Introduction to Python

Ahmed Moustafa

Table of contents

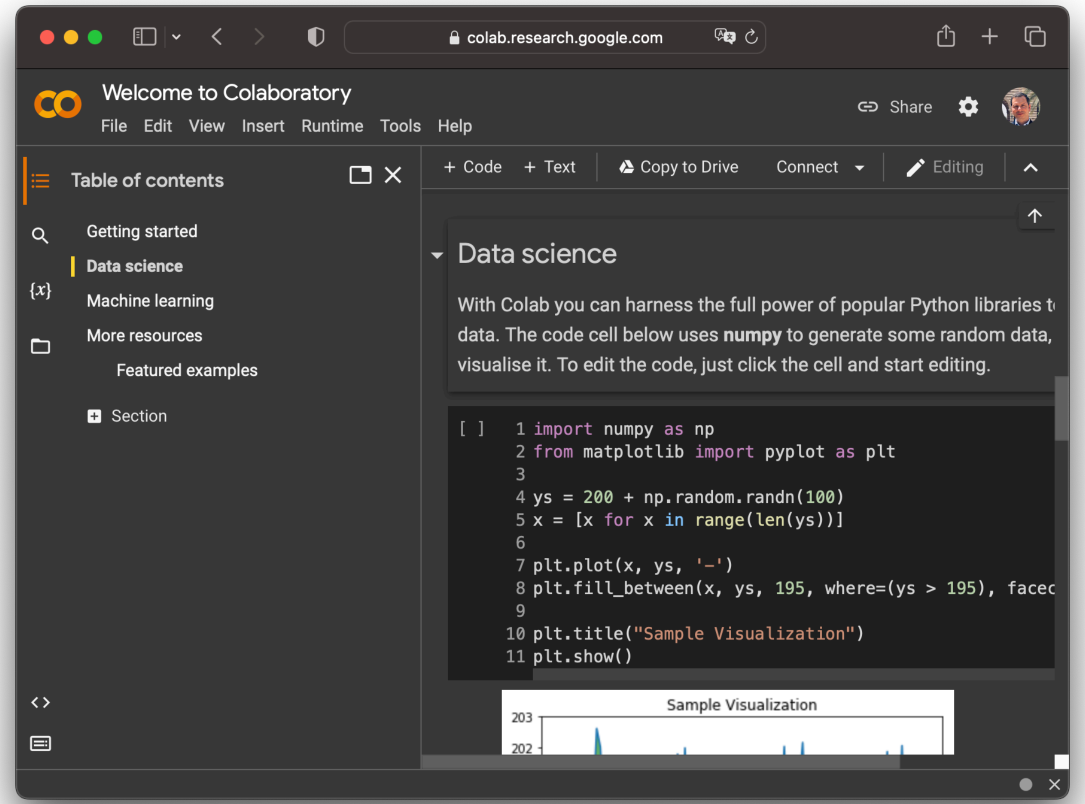
0.1 Why is it called Python?

When he began implementing Python, Guido van Rossum (left) was also reading the published scripts from Monty Python's Flying Circus (Right), a BBC comedy series from the 1970s. Van Rossum thought he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python. [Source: General Python FAQ]

Figure 1: Guido van Rossum & Monty Python's Flying Circus

0.2 Working with Python using Google Colab

Homepage: https://colab.research.google.com/ (runs online, cloud-computing like)



Data science

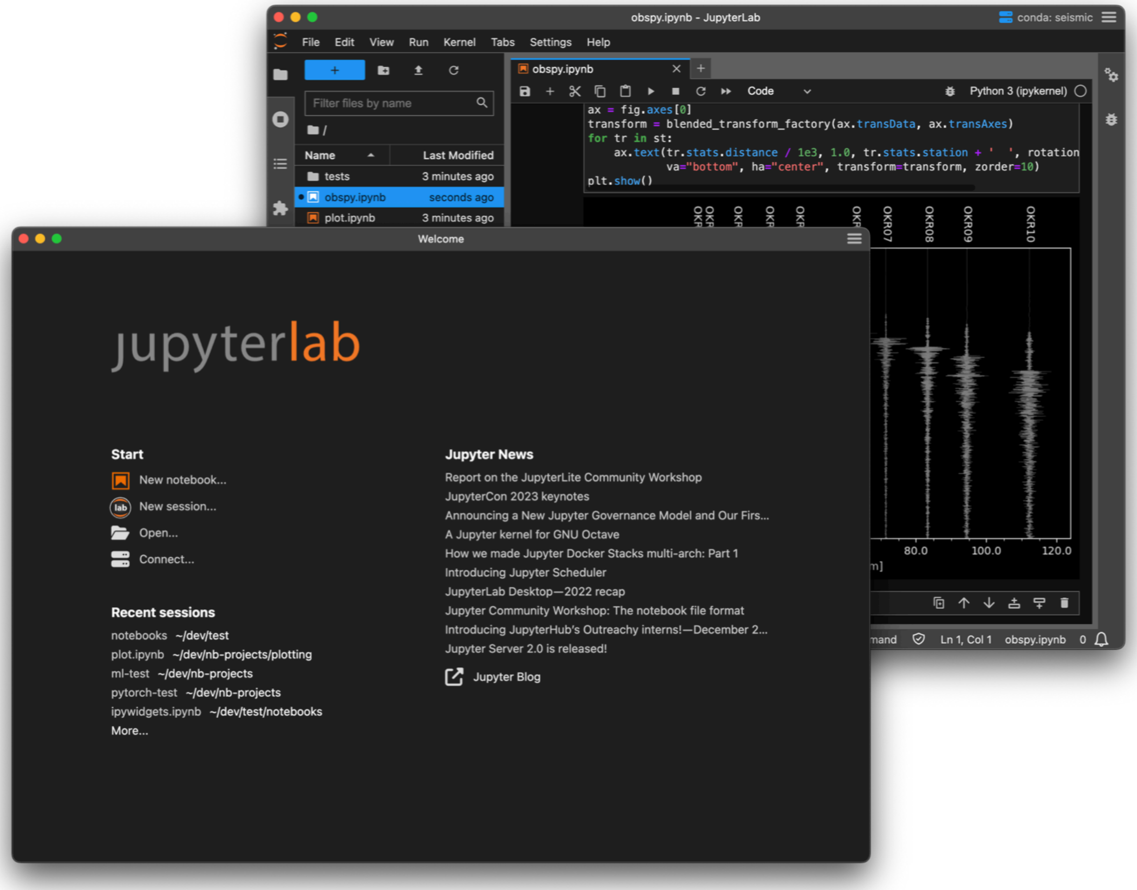
With Colab you can harness the full power of popular Python libraries t
data. The code cell below uses **numpy** to generate some random data,
visualise it. To edit the code, just click the cell and start editing.

```
import numpy as np
from matplotlib import pyplot as plt

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facec

plt.title("Sample Visualization")
plt.show()
```

Figure 2: Google Colab

0.3 Working with Python using JupyterLab Desktop

Homepage: https://github.com/jupyterlab/jupyterlab-desktop (runs offline, desktop)



```
ax = fig.axes[0]
transform = blended_transform_factory(ax.transData, ax.transAxes)
for tr in st:
    ax.text(tr.stats.distance / 1e3, 1.0, tr.stats.station + '  ', rotation
            va="bottom", ha="center", transform=transform, zorder=10)
plt.show()
```

Start

New notebook...
New session...
Open...
Connect...

Recent sessions

notebooks ~/dev/test
plot.ipynb ~/dev/nb-projects/plotting
ml-test ~/dev/nb-projects
pytorch-test ~/dev/nb-projects
ipywidgets.ipynb ~/dev/test/notebooks
More...

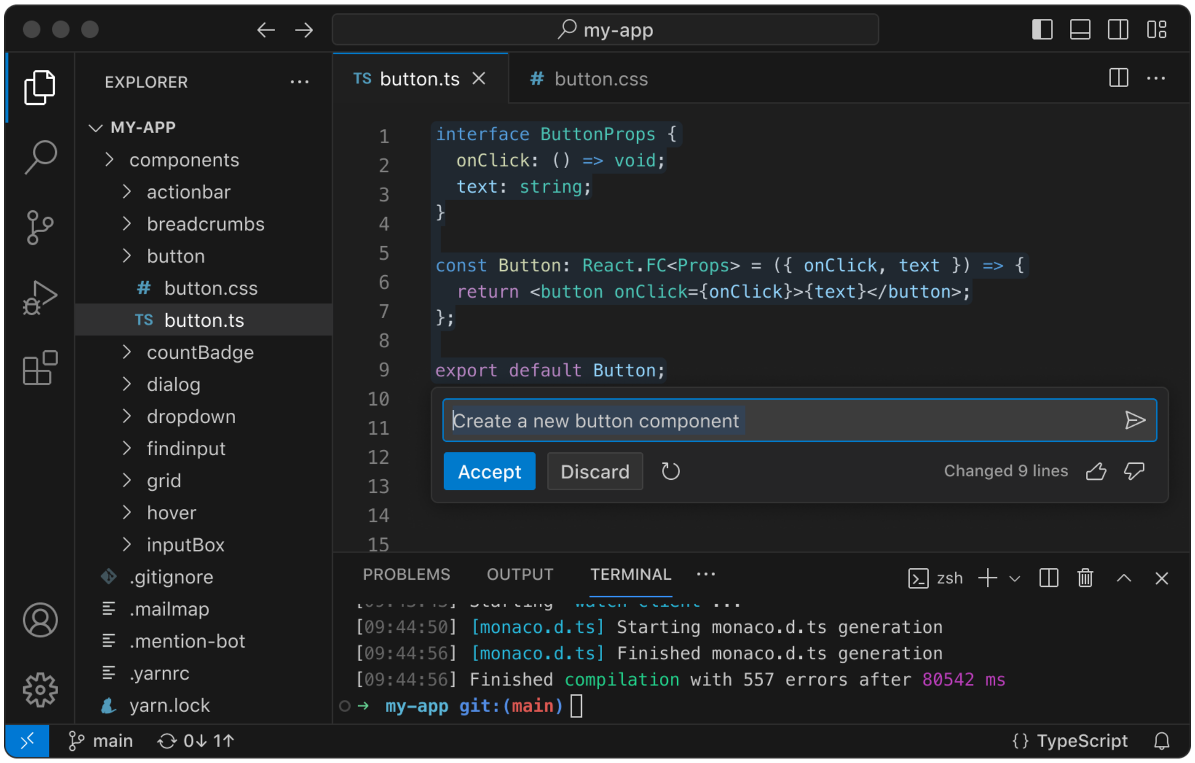
Jupyter News

Report on the JupyterLite Community Workshop
JupyterCon 2023 keynotes
Announcing a New Jupyter Governance Model and Our Firs...
A Jupyter kernel for GNU Octave
How we made Jupyter Docker Stacks multi-arch: Part 1
Introducing Jupyter Scheduler
JupyterLab Desktop—2022 recap
Jupyter Community Workshop: The notebook file format
Introducing JupyterHub's Outreachy interns!—December 2...
Jupyter Server 2.0 is released!
Jupyter Blog

Figure 3: JupyterLab Desktop

0.4 Working with Python using Visual Source Code

Homepage: https://code.visualstudio.com/ (runs offline, desktop)



```typescript
interface ButtonProps {
  onClick: () => void;
  text: string;
}

const Button: React.FC<Props> = ({ onClick, text }) => {
  return <button onClick={onClick}>{text}</button>;
};

export default Button;
```

```
[09:44:50] [monaco.d.ts] Starting monaco.d.ts generation
[09:44:56] [monaco.d.ts] Finished monaco.d.ts generation
[09:44:56] Finished compilation with 557 errors after 80542 ms
○ → my-app git:(main)
```

Figure 4: VS Code

0.5 First Things First

As with any programming course, here is the `Hello World!` in Python.

```python
print ("Hello World!")
```

```
Hello World!
```

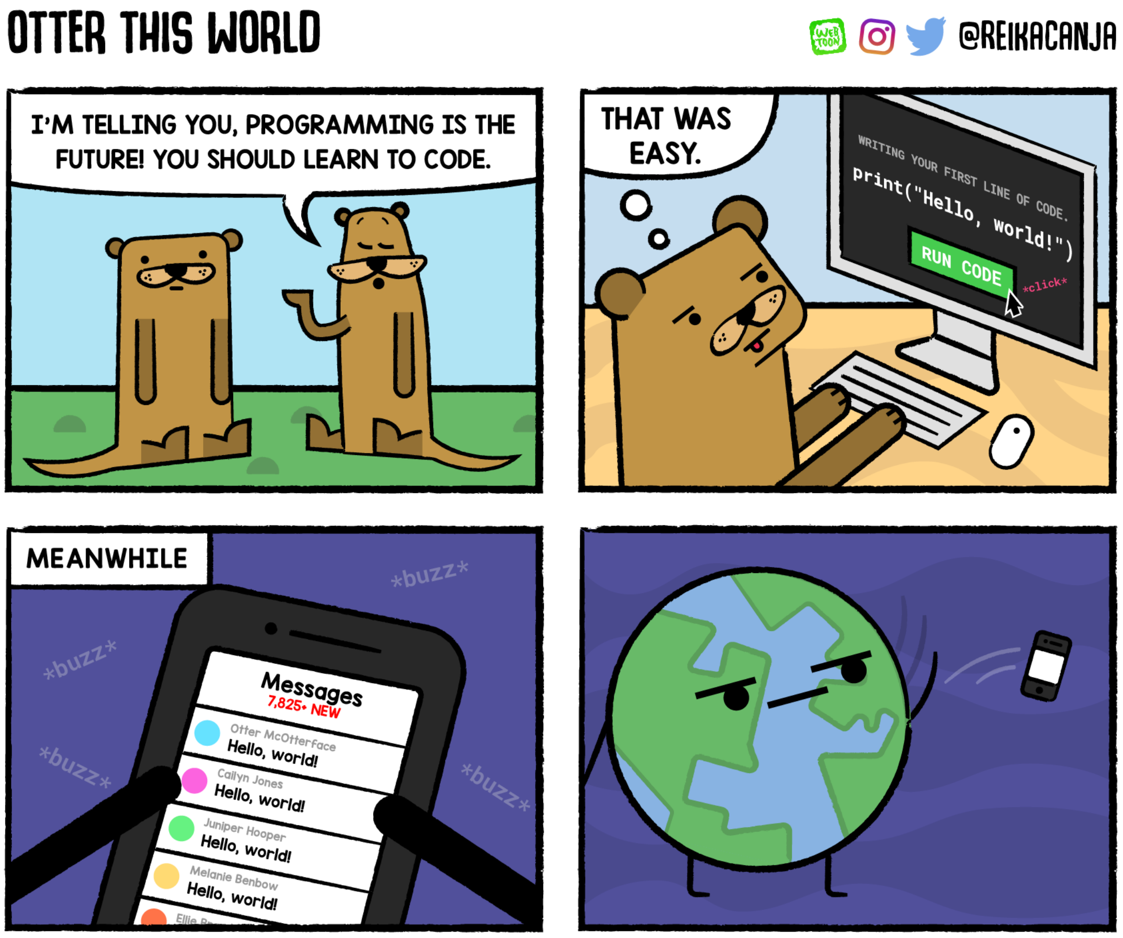


Figure 5: Hello, World!

0.6 Variables in Python

Variables are containers for storing data values. In Python, variables are created the moment you assign a value to it.

0.6.1 Example

```python
x = 5
name = "Alice"
```

0.7 Naming Conventions

- Variables names must start with a letter or an underscore.
- Can contain letters, numbers, and underscores.
- Case-sensitive (`age`, `Age`, and `AGE` are three different variables).

1 Data Types in Python

Python has various data types including:

- **Integers**: `int` (e.g., `5`)
- **Floating-point numbers**: `float` (e.g., `5.0`)
- **Strings**: `str` (e.g., `"Hello, World!"`)
- **Booleans**: `bool` (e.g., `True` or `False`)

1.1 Dynamic Typing

Python is dynamically typed, which means the type of a variable is determined at runtime.

```python
x = 4       # x is an integer
x = "Sally" # x is now a string
```

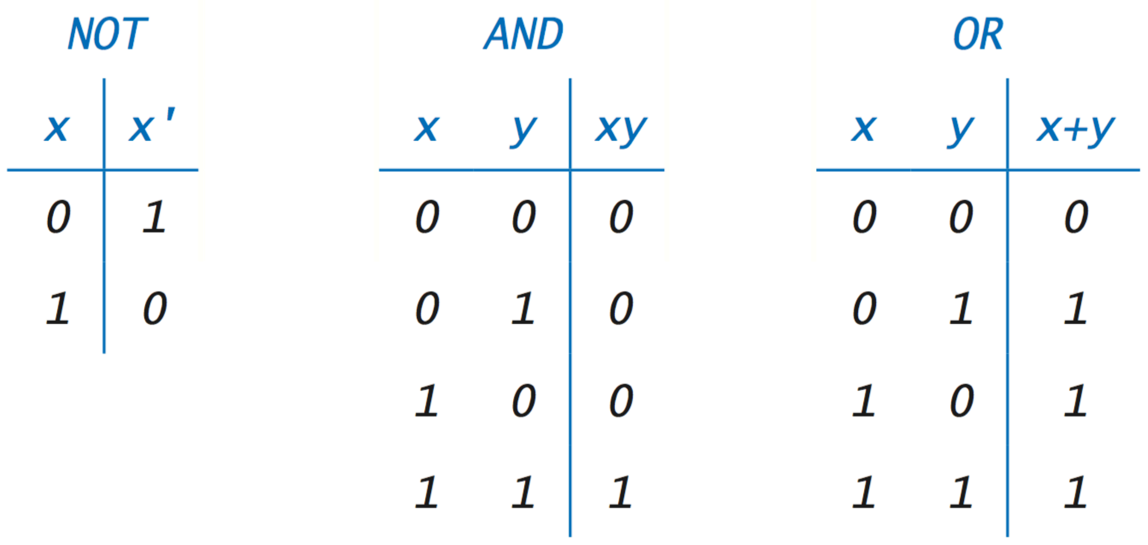
- Pros of Dynamic Typing:
  - very easy to work with
  - faster development time
- Cons of Dynamic Typing:
  - may result in unexpected bugs!

1.2 Boolean Variables and Logic Operations

Boolean variables in Python are defined by two constant objects `True` and `False`. Understanding how Boolean operations work is crucial for control flow in programming.

1.2.1 Truth Table

- `AND` operation (`True` if both are true)
- `OR` operation (`True` if at least one is true)
- `NOT` operation (Inverts the Boolean value)



NOT

x	x'
0	1
1	0

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

Figure 6: The Truth Table (Modified from Intro To Programming)

1.3 Types of Operators

- **Arithmetic Operators**: `+`, `-`, `*`, `/`, `//`, `%`, `**` for basic math operations.
- **Comparison Operators**: `==`, `!=`, `<`, `>`, `<=`, `>=` for comparing values.
- **Logical Operators**: `and`, `or`, `not` for boolean logic.
- **Assignment Operators**: `=`, `+=`, `-=`, `*=`, `/=`, etc., to assign values to variables.
- **Membership Operators**: `in`, `not in`, to check membership in sequences.

1.4 Examples

- `5 + 3` results in `8` (Arithmetic)
- `5 == 3` results in `False` (Comparison)
- `True and False` results in `False` (Logical)

1.5 Type Conversion

You can convert between different types using Python's built-in functions like `int()`, `float()`, and `str()`.

```python
int(5.4) # Converts to integer
str(20) # Converts to string
float("3.14") # Converts to float
```

1.6 Working with Strings

Strings in Python are used to handle textual data. They can be enclosed in either single quotes (`'...'`) or double quotes (`"..."`).

1.7 Operations

- Concatenation: `'Hello ' + 'world!'`
- Repetition: `'Ha' * 3`
- Indexing: `'Hello'[1]` returns `'e'`
- Slicing: `'Hello'[1:4]` returns `'ell'`

1.8 Useful String Methods

- `.upper()`, `.lower()`, `.strip()`, `.split()`, `.replace()`, `.find()`, `.join()`, `.count()`, `.startswith()`, `.endswith()`
- An even more comprehensive list of string methods in Python can be found:
  - here: Python String Functions at Digital Ocean, and
  - here: Python String Methods at Geeks for Geeks
  - BTW, both are excellent resources for additional documentation and examples.

2 Introduction to Lists

Lists in Python are used to store multiple items in a single variable. Lists are ordered, changeable, and allow duplicate values.

2.1 Creating a List

```python
my_list = [1, 2, 3]
names = ["Alice", "Bob", "Charlie"]
```

2.2 Accessing Elements

- Access by index: `my_list[0]` returns `1`.
- Slicing: `names[1:3]` returns `['Bob', 'Charlie']`.

2.3 List Operations

- Append: `my_list.append(4)`
- Remove: `my_list.remove(1)`
- Sort: `names.sort()`

3 Advanced Variable Usage

Understanding variable scopes and mutability is crucial for Python programming.

3.1 Variable Scope

- **Global Scope**: Variables defined at the top-level of a script or module are global.
- **Local Scope**: Variables created within a function are local to that function.

3.2 Mutability

- Immutable types: `int`, `float`, `bool`, `str`. Changing the value creates a new object.
- Mutable types: `list`, `dict`, `set`. They can be changed in place without creating a new object.

4 In-depth Data Types

Exploring Python's built-in data types reveals the language's flexibility.

4.1 Sequences

- **Tuples**: Immutable and ordered. `my_tuple = (1, 2, 3)`
- **Ranges**: Immutable sequence of numbers. `range(1, 10)`

4.2 Mapping Type

- **Dictionaries**: Key-value pairs. `my_dict = {"name": "Alice", "age": 30}`

4.3 Set Types

- **Sets**: Unordered collection of unique elements. `my_set = {1, 2, 3}`
- **Frozen Sets**: Immutable version of a set. `frozen_set = frozenset([1, 2, 3])`

4.4 Escape Characters

- Use \ to insert special characters, e.g.,
- new line:

```python
print ("Hello\nWorld!")
```

```
Hello
World!
```

- tab:

```python
print ("Hello\tWorld!")
```

```
Hello   World!
```

4.5 Summary

- Python is awesome
- Python uses **dynamic typing**
- Parentheses `( )` are for calling **functions**
- Square brackets `[ ]` are are indexing **lists**
- Strings are **immutable** lists
- Lists start indexing at **zero**