How to Easily Find and Use Matplotlib’s Default Colors

Authored by

stats writer

November 28, 2025

In the realm of data visualization, the choice of color is foundational to conveying information effectively. While custom palettes offer precise control, understanding the built-in defaults of a plotting library is essential for efficient analysis. This comprehensive guide serves as an authoritative reference detailing the default color scheme utilized by Matplotlib, the foundational plotting library for the Python programming language. We will explore the specific sequence of colors, often referred to as the default **Color Cycle**, including their precise Hex codes and typical application in multi-series plots.

The default colors in Matplotlib are not randomly chosen; they are part of a carefully curated palette designed for visual distinctness and accessibility. Knowing this sequence--which includes the underlying RGB codes, names, and hexadecimal representations--allows users to predict the appearance of their plots without manual specification. This documentation is crucial whether you are creating simple line graphs, complex scatter plots, or sophisticated statistical visualizations, ensuring that your data narratives are clear and consistent from the outset.

Understanding Matplotlib's Default Color Philosophy

When generating charts in Matplotlib, the library employs a predefined sequence of colors to differentiate between multiple elements on the same axes. This mechanism is known as the **property cycle**, specifically the Color Cycle. This design decision is critical: if a user plots multiple lines or groups of data points without explicitly assigning a color to each, Matplotlib automatically iterates through this default list. This ensures that every element receives a unique, distinguishable color until the cycle repeats, thereby maximizing clarity in complex analytical figures.

The standard default color set used since Matplotlib version 2.0 is based on the 'tab10' palette. This palette was introduced to replace the older, less perceptually uniform color scheme, offering colors that are generally more distinct, even when viewed by individuals with common types of color blindness. The philosophy behind this cycle is to provide immediate visual separation for the first several data series, preventing confusion and enhancing the overall clarity of complex plots. The order is optimized so that the first few colors (e.g., blue, orange, green) are maximally distinct from one another, adhering to best practices in visualization design.

Understanding where this color cycle resides within the library's configuration is key to advanced customization. These default settings are stored within Matplotlib's comprehensive configuration dictionary, accessible via `plt.rcParams`. By inspecting or modifying the `axes.prop_cycle` parameter, advanced users can change the entire default palette, defining not just colors, but also line styles, markers, and other aesthetic properties that cycle through plots automatically. This foundational knowledge is essential for maintaining aesthetic consistency across large analytical projects or adapting plots for specific publishing requirements.

Visualizing the Default Color Sequence

To fully appreciate the sequence of colors Matplotlib utilizes, it is instructive to visualize them in action. The following Python code snippet generates a simple plot containing ten distinct lines, thereby consuming the first ten colors in the default cycle. This demonstrates how Matplotlib handles iterating through the sequence when multiple data series are added to a single set of axes. Notice the use of NumPy for array generation and the systematic way the lines are plotted using a loop, which implicitly relies on the default color assignment mechanism to assign a unique color to each line object.

The code dynamically retrieves the color assigned to the line object using the `line.get_color()` method. This confirms that even when the color is not explicitly passed to the plotting function, Matplotlib assigns a specific value from its internal cycle, which we then retrieve and display. This robust automatic assignment mechanism significantly simplifies the coding process, especially when iterating over unknown numbers of data sets, removing the need for explicit color mapping in initial analyses.

Below is the executable Python code demonstrating the first ten colors in the default Color Cycle. We observe how the plot function automatically handles the sequencing of colors for each iteration of the loop, resulting in a visually distinct line for every data series generated, which is crucial for comparative analysis.

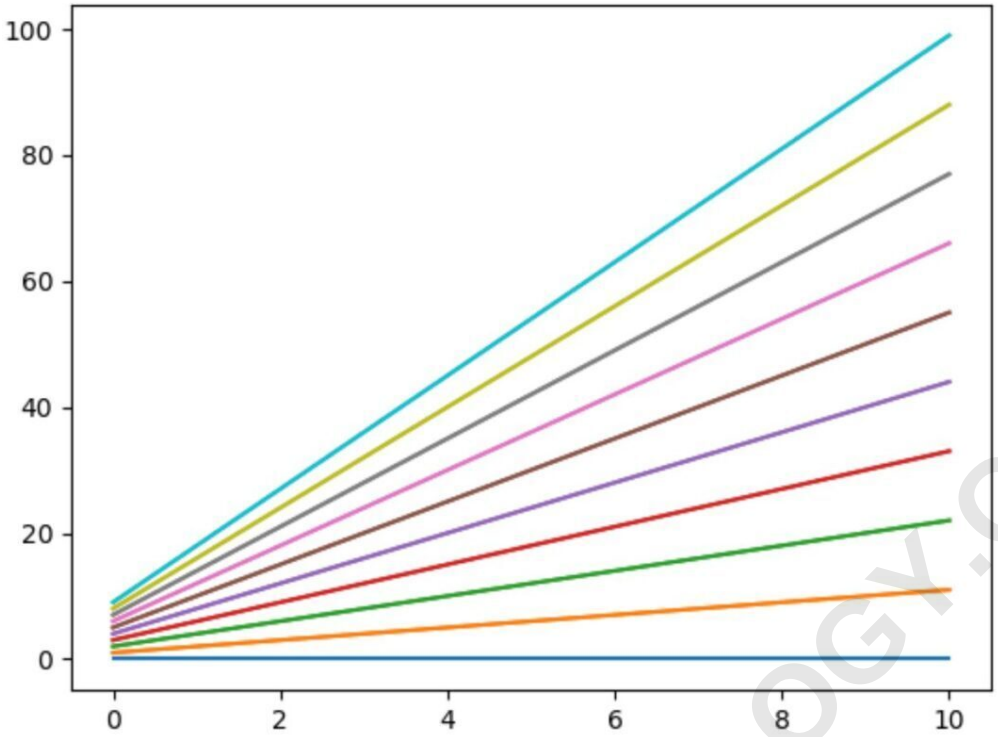
```
import numpy as np
import matplotlib.pyplot as plt

#define plot
fig = plt.figure()
ax = fig.add_subplot(111)

#define range
j = np.arange(11)

#add lines to plot
for i in range(10):
line, = ax.plot(j,i*(j+1))
ax.plot(j,i*(j+1), color = line.get_color())

#display plot
plt.show()
```



As demonstrated by the visual output, Matplotlib effectively uses the first ten colors in its cycle for the lines in the plot, ensuring maximal contrast between the adjacent lines. The resulting visualization clearly separates the data series, fulfilling the core objective of the default color scheme in multi-series data presentation and confirming the automatic assignment behavior inherent in the library's design.

Retrieving and Analyzing Hex Codes

While the visual representation is helpful, analysts often require the exact color specifications for documentation, reporting, or for ensuring consistency across different plotting environments or software packages. Matplotlib stores these precise color definitions as hexadecimal color codes. We can directly query the configuration parameters to extract this list programmatically, giving us the definitive set of default colors that comprise the 'tab10' palette.

The list of default colors is stored within the configuration dictionary, specifically under the `axes.prop_cycle` entry in rcParams. This parameter manages all aesthetic properties that cycle through plots. By accessing this property and calling the `by_key()` method, we can isolate and print the array containing the Hex codes for the default color sequence. This provides the ground truth for Matplotlib's default styling.

The following Python script demonstrates how to retrieve and display the actual hex color codes corresponding to the ten lines shown in the previous figure. This technique is invaluable for users

who need to confirm or manually implement these specific color values in external applications, ensuring that color identity is preserved regardless of the plotting context.

```
import matplotlib.pyplot as plt

#display hex color codes
print(plt.rcParams.by_key())
```

The output above provides the definitive sequence of ten hexadecimal strings. These codes represent the exact color values assigned automatically by Matplotlib when iterating through the default color cycle. Understanding how to access and interpret this list empowers the user to work with Matplotlib's internals more effectively, especially when attempting to debug color assignments or create highly reproducible plots across different environments.

Interpreting the Default Color Order

The sequence of Hex codes is critical because it dictates the aesthetic progression of elements in any plot where colors are not manually specified. The order is determined by the 'tab10' palette, which is optimized for maximum distinction among the first few elements. This careful arrangement ensures that the most important comparisons, often between the first two or three series, are immediately apparent to the viewer without requiring a detailed legend reference.

Let's examine the first few entries in this critical list and their corresponding visual appearance:

The first hex color code is **#1f77b4**. This color corresponds to a strong, reliable blue, and it will always be the color of the very first data series plotted on new axes, provided no color argument is specified. This blue is widely recognized and provides a stable starting point for any visualization.
The second hex color code is **#ff7f0e**. This represents a vibrant orange hue, chosen specifically for its high contrast against the initial blue. If a plot contains two series, the second series will automatically adopt this orange, providing excellent visual separation crucial for comparing two key metrics.
The third hex color code is **#2ca02c**. This corresponds to a deep, stabilizing green. This green continues the pattern of high contrast, ensuring that even with three series simultaneously displayed, visual separation is maintained effortlessly, reducing the cognitive load on the viewer.
The remaining colors follow a similar principle, cycling through distinct reds (**#d62728**), purples (**#9467bd**), and browns (**#8c564b**), before utilizing slightly more muted tones like gray (**#7f7f7f**), guaranteeing ten unique and reasonably high-contrast options before any repetition occurs.

This sequential assignment means that if you create a plot in Matplotlib with one line, the color of the line will be **#1f77b4** unless you specify otherwise. Likewise, if you create a plot with two lines,

the color of the first line will be **#1f77b4** and the color of the second line will be **#ff7f0e**. This deterministic behavior is a core characteristic of Matplotlib's default styling, guaranteeing predictable results and consistent visual language across different analyses.

Implications for Multi-Series Plotting

The primary utility of the default Color Cycle becomes evident when dealing with visualizations involving multiple data sets, such as comparing five different stocks over time or analyzing ten different experimental groups. Instead of manually defining a list of colors and iterating through them--a tedious and error-prone process--Matplotlib manages the assignment internally, simplifying the user workflow considerably.

Consider the practical scenario of plotting multiple lines or bars: if you generate a plot with five distinct lines, Matplotlib will automatically assign the first five colors from the sequence (Blue, Orange, Green, Red, Purple, etc.). If you plot eleven lines, the cycle will automatically repeat, starting again with the blue (**#1f77b4**) for the eleventh line. While the repetition is necessary when exceeding the cycle length, users should be mindful that relying on the cycle for highly complex plots (more than ten series) can lead to potential visual confusion, necessitating manual intervention or the use of larger, specialized colormaps.

The deterministic nature of the color assignment is one of Matplotlib's greatest strengths for rapid prototyping and routine analysis. It ensures that the visual hierarchy remains stable. For instance, the first data set introduced to the plot (often representing the control or baseline) always receives the prominent blue color, reinforcing its position as the initial reference point. Leveraging this consistency improves both development speed and the reliability of analytical visualizations in Matplotlib environments, particularly when integrating these plots into larger automated reporting pipelines.

Customizing the Color Cycle via rcParams

While the default 'tab10' palette is optimized for general use, there are many scenarios where customization is necessary, perhaps to align with corporate branding, achieve specific accessibility requirements, or utilize a different perceptually uniform palette (like 'viridis' or 'plasma'). Matplotlib provides robust control over this through the **runtime configuration parameters**, or rcParams.

To change the default color cycle, one modifies the `axes.prop_cycle` setting. This setting accepts a cycler object, which can define cycles for various properties, including `color`, `linestyle`, and `marker`. For example, a user might choose to use a built-in Matplotlib colormap and extract colors from it, or define an entirely custom list of Hex codes or standard color names to suit their specific aesthetic needs.

For instance, switching the default cycle to a palette of five specific colors requires setting the `axes.prop_cycle` using specialized code involving the `cycler` module. After executing such a command, all subsequent plots in that session will utilize this new, five-color cycle, demonstrating the immense flexibility that rcParams offers. This ability to globally or locally override defaults is what makes Matplotlib so powerful for professional-grade data visualization, allowing engineers to create custom styling themes easily.

Further Resources for Matplotlib Mastery

Mastering Matplotlib extends beyond merely knowing the default colors; it involves understanding how to manage axes, apply intricate styling, and integrate data effectively. The default color cycle is a foundational concept, but true expertise requires exploring related functionalities like custom colormaps, line properties, and sophisticated legend generation. For users seeking to deepen their Matplotlib knowledge, consulting the official documentation for advanced tutorials is highly recommended.

A deep dive into the official documentation will provide a complete explanation of the default colors and the extensive options available for property cycling and styling, including accessibility considerations related to color choice. Understanding how to manage the visual attributes of your data is paramount for generating publication-quality figures that communicate complex findings accurately and efficiently.

The following tutorials explain how to perform other common tasks in Matplotlib, providing necessary context for advanced plotting techniques and full mastery of the visualization library: