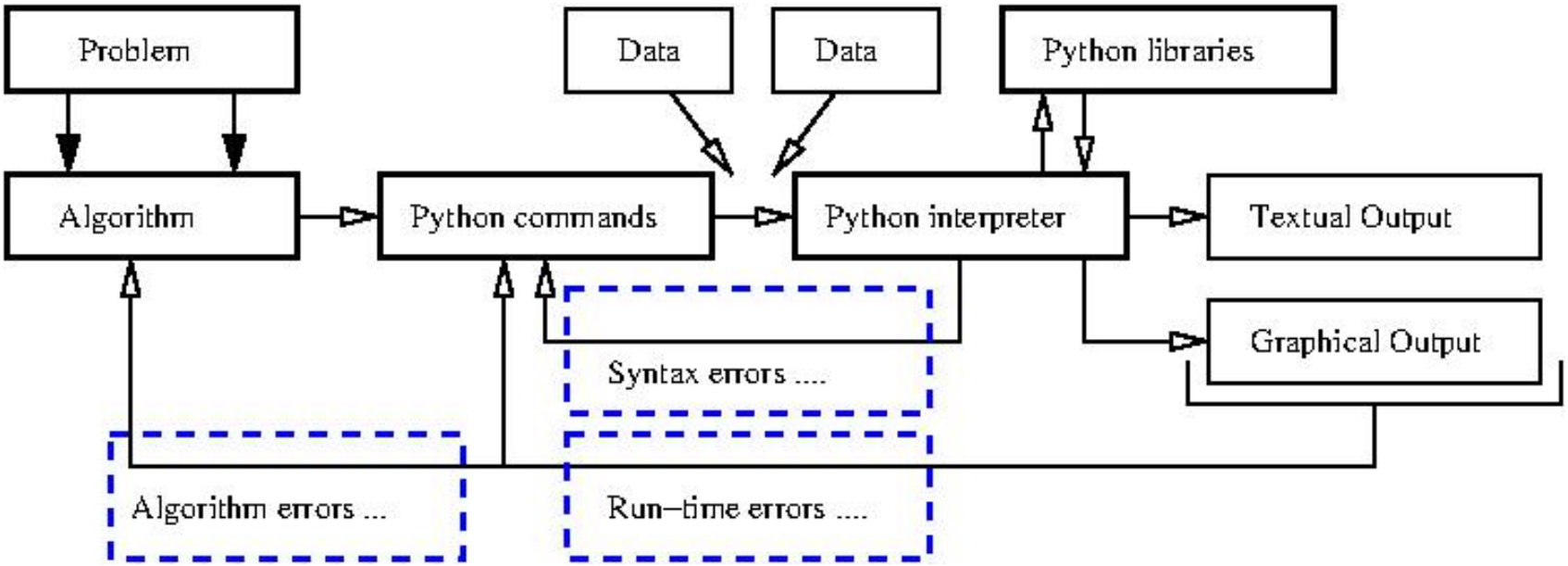
Program Development in the Terminal Window:



Step-by-Step Procedure:

1. Write, compile, fix, run, fix, run, validate $\rightarrow$ success!

First Steps: Fixing Mistakes



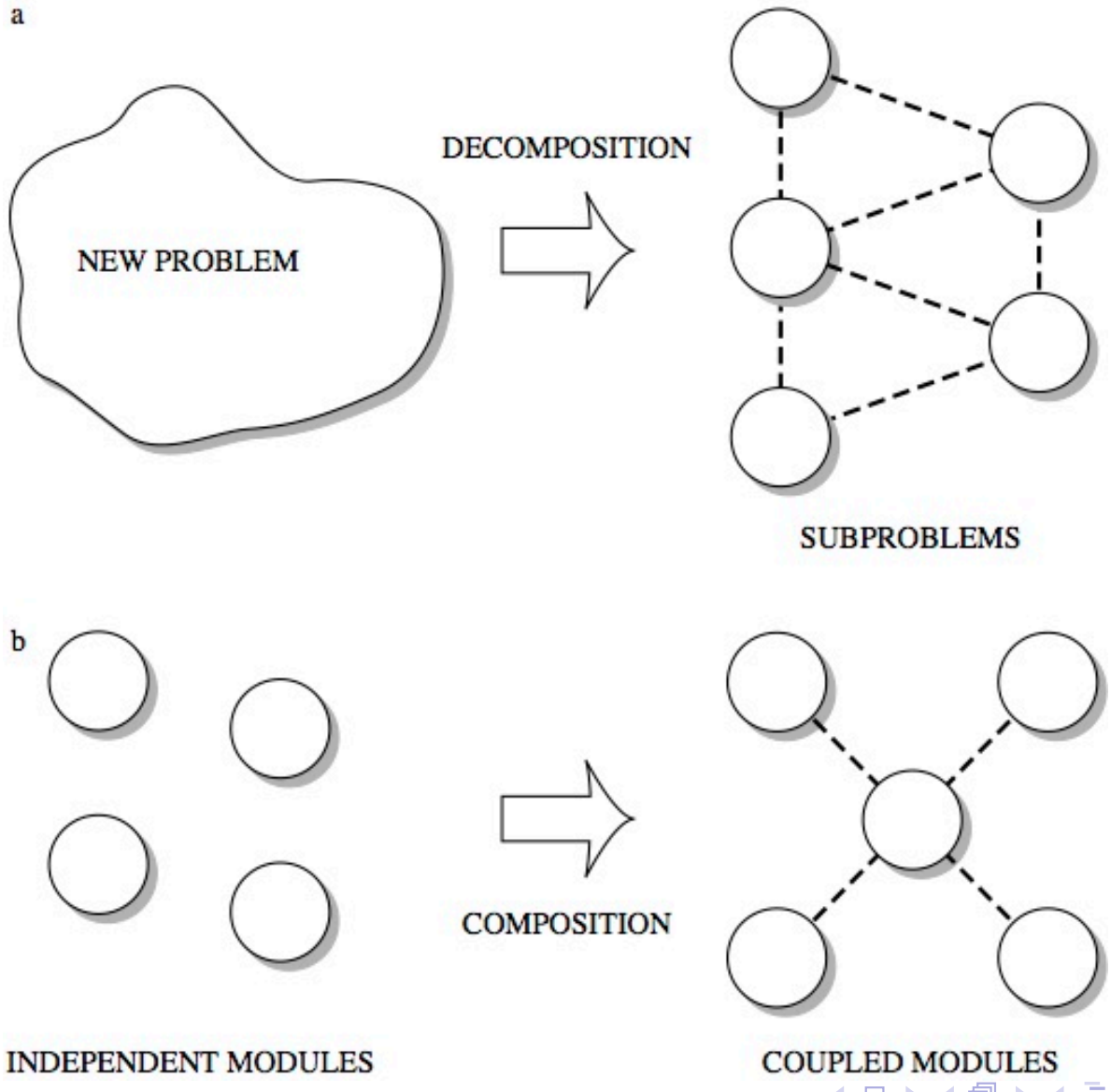
1. Syntax Errors: Check your typing ...
2. Runtime Errors: Program runs, but you have divide by zero and/or NaNs, etc.
3. Algorithm Errors: Does your program solve the right problem?

First Steps: Program Evaluation

Program Evaluation

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

Top-Down and Bottom-Up Program Design



Top-Down and Bottom-Up Program Design

Advantages/Disadvantages of Top-Down Development

- Can customize a design to provide what is needed and no more.
- Start from scratch implies slow time-to-market.

Advantages/Disadvantages of Bottom-up Development

- Reuse of components enables fast time-to-market.
- Reuse of components improves quality because components will have already been tested.
- Design may contain (many) features that are not needed.

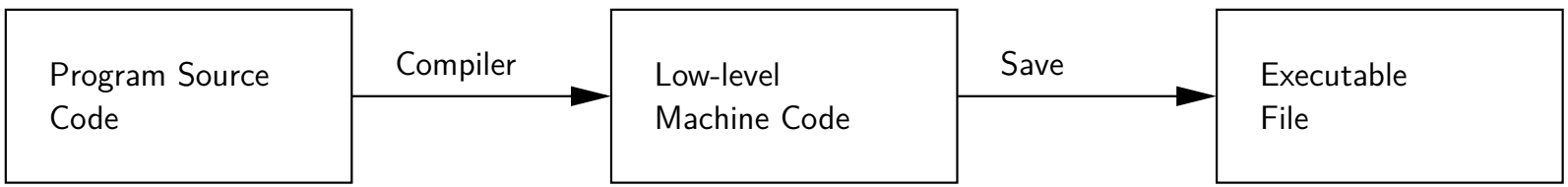
Interpreted and Compiled Programming Languages

Interpreted Programming Languages:

- High-level statements are read one by one, and translated and executed on the fly (i.e., as the program is running).

Compiled Programming Languages:

- A compiler translates the computer program source code into lower level (e.g., machine code) instructions.



- High-level programming constructs (e.g., evaluation of logical expressions, loops, and functions) are translated into equivalent low-level constructs that a machine can work with.

Interpreted and Compiled Programming Languages

Benefits of Compiled Code:

- Compiled programs generally run faster than interpreted ones.
- This is because an interpreter must analyze each statement in the program each time it is executed and then perform the desired action.

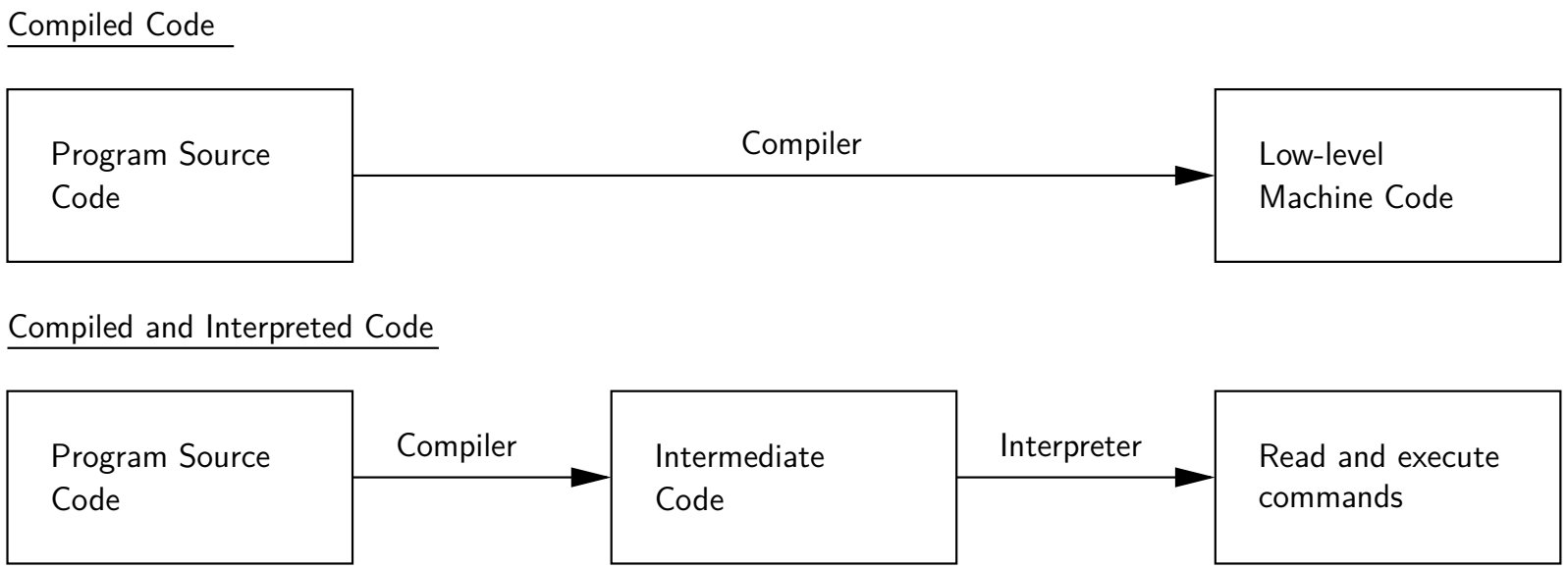
Benefits of Interpreted Code:

- Interpreted programs can modify themselves by adding or changing functions at runtime.
- Cycles of application development are usually faster than with compiled code because you don't have to recompile your application each time you want to test a small section.

Interpreted and Compiled Programming Languages

Modern Interpreter Systems

Transform source code into a lower-level intermediate format.
Interpreter then executes commands.



Examples: Java and Python (even MATLAB).

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:

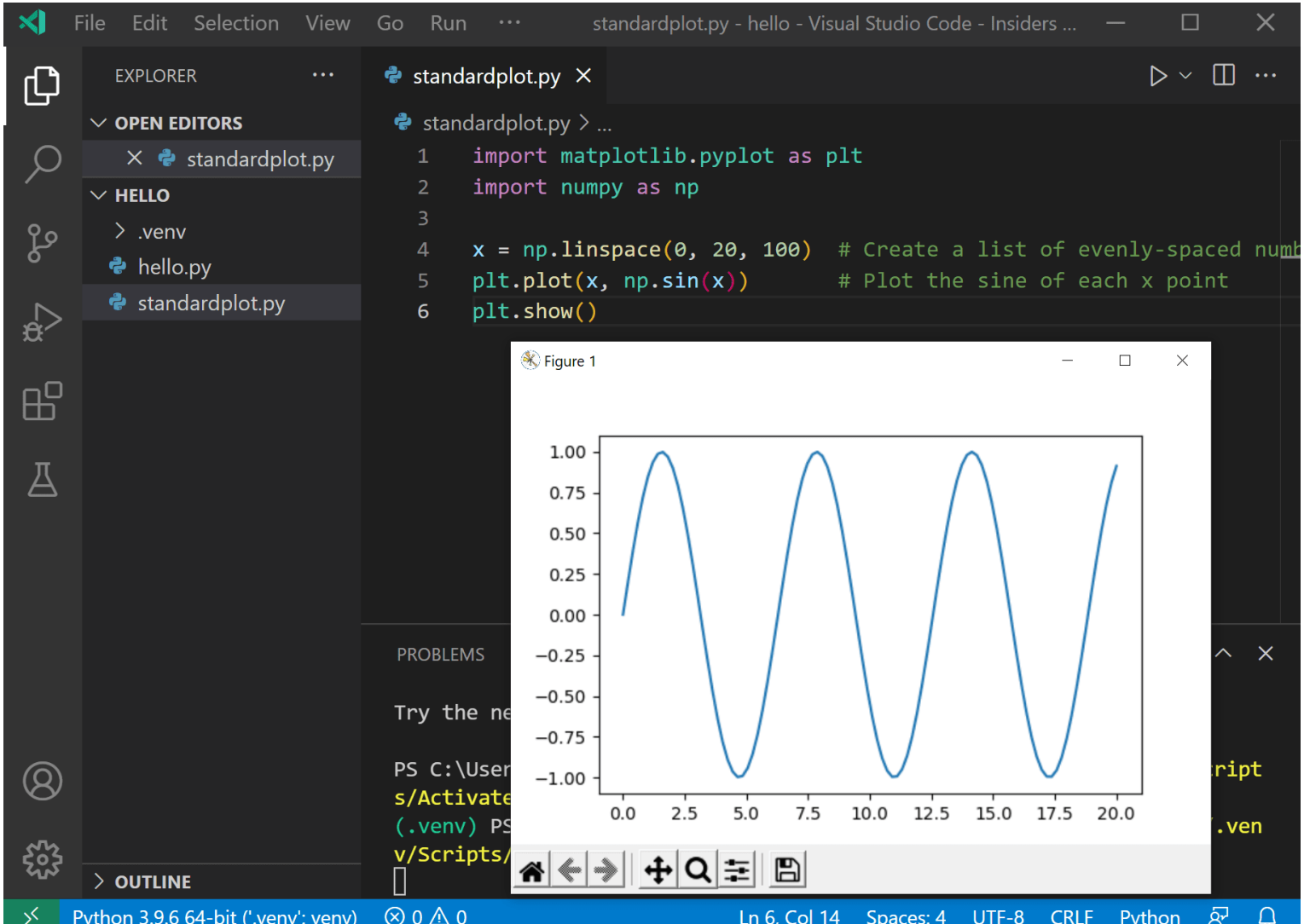
- Edit, debug, run, debug, run, test.
- Develop desktop apps.
- Numerical and scientific computing.

Advanced Use Cases:

- Deploy code to the cloud (Github).

Visual Studio Code

Graphical Interface



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

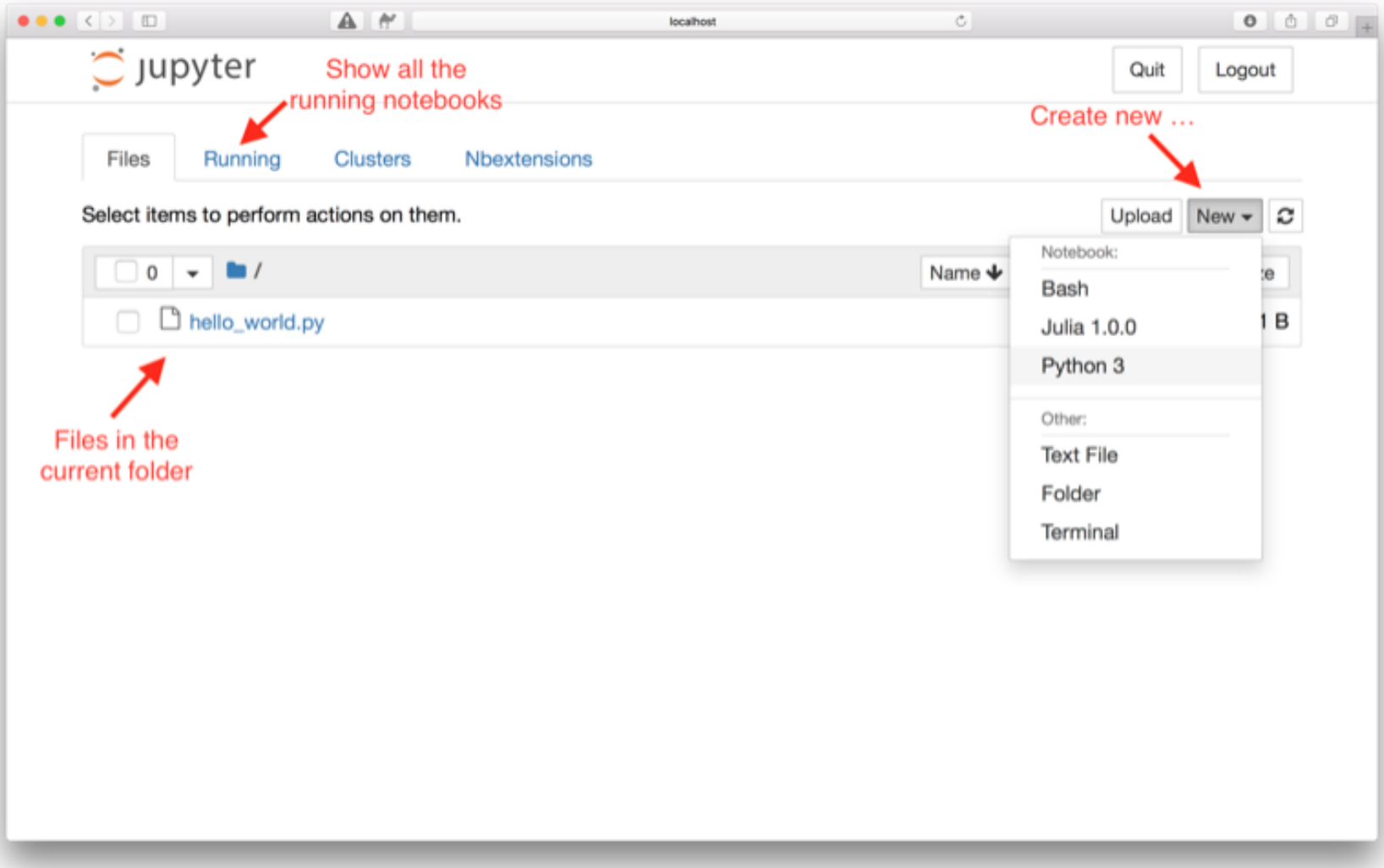
Use Cases:

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

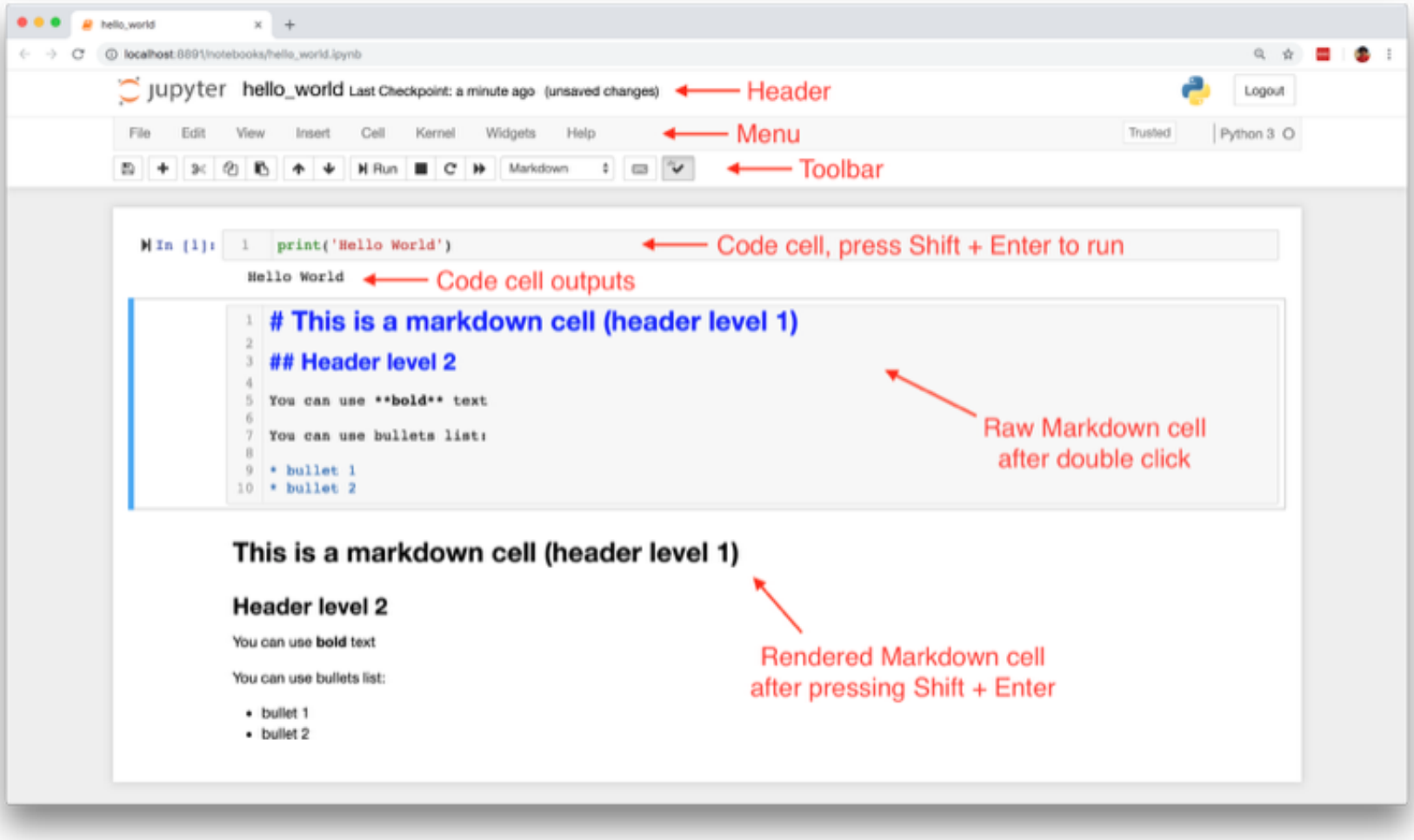
Jupyter Notebook File Format:

- 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



Jupyter Notebook Cells and Code Execution

Jupyter Notebook Cells:

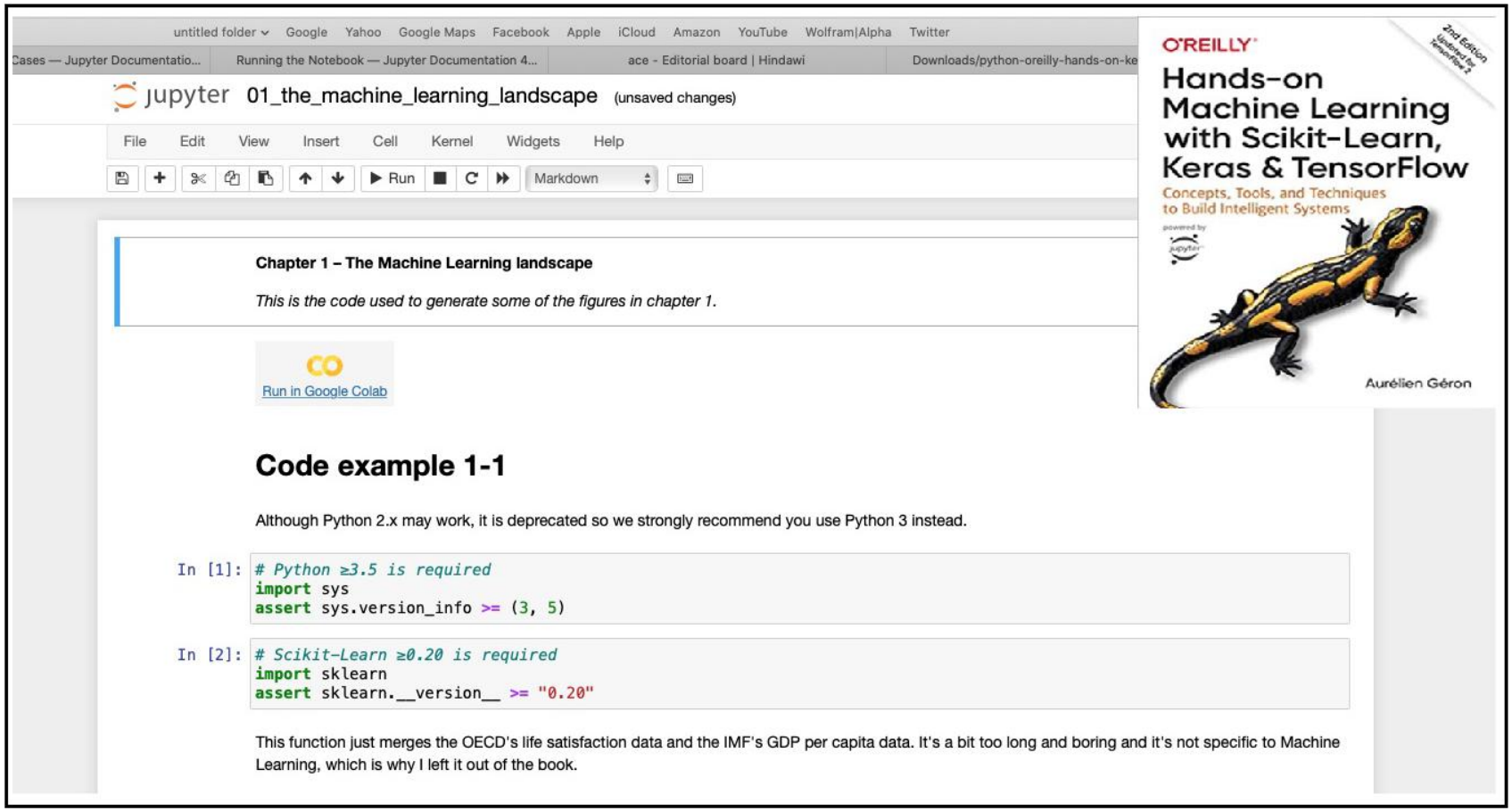
- **Code Cells:** Allows for development and editing of new code, with syntax highlighting and tab completion.
- **Markdown Cells:** Document the computational process with the Markdown language (a simple way to perform text markup). Can also include mathematics with LaTeX notion.
- **Raw Cells:** Provide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



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], "y^", 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], "y^", markersize=20)
plt.title("Activation function: sigmoid", fontsize=14)
plt.grid(True)
```