TITLE: Mastering Matplotlib: The Art of Line Graphs

SUBTITLE: Guidebook 1: From Basic Plots to Professional Visualizations

SERIES: *Python Data Visualization Series (Book 1 of 3)*

AUTHOR: Faizan Toheed

DATE: January 19, 2026

ABSTRACT: This guide focuses exclusively on the foundational skill of data visualization: the Line Graph. You will learn how to visualize trends, customize line thickness, add professional labels, and use built-in style sheets to create publication-quality charts in under 10 lines of code.

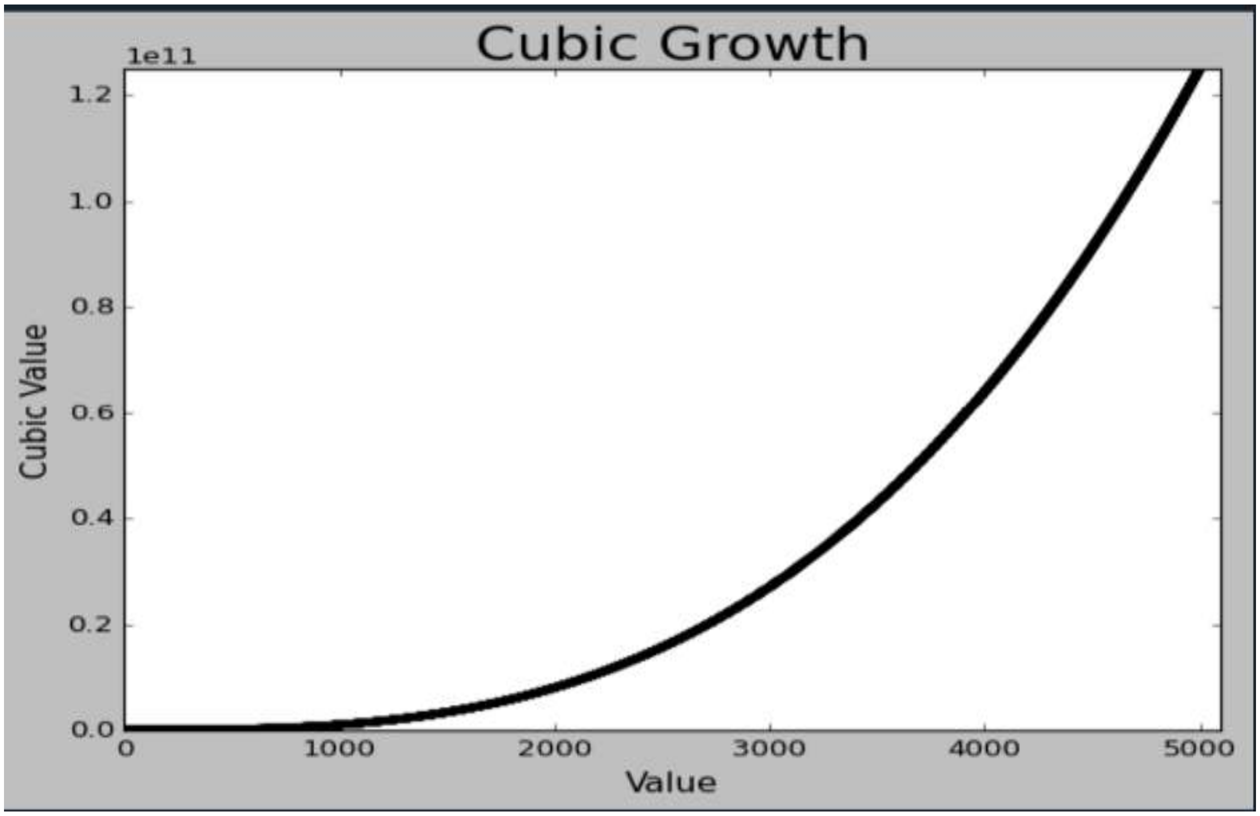
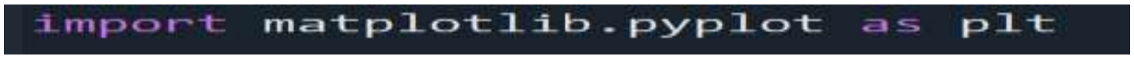


Table of Contents

1. The Setup: Importing Matplotlib

Before we draw anything, we need to import the pyplot module. This module contains all the functions we need to generate charts.

- **Standard Import:**

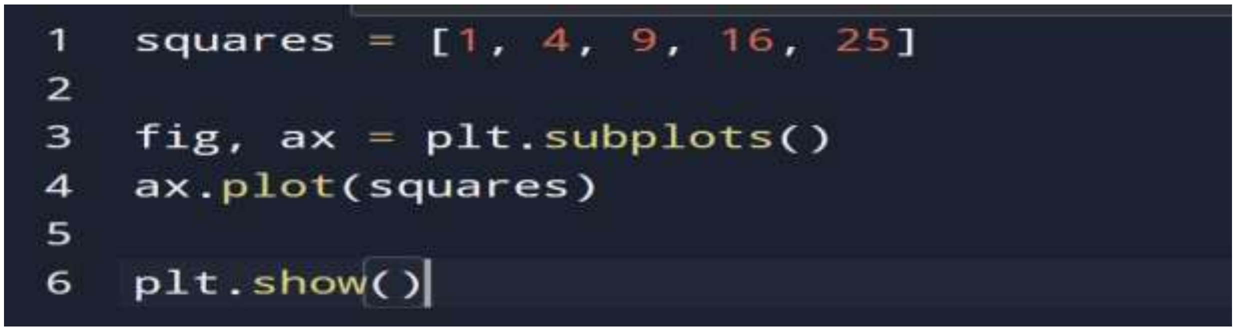


```python
import matplotlib.pyplot as plt
```

Note: We use the alias plt so we don't have to type matplotlib.pyplot every time.

Step1: Drawing Your First Line

The simplest graph consists of a single list of numbers. Matplotlib assumes these numbers are the Y-values (heights) and automatically generates the X-values (0, 1, 2, 3...) for you.



```python
squares = [1, 4, 9, 16, 25]

fig, ax = plt.subplots()
ax.plot(squares)

plt.show()
```

- **fig**: Represents the entire window or collection of plots.
- **ax**: Represents the single plot (the axis) we are drawing on.
- **ax.plot()**: The function that connects the points with a line.

Step 2: Enhancing Readability (Titles & Labels)

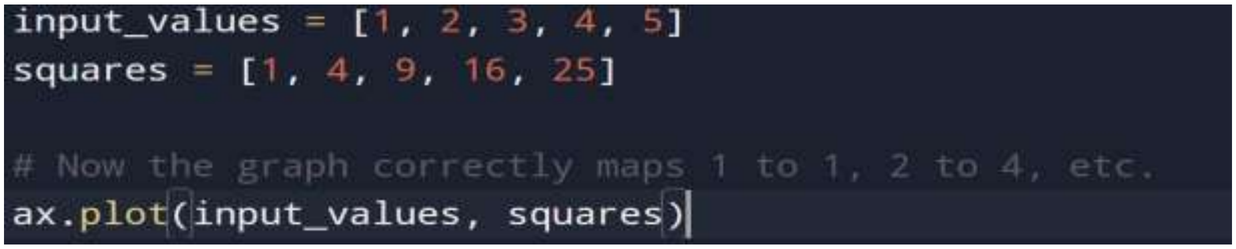
A graph without labels is meaningless. We use "Setter" methods to add context.

- **Title:** ax.set_title("Square Numbers", fontsize=24)
- **X-Axis:** ax.set_xlabel("Value", fontsize=14)
- **Y-Axis:** ax.set_ylabel("Square of Value", fontsize=14)

Step 3: Correcting the Data (The Index Error)

In Step 1, if you looked closely, the number 4 was plotted at index 1. But $2^2 = 4$, so it should be at index 2. This happens because Python lists start at 0.

To fix this, we must provide **both** input (X) and output (Y) values.



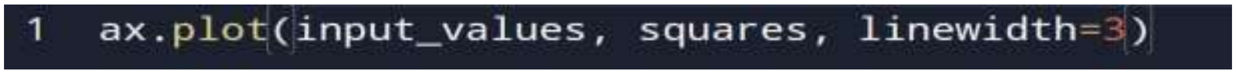
```
input_values = [1, 2, 3, 4, 5]
squares = [1, 4, 9, 16, 25]

# Now the graph correctly maps 1 to 1, 2 to 4, etc.
ax.plot(input_values, squares)
```

Step 4: Visual Styling (Thickness & Fonts)

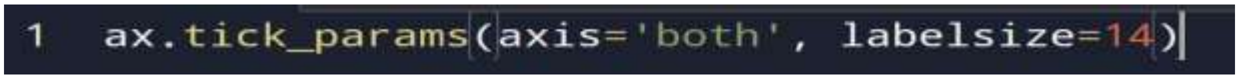
To make the chart look professional, we can adjust the line thickness and the size of the tick marks (the numbers on the rulers).

- **Line Thickness:** Inside plot(), add the linewidth argument.



```
ax.plot(input_values, squares, linewidth=3)
```

- **Tick Size:** Use tick_params to make the numbers on the axes readable.

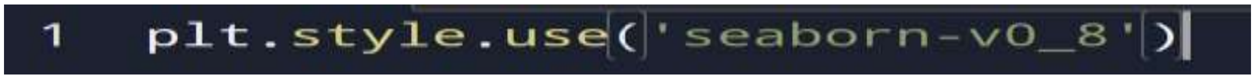


```
ax.tick_params(axis='both', labelsize=14)
```

Step 5: Using Built-in Themes

Matplotlib comes with pre-defined styles (like 'seaborn' or 'classic') that instantly change the background grid, fonts, and colors.

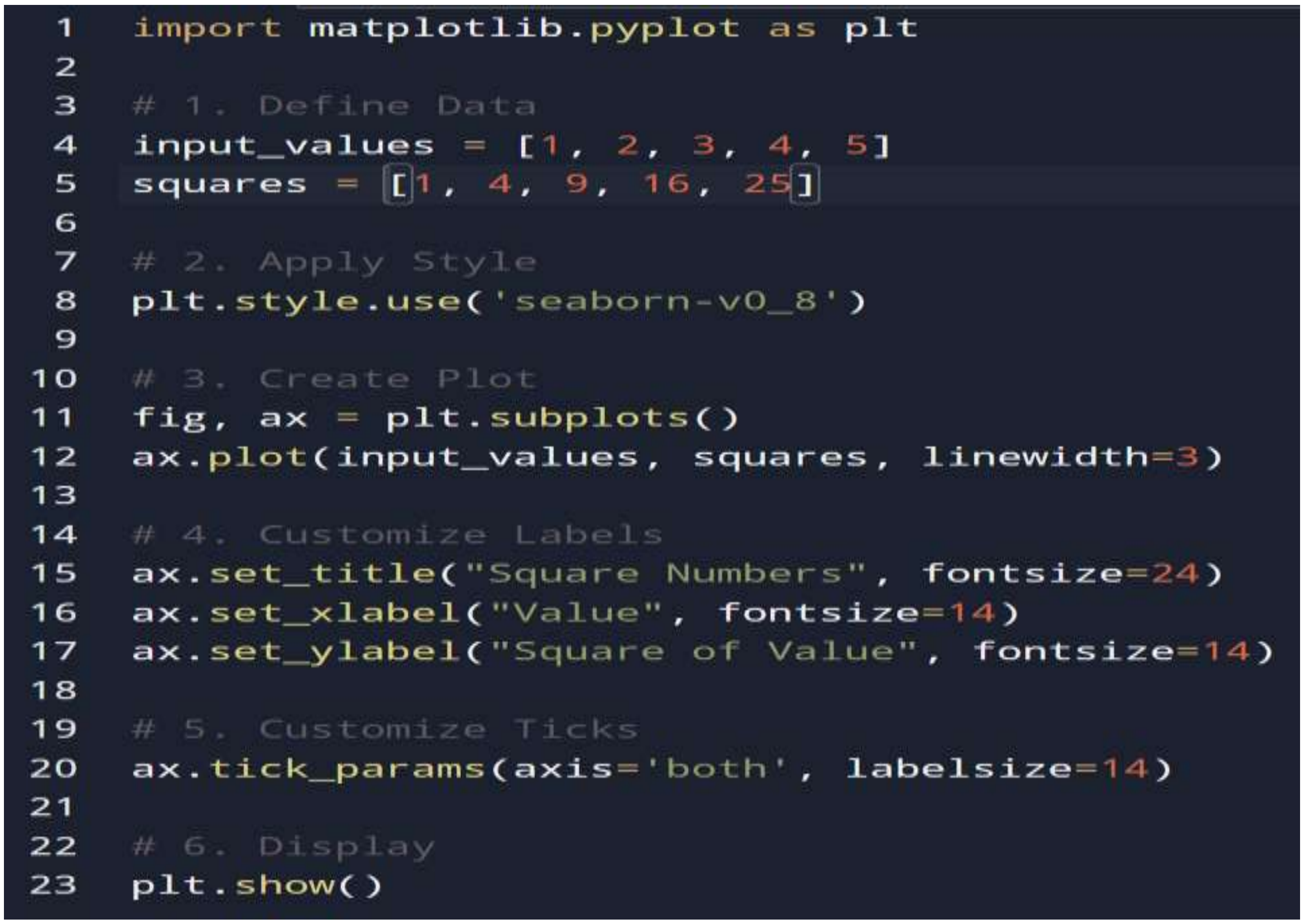
- **View available styles:** print(plt.style.available)
- **Apply a style:** (Must be done *before* creating the plot)



```
plt.style.use('seaborn-v0_8')
```

Summary: The Complete Code Block

Here is the final, polished code combining all the steps above.



```python
import matplotlib.pyplot as plt

# 1. Define Data
input_values = [1, 2, 3, 4, 5]
squares = [1, 4, 9, 16, 25]

# 2. Apply Style
plt.style.use('seaborn-v0_8')

# 3. Create Plot
fig, ax = plt.subplots()
ax.plot(input_values, squares, linewidth=3)

# 4. Customize Labels
ax.set_title("Square Numbers", fontsize=24)
ax.set_xlabel("Value", fontsize=14)
ax.set_ylabel("Square of Value", fontsize=14)

# 5. Customize Ticks
ax.tick_params(axis='both', labelsize=14)

# 6. Display
plt.show()
```