A brief introduction to Python

Samuel Repka, Joona Kareinen

January 3, 2025

1 Python in a nutshell

Python is a versatile, high-level programming language widely used in academia, industry, and research. Its simplicity and readability make it an excellent choice for both beginners and experienced developers. Python is popular for several key reasons:

- **Open-Source and Free**: Python is free to use and open-source, making it accessible to everyone, including students and researchers on tight budgets.
- **Extensive Libraries**: Python has a rich ecosystem of libraries like NumPy, SciPy, and Matplotlib for scientific computing, Pandas for data manipulation, and TensorFlow, JAX, and PyTorch for machine learning.
- **General-Purpose**: Python isn't limited to scientific computing. It's also used for web development, automation, data analysis, and much more.
- **Community Support**: Python has a large, active community, offering extensive tutorials, forums, and documentation to help you.

1.1 Tips for Getting the Most Out of Python

To work efficiently with Python, here are a few recommendations:

- Learn to use **jupyter notebooks**: For interactive coding, data visualization, and documentation, Jupyter Notebooks (.ipynb files) are an excellent choice. [**Short tutorial**]
- IPython for Interactive Sessions: IPython provides an enhanced interactive shell for executing Python code, making it easier to test small code snippets and experiment with your ideas in real-time.

1.2 Installing Python

You can follow this guide. Install some reasonably recent version (>=3.9).

1.2.1 Virtual environment

It's **highly** recommended to use a virtual environment. This allows you to manage different versions of packages for each project. To create a virtual environment with the built-in venv module, follow this tutorial.

1.2.2 Integrated Development Environments (IDE)

For a more robust development experience use an IDE. Some popular ones are VS code and PyCharm. Both are really good, VSCode is free and you can acquire a free student license for PyCharm. I recommend going for VSCode, as this is what I use so I can help you better if any problems arise.

If using VSCode, after installing it you should also install `Python` and `Jupyter` extensions from Microsoft.

1.3 Useful Resources for Python

Here are some good resources to help you:

- **Official Python Documentation**: The official documentation is an excellent starting point, though it can be quite detailed. It's great for in-depth information on Python's features.
- **Python for Data Science Handbook**: A comprehensive guide by Jake VanderPlas for using Python in data science, covering libraries like NumPy, Pandas, Matplotlib, and Scikit-Learn.
- **Internet**: When you encounter issues, don't forget to Google for solutions. The Python community is vast, and you'll often find answers to common problems online.
- **NumPy for MATLAB user**: An official guide to help MATLAB users transition to NumPy.

1.4 Basics - A quick cheatsheet

[18]:
```python
# Variable Assignment
x = 5          # Integer
y = 3.14       # Float
name = "Alice" # String
is_valid = True # Boolean

# Lists (like arrays in MATLAB but more flexible)
my_list = [1, 2, 3, 4]       # Create a list
my_list.append(5)            # Add an element to the end
my_list[0]                   # Access first element (0-indexed)
my_list[-1]                  # Access last element
my_list[1:3]                 # Slice: elements 1 to 2 (not including 3)
my_list.reverse()            # Reverse the list
len(my_list)                 # Length of the list

# List Comprehensions (concise ways to create lists)
squares = [x**2 for x in range(5)] # [0, 1, 4, 9, 16]

# Dictionaries (key-value pairs)
my_dict = {'name': 'Alice', 'age': 25} # Create a dictionary
my_dict['name']                        # Access value by key
```

```
my_dict['city'] = 'New York'           # Add a new key-value pair
del my_dict['age']                     # Remove a key-value pair
my_dict.keys()                         # Get all keys
my_dict.values()                       # Get all values



# Loops
for i in range(5):         # Loop through a range of numbers (0 to 4)
    print("value of i:", i)

for item in my_list:       # Loop through a list
    print(f"value of i: {item}") # Using f allows you to add variables to␣
 ↪strings with {}

for key, value in my_dict.items(): # Loop through dictionary items
    print(key, value)

# Conditional Statements
if x > 3:
    print("x is greater than 3")
elif x == 3:
    print("x is 3")
else:
    print("x is less than 3")

# Functions
def greet(name):
    return f"Hello, {name}!"

print(greet("Bob")) # Call the function with "Bob"

# String Manipulations
text = "Hello, World!"
text.lower()       # Convert to lowercase
text.upper()       # Convert to uppercase
text.split(", ")   # Split into list by ", "
text.replace("World", "Python") # Replace substring
len(text)          # Length of the string
```

```
value of i: 0
value of i: 1
value of i: 2
value of i: 3
value of i: 4
value of i: 5
value of i: 4
value of i: 3
```

```
value of i: 2
value of i: 1
name Alice
city New York
x is greater than 3
Hello, Bob!
```

[18]: 13

1.5 Packages

Python's functionality is extended using packages, which are collections of functions and classes (similar to MATLAB's toolboxes). After installing a package, you must explicitly import it in your code. **NOTE** that many packages exist, some of which can even solve the exercises for you. You are **not permitted** to use such packages, unless otherwise stated. You can always ask on discord if in doubt.

The most commonly used package manager is `pip`. You can install packages using `pip install package_name` in the terminal. You must have your virtual environment active, if you want to install a package there. For example, if you want to install `numpy` you would write `pip install numpy`.

To use them in the code you need to use the `import` statement. For example, usage of installed numpy package, it would look like this:

[19]:
```
import numpy
m = numpy.zeros((3,3))
```

You can also alias them

[20]:
```
import numpy as np
m = np.zeros((3,3))
```

import just a specific function

[21]:
```
from numpy import zeros
m = zeros((3,3))
```

or use a wildcard import to import everything into the current namespace. But **DON'T** do this. There are many reasons for that.

[22]:
```
from numpy import *
m = zeros((3,3))
```

1.5.1 NumPy

NumPy is an incredibly powerful package that provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

Here are some of the most commonly used functions and methods in NumPy:

[23]:
```python
import numpy as np  # Import NumPy library

# Creating Arrays
arr = np.array([1, 2, 3])           # 1D Array (like a MATLAB row vector)
mat = np.array([[1, 2], [3, 4]])    # 2D Array (like a MATLAB matrix)
zeros = np.zeros((3, 3))            # 3x3 array of zeros
ones = np.ones((2, 3))              # 2x3 array of ones
lin = np.linspace(0, 10, 5)         # 5 evenly spaced points between 0 and 10
rng = np.arange(0, 10, 2)           # Array of values from 0 to 10 with step 2

# Array Shape and Reshaping
arr.shape                           # Get the shape of the array (e.g., (3,))
mat.reshape(4, 1)                   # Reshape a 2x2 array to 4x1
mat.T                               # Transpose of a 2D array

# Key Difference: Transposing 1D Arrays in NumPy vs MATLAB
vec = np.array([1, 2, 3])           # 1D array in NumPy
vec.T                               # Transpose does nothing for a 1D array!␣
 ↪Shape remains (3,)
vec_2d = vec.reshape(-1, 1)         # Convert to a column vector (3, 1)
vec_row = vec.reshape(1, -1)        # Convert to a row vector (1, 3)

# Equivalent MATLAB Code:
# MATLAB automatically handles 1D vectors as row or column vectors:
# vec = [1, 2, 3];          % Row vector
# vec_T = vec';             % Transposed to a column vector

# Element-wise Operations
arr = np.array([1, 2, 3])
arr + 2                             # Add 2 to each element: [3, 4, 5]
arr * 3                             # Multiply each element by 3: [3, 6, 9]
arr ** 2                            # Square each element: [1, 4, 9]
arr / 2                             # Divide each element by 2: [0.5, 1.0, 1.5]

# Matrix Operations
mat1 = np.array([[1, 2], [3, 4]])
mat2 = np.array([[5, 6], [7, 8]])
mat_add = mat1 + mat2               # Matrix addition
mat_mult = mat1 @ mat2              # Matrix multiplication (use @ or np.dot)
elementwise_mult = mat1 * mat2      # Element-wise multiplication

# Broadcasting (automatic expansion of dimensions for operations)
a = np.array([1, 2, 3])
b = np.array([[1], [2], [3]])
result = a + b                      # Broadcasts to add each row of `b` to `a`

# Mathematical Functions
```

```
np.sin(arr)                          # Apply sine function element-wise
np.exp(arr)                          # Exponential (e^x) element-wise
np.sqrt(arr)                         # Square root element-wise
np.sum(mat)                          # Sum all elements in the matrix
np.mean(mat)                         # Mean of all elements
np.max(mat)                          # Maximum value
np.min(mat)                          # Minimum value

# Array Methods (Alternative to np functions)
mat.sum()                            # Sum all elements (same as np.sum(mat))
mat.mean()                           # Mean of all elements (same as np.mean(mat))
mat.max()                            # Maximum value (same as np.max(mat))
mat.min()                            # Minimum value (same as np.min(mat))
mat.prod()                           # Product of all elements

# Indexing and Slicing
arr = np.array([10, 20, 30, 40])
arr[0]                               # Access the first element: 10
arr[-1]                              # Access the last element: 40
arr[1:3]                             # Slice from index 1 to 2: [20, 30]

# Boolean Indexing
arr[arr > 20]                        # Get elements greater than 20: [30, 40]
arr[arr % 2 == 0]                    # Get even elements: [10, 20, 30, 40]

# Combining and Splitting Arrays
arr1 = np.array([1, 2])
arr2 = np.array([3, 4])
np.concatenate((arr1, arr2))         # Combine arrays: [1, 2, 3, 4]
np.stack((arr1, arr2), axis=0)       # Stack vertically: [[1, 2], [3, 4]]
np.stack((arr1, arr2), axis=1)       # Stack horizontally: [[1, 3], [2, 4]]

# Useful NumPy Functions
np.eye(3)                            # Identity matrix of size 3x3
np.linalg.inv(mat1)                  # Inverse of a matrix
np.linalg.eig(mat1)                  # Eigenvalues and eigenvectors
np.random.rand(3, 3)                 # Generate a 3x3 matrix of random numbers␣
 ↪between 0 and 1
np.random.randint(0, 10, (2, 2))     # Generate random integers between 0 and 10␣
 ↪in a 2x2 matrix

# random numpy matrix
print(np.random.rand(3,3))
```

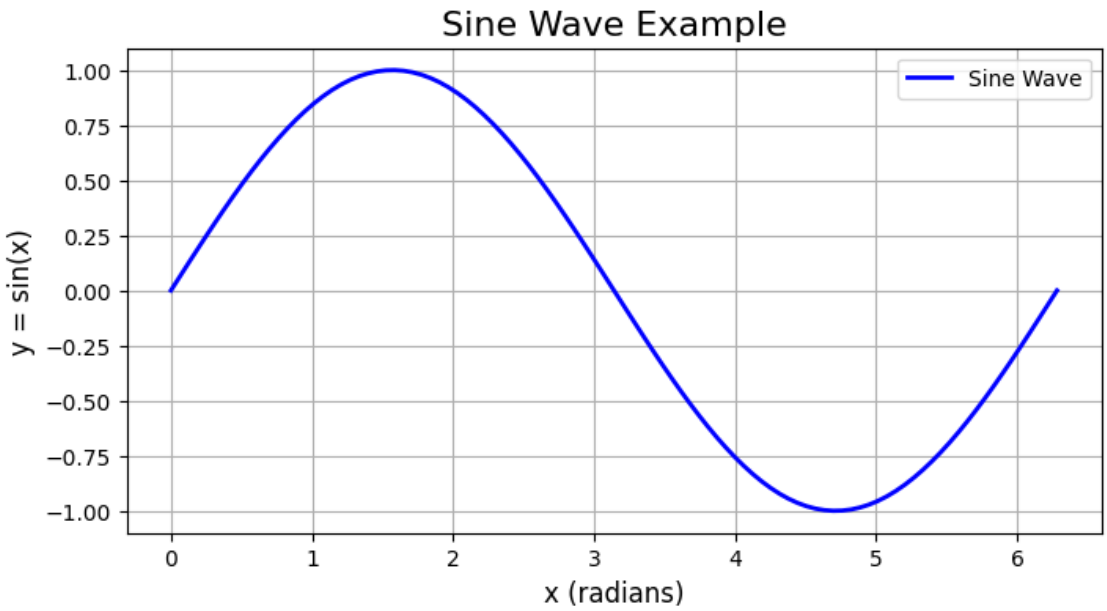
```
[[0.75631139 0.883283   0.00278173]
 [0.20524394 0.87087055 0.20178081]
 [0.15899667 0.31059146 0.91877901]]
```

1.5.2 Matplotlib - A plotting library

Matplotlib is a powerful and versatile Python library used for creating static, interactive, and animated visualizations. It is especially popular for scientific and engineering plots, and it is a go-to library for data visualization in Python. [Plot types]

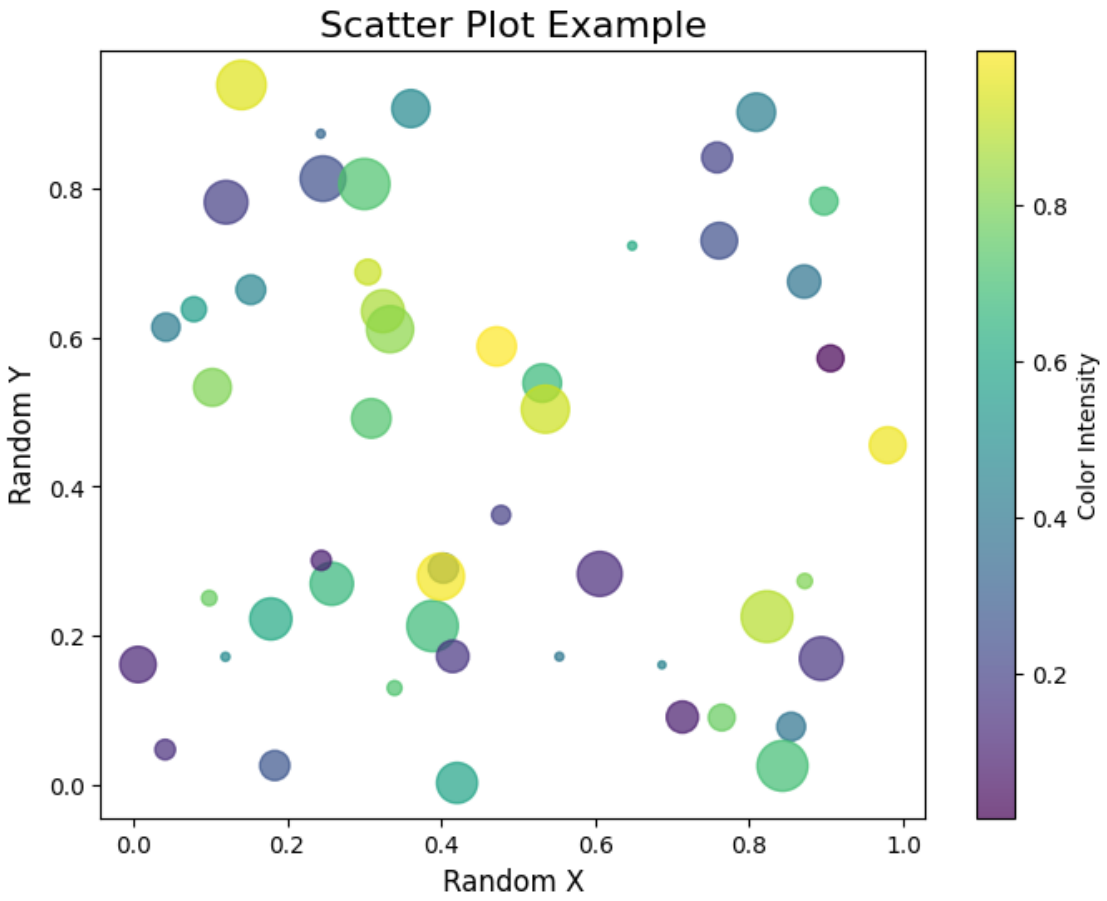
You can use it for visualizing 3D data, but it is not GPU accelerated, so you may run into some performance problems. A 3D plotting library will be introduced further.

[24]:
```
import matplotlib.pyplot as plt

# Line Plot
x = np.linspace(0, 2 * np.pi, 100)  # Generate 100 points between 0 and 2
y = np.sin(x)

plt.figure(figsize=(8, 4))          # Create a figure with a custom size
plt.plot(x, y, color='blue', label='Sine Wave', linewidth=2)  # Plot y = sin(x)
plt.title("Sine Wave Example", fontsize=16)                   # Add a title
plt.xlabel("x (radians)", fontsize=12)                        # Label x-axis
plt.ylabel("y = sin(x)", fontsize=12)                         # Label y-axis
plt.grid(True)                                                # Add a grid
plt.legend()                                                  # Show the legend
plt.show()                                                    # Display the plot
```



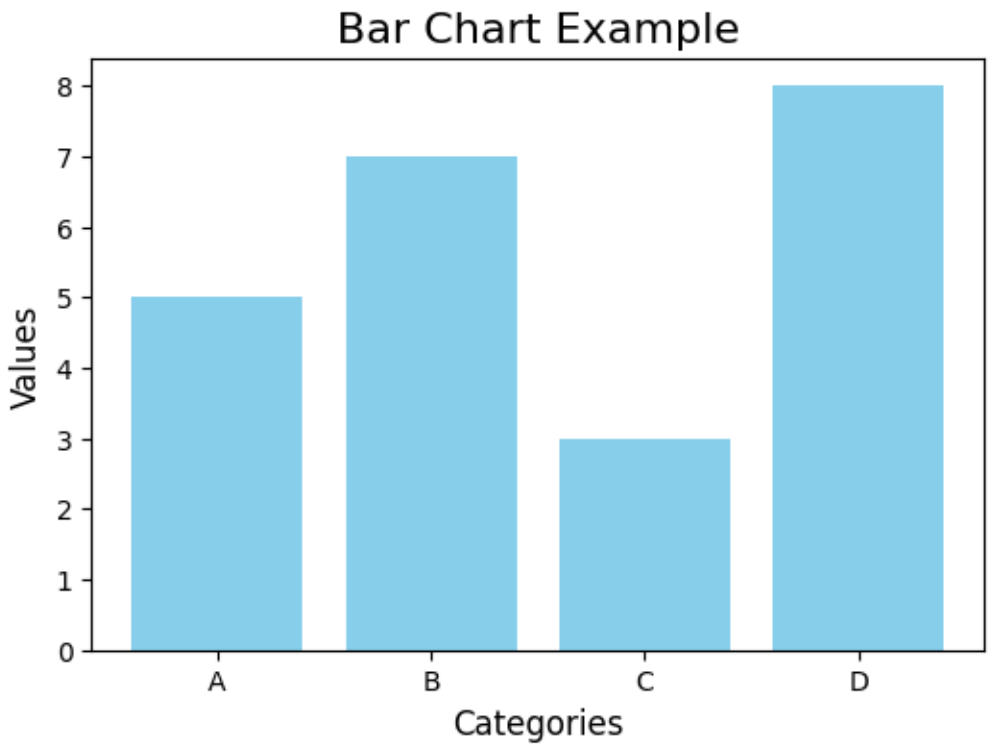
[25]:
```
# Scatter Plot
x = np.random.rand(50)          # 50 random x-coordinates
y = np.random.rand(50)          # 50 random y-coordinates
colors = np.random.rand(50)     # Random colors for each point
```

```
sizes = np.random.rand(50) * 500  # Random sizes for each point

plt.figure(figsize=(8, 6))
plt.scatter(x, y, c=colors, s=sizes, alpha=0.7, cmap='viridis')  # Create␣
 ↪scatter plot
plt.colorbar(label='Color Intensity')                            # Add color bar
plt.title("Scatter Plot Example", fontsize=16)                   # Add title
plt.xlabel("Random X", fontsize=12)                              # Label x-axis
plt.ylabel("Random Y", fontsize=12)                              # Label y-axis
plt.show()                                                       # Display the␣
 ↪plot
```



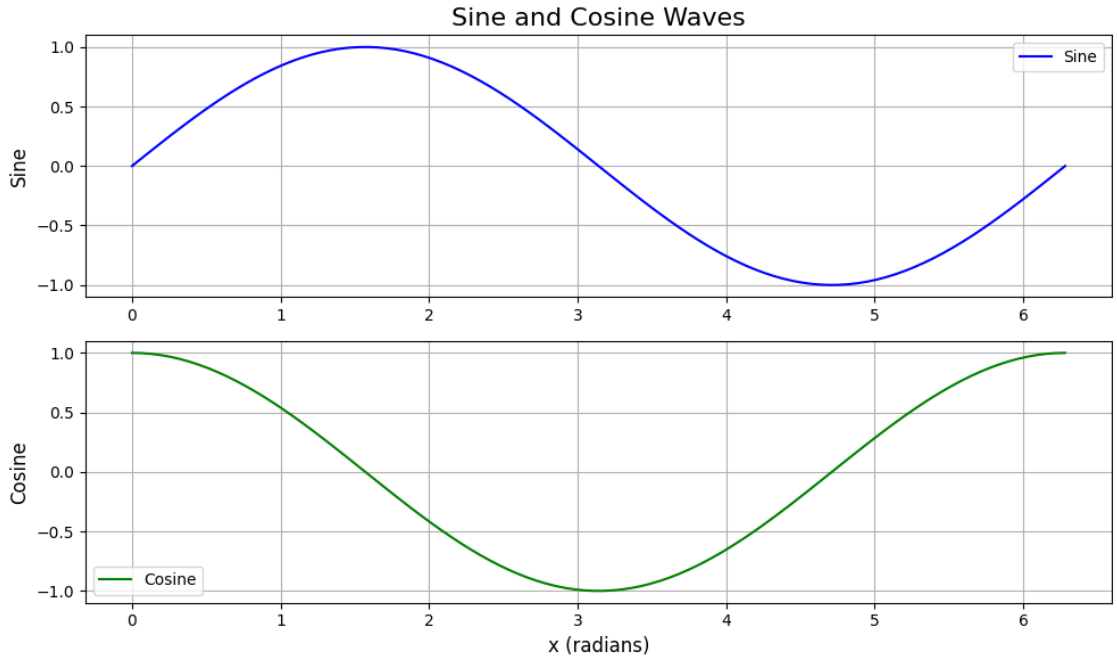
[26]:
```
# Bar Chart
categories = ['A', 'B', 'C', 'D']
values = [5, 7, 3, 8]

plt.figure(figsize=(6, 4))
plt.bar(categories, values, color='skyblue')  # Create bar chart
```

```
plt.title("Bar Chart Example", fontsize=16)    # Add title
plt.xlabel("Categories", fontsize=12)          # Label x-axis
plt.ylabel("Values", fontsize=12)              # Label y-axis
plt.show()
```

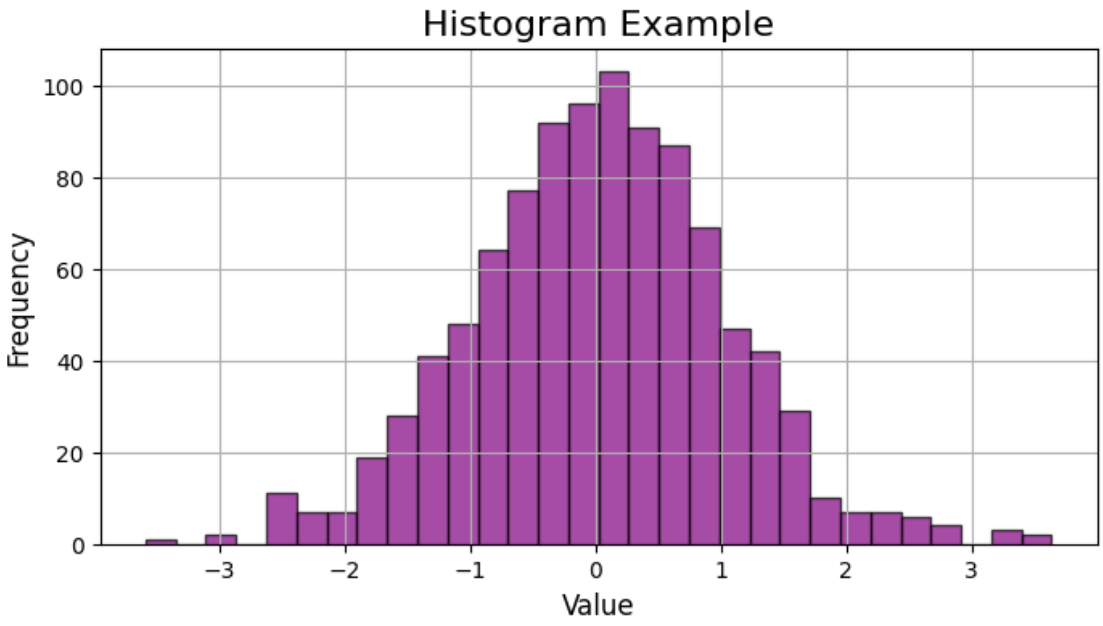


[27]:
```
# Subplots
x = np.linspace(0, 2 * np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)

plt.figure(figsize=(10, 6))

# First subplot: Sine wave
plt.subplot(2, 1, 1)  # (rows, columns, index)
plt.plot(x, y1, label="Sine", color='blue')
plt.title("Sine and Cosine Waves", fontsize=16)
plt.ylabel("Sine", fontsize=12)
plt.grid(True)
plt.legend()

# Second subplot: Cosine wave
plt.subplot(2, 1, 2)  # (rows, columns, index)
```

```
plt.plot(x, y2, label="Cosine", color='green')
plt.xlabel("x (radians)", fontsize=12)
plt.ylabel("Cosine", fontsize=12)
plt.grid(True)
plt.legend()

plt.tight_layout()  # Adjust spacing to prevent overlap
plt.show()
```

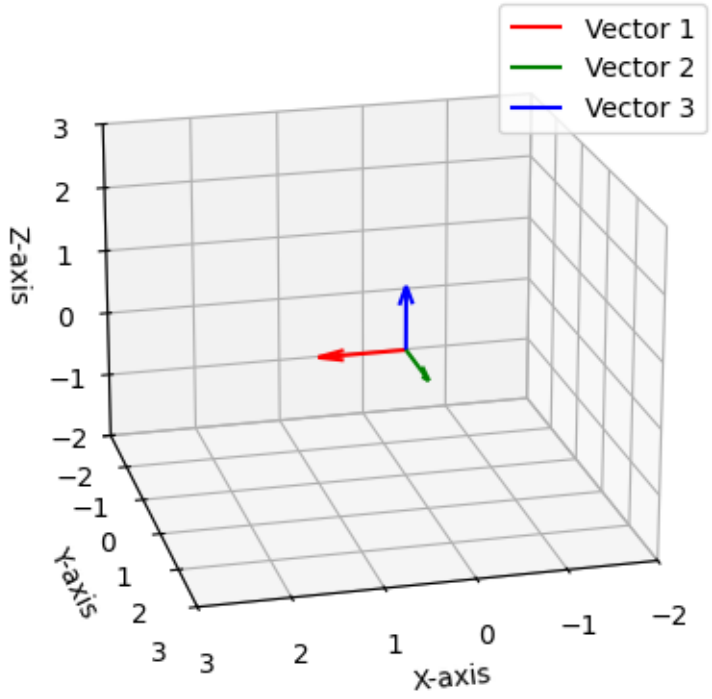


[28]:
```
# Histogram
data = np.random.randn(1000)  # Generate 1000 random numbers from a normal␣
 ↪distribution

plt.figure(figsize=(8, 4))
plt.hist(data, bins=30, color='purple', edgecolor='black', alpha=0.7)  # Create␣
 ↪histogram
plt.title("Histogram Example", fontsize=16)                            # Add␣
 ↪title
plt.xlabel("Value", fontsize=12)                                       # Label␣
 ↪x-axis
plt.ylabel("Frequency", fontsize=12)                                   # Label␣
 ↪y-axis
plt.grid(True)
plt.show()
```



[29]:
```python
# 3D plot
from mpl_toolkits.mplot3d import Axes3D #!important


# Define the starting point (origin)
origin = np.array([0, 0, 0])

# Define the vectors
vectors = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])  # Vectors (arrows)


# Create a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Plot the vectors using quiver
ax.quiver(
    origin[0], origin[1], origin[2],   # Starting point (x, y, z)
    vectors[0][0], vectors[0][1], vectors[0][2], color='r', label='Vector 1'
)
ax.quiver(
    origin[0], origin[1], origin[2],
    vectors[1][0], vectors[1][1], vectors[1][2], color='g', label='Vector 2'
)
ax.quiver(
    origin[0], origin[1], origin[2],
```

```
    vectors[2][0], vectors[2][1], vectors[2][2], color='b', label='Vector 3'
)

# Set labels and limits
ax.set_xlim([-2, 3])
ax.set_ylim([-2, 3])
ax.set_zlim([-2, 3])
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')

# Add a legend
ax.legend()
ax.view_init(elev=20, azim=76)  # Set elevation (up-down) and azimuth (rotation)

# Show the plot
plt.show()
```

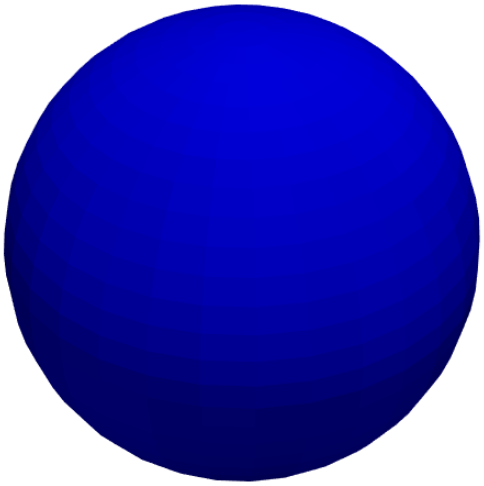


1.5.3 PyVista: Simplified 3D Visualization

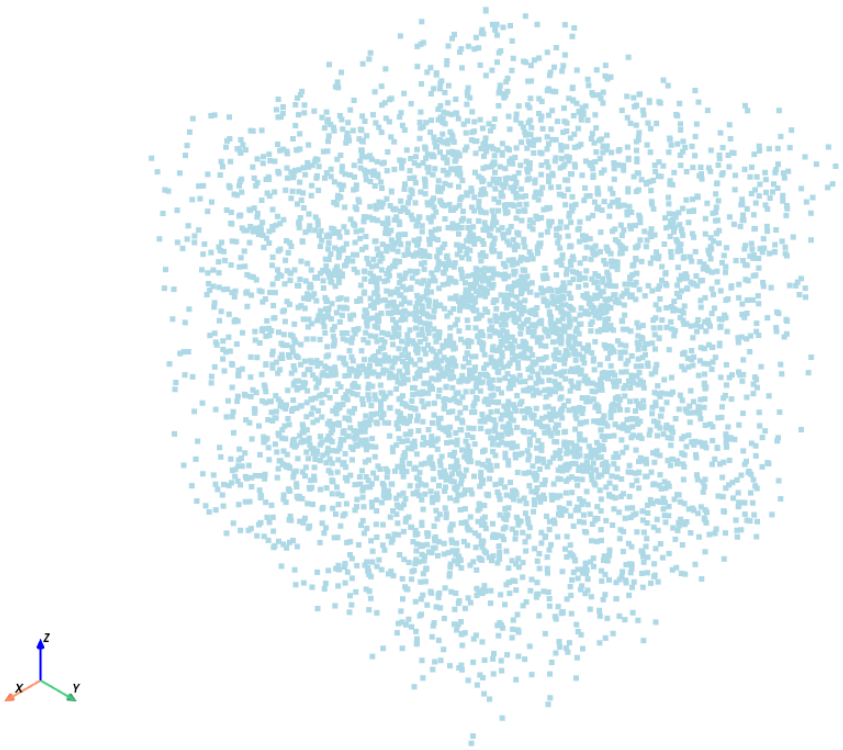
PyVista is an open-source Python library designed to make 3D visualization and mesh processing more accessible and intuitive. Built on top of VTK (Visualization Toolkit), it provides a high-level, user-friendly interface for scientific visualization, spatial data analysis, and 3D graphics. [More

examples]

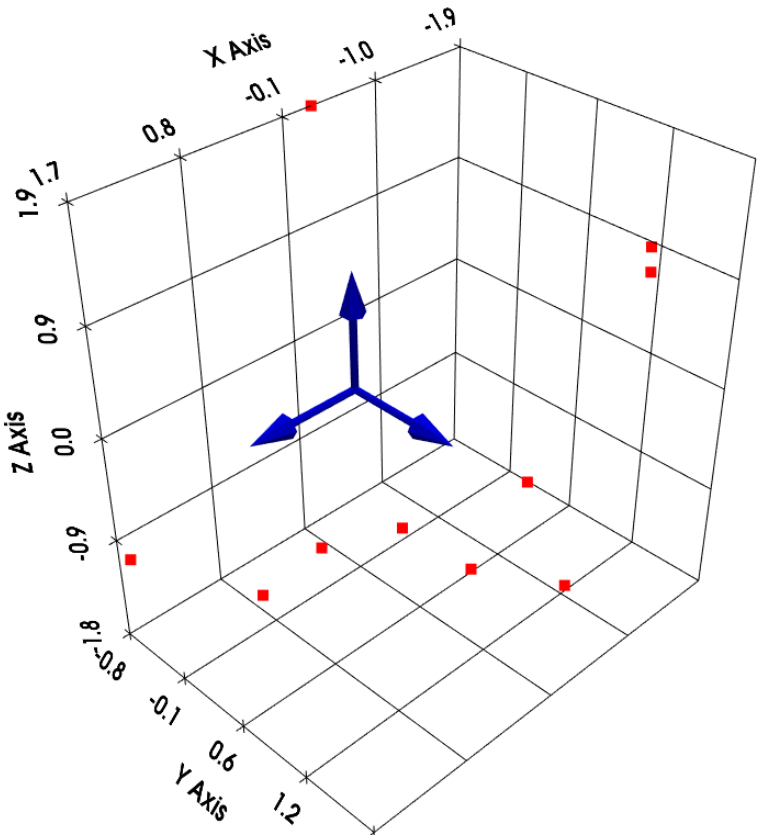
[ ]:
```
import pyvista as pv
# print(pv.Report(gpu=False))
pv.set_jupyter_backend('static')

# Create and plot a simple colored sphere
sphere = pv.Sphere(radius=1, center=(0, 0, 0))
sphere.plot(color='blue', point_size=10)
```



[31]:
```
# Generate and plot a random 3D point cloud
# Generate random points
points = np.random.random((5000, 3)) * 2

# Create point cloud
cloud = pv.PolyData(points)

# Plot with colored points based on height
cloud.plot(style='points', point_size=5, cmap='jet')
```

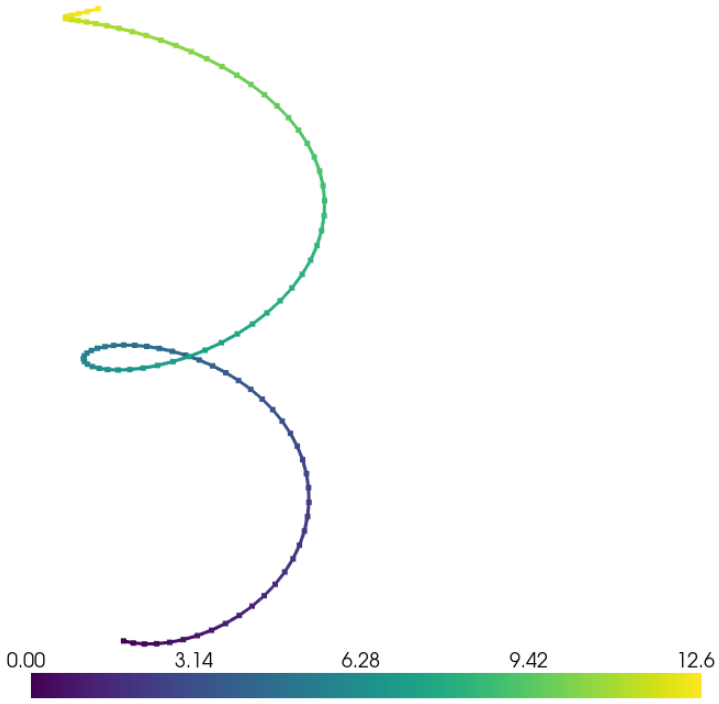


[32]:
```
# Create points in a zigzag pattern

# Create PolyData with points and lines
start_points = np.zeros((3,3))
vectors = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])  # Vectors (arrows)

# Create a PyVista plotter
plotter = pv.Plotter()

# Add arrows to the plot
plotter.add_arrows(start_points, vectors, color="blue")

points = np.random.uniform(-2, 2, (10, 3))  # 10 points in 3D space
plotter.add_points(points, color="red", point_size=10)

plotter.show_grid()
# Show the plot
plotter.show()
```



[33]:
```python
# Generate points in a 3D spiral
t = np.linspace(0, 4*np.pi, 100)
points = np.column_stack([
    np.cos(t),      # x
    np.sin(t),      # y
    t/2             # z
])

# Create PolyData with points and lines
path = pv.PolyData(points)
path.lines = np.column_stack([
    np.full(len(points)-1, 2),  # Number of points per line
    np.column_stack([range(len(points)-1), range(1, len(points))])
])

# Plot with gradient coloring and thin lines
p = pv.Plotter()
p.add_mesh(path,
           scalars=t,  # Color based on parameter
           cmap='viridis',
```

```
            line_width=3,
            point_size=5)
p.show()
```



1.5.4 OpenCV - Open Computer Vision Library

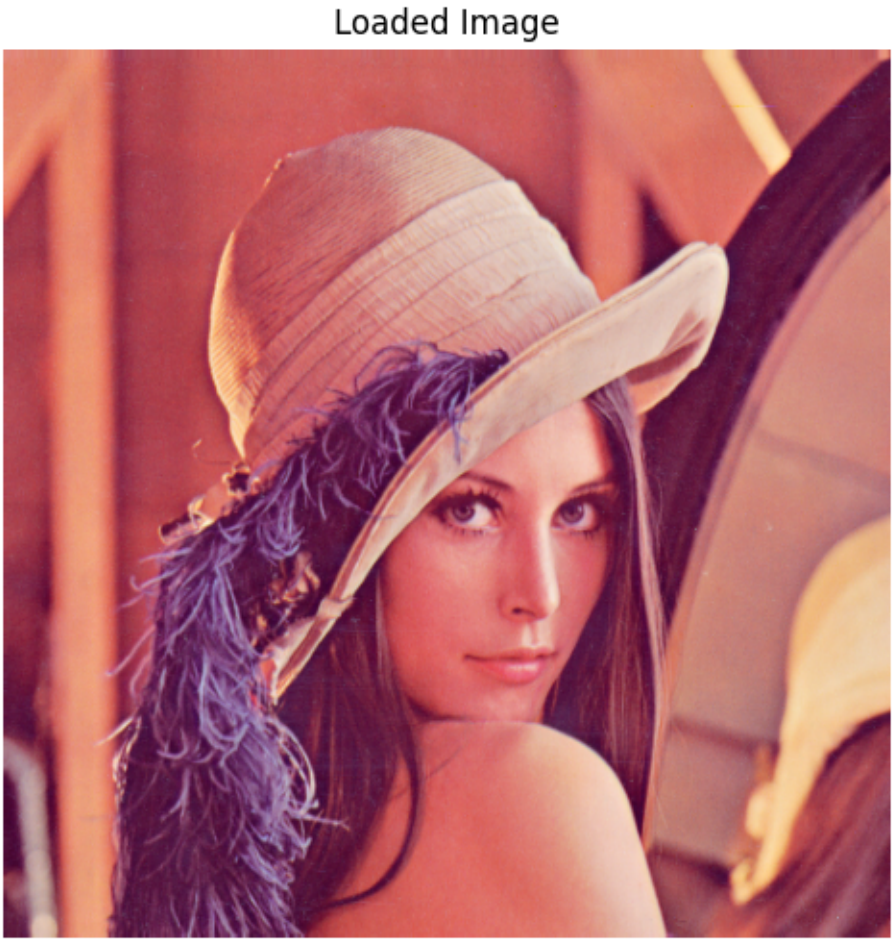
OpenCV is a powerful computer vision and machine learning software library designed to help developers create innovative applications. It provides a wide range of tools for image processing, object detection, machine learning, and more. Unless otherwise stated you are **not allowed** to use other than the basic functionality like reading images, color conversions, resizing, etc.

When loading the color images with OpenCV, the color channels are loaded in the order of Blue, Green, Red (BGR). If you want to convert it to the standard Red, Green, Blue (RGB) order, you can use the `cv2.cvtColor()` function.

The images are loaded as numpy arrays.

[1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

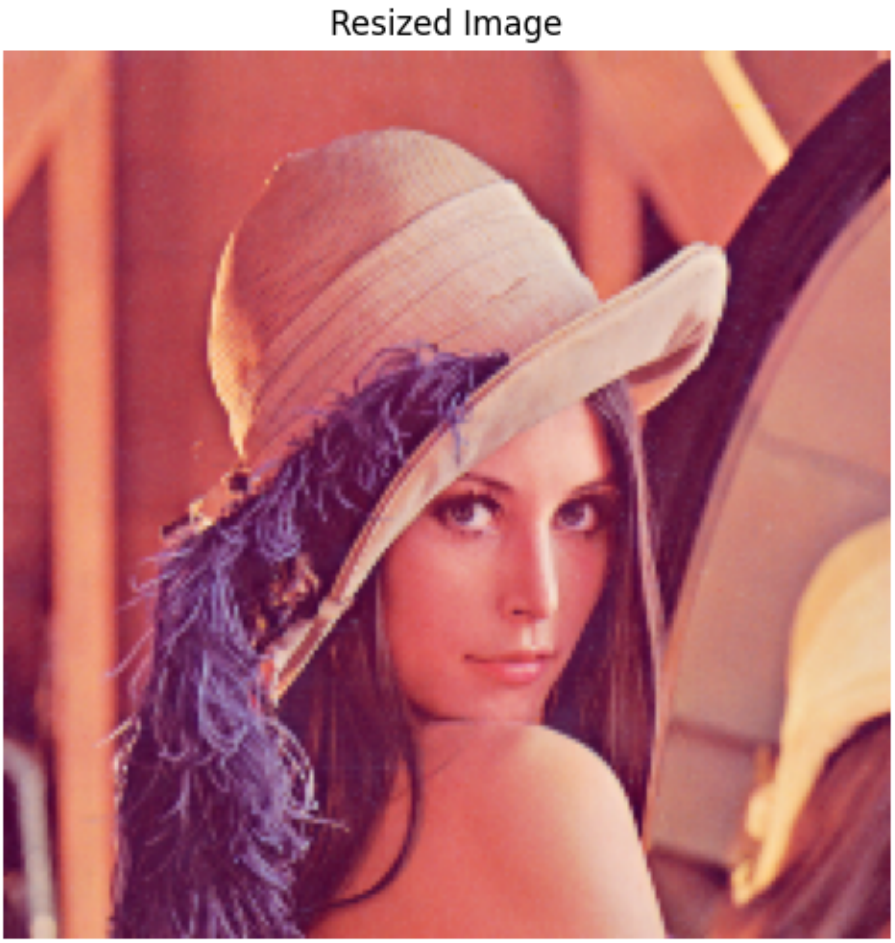
```
# Helper function to display images using Matplotlib
def show_image(img, title="Image", cmap=None):
    plt.figure(figsize=(6, 6))
    plt.imshow(img, cmap=cmap)
    plt.title(title)
    plt.axis("off")
    plt.show()


# 1. Loading an Image
image = cv2.imread("example.png")  # Load an image (default is in BGR format)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert to RGB for␣
 ↪Matplotlib
show_image(image_rgb, "Loaded Image")
```



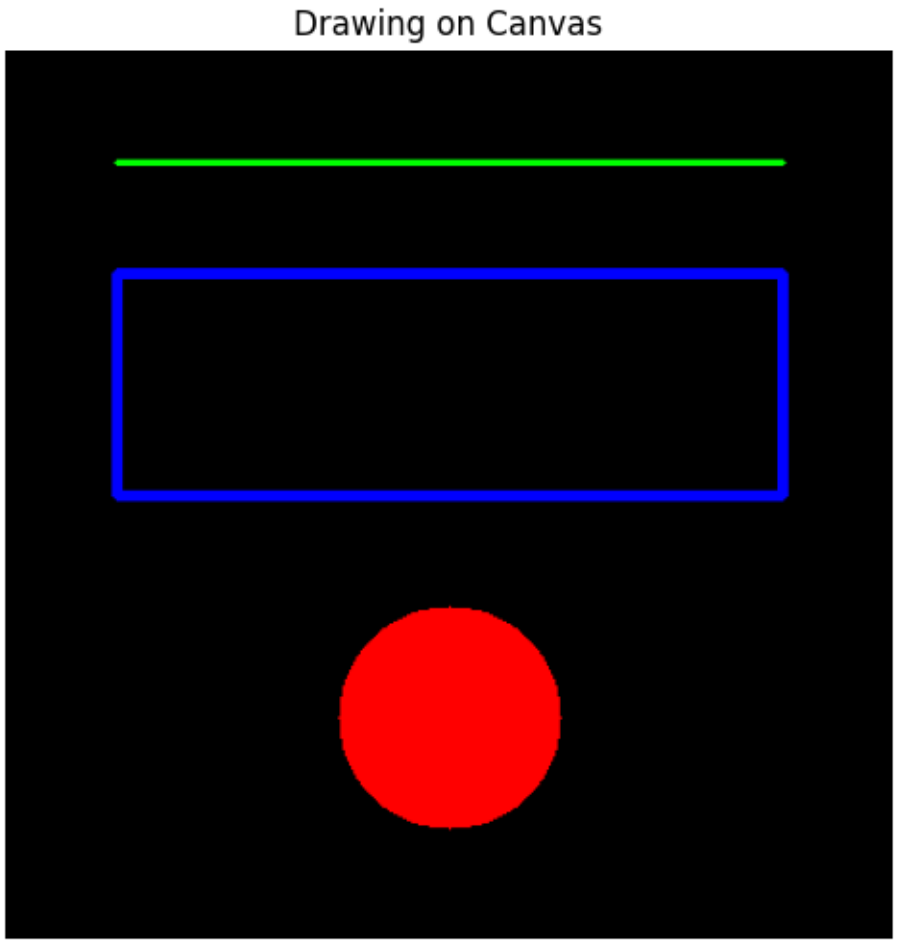
[2]:
```
# 2. Saving an Image
cv2.imwrite("output.png", image)  # Save the image to a file
```

```
# 3. Converting Between Color Spaces
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)  # Convert to HSV
show_image(gray_image, "Grayscale Image", cmap="gray")
```

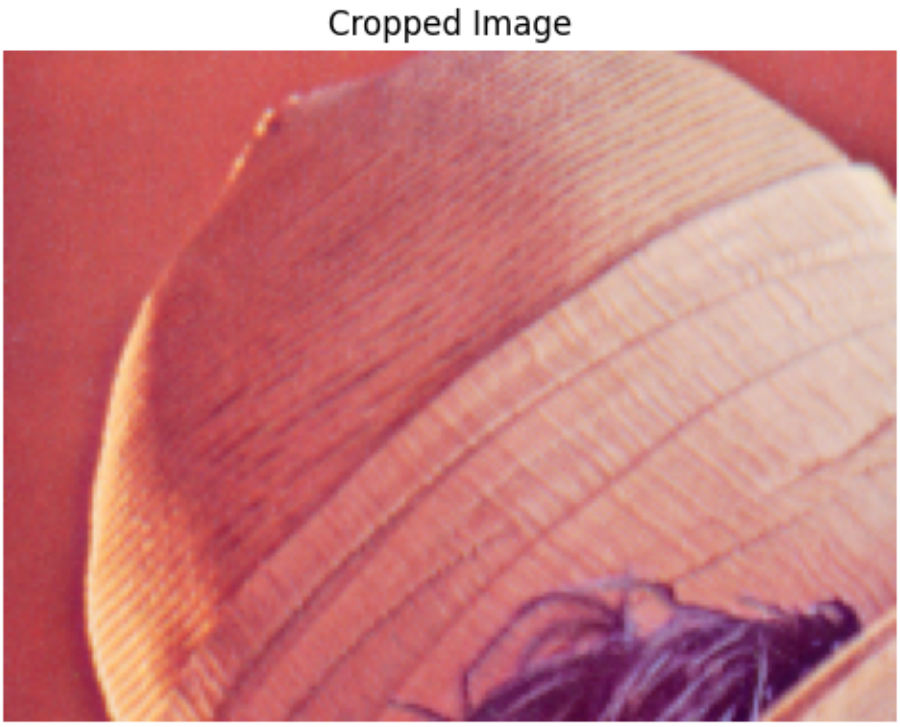
[3]:
```
# 4. Resizing an Image
resized_image = cv2.resize(image_rgb, (200, 200))  # Resize to 200x200 pixels
show_image(resized_image, "Resized Image")
```



[4]:
```
# 5. Drawing on Images
canvas = np.zeros((400, 400, 3), dtype="uint8")  # Create a black canvas

# Draw a line
cv2.line(canvas, (50, 50), (350, 50), (0, 255, 0), 2)  # Green line

# Draw a rectangle
cv2.rectangle(canvas, (50, 100), (350, 200), (255, 0, 0), 3)  # Blue rectangle

# Draw a circle
cv2.circle(canvas, (200, 300), 50, (0, 0, 255), -1)  # Red filled circle

canvas_rgb = cv2.cvtColor(canvas, cv2.COLOR_BGR2RGB)  # Convert to RGB for␣
↪Matplotlib
show_image(canvas_rgb, "Drawing on Canvas")
```



[5]:
```
# 6. Image Cropping
cropped_image = image_rgb[50:200, 100:300]  # Crop a region of interest (y1:y2,␣
↪x1:x2)
show_image(cropped_image, "Cropped Image")
```



[6]:
```
# 7. Adding Text to an Image
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(
    image_rgb, "Hello, OpenCV!", (50, 50), font, 1, (255, 255, 255), 2, cv2.
 ↪LINE_AA
)
show_image(image_rgb, "Image with Text")
```

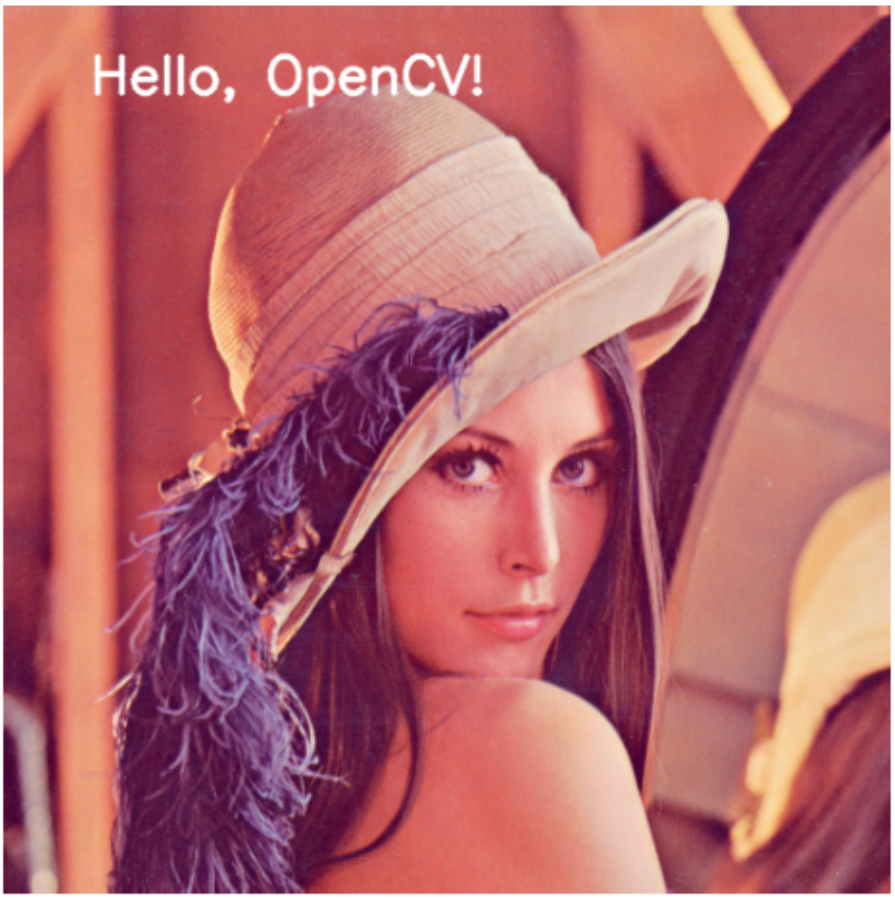


Image with Text
Hello, OpenCV!