# **Problems for List**

# **Example 1: Visualizing a Simple List with Different Data Types**[**¶**](http://localhost:8892/notebooks/Desktop/project/update_last_Python-project/update_List.ipynb#Example-1:-Visualizing-a-Simple-List-with-Different-Data-Types)

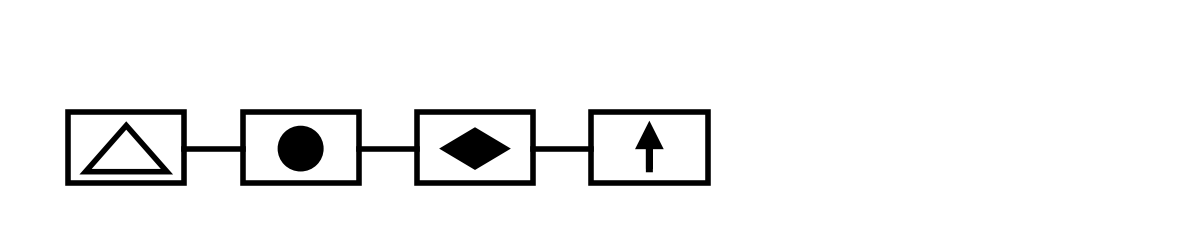
Objective: Understand how different basic data types (integer, string, boolean) are represented in a list.

Problem: Create a list with three elements: an integer, a string, and a boolean value. Visualize this list.

simple\_list = [{1, 2}, "A", 4.5, True]

List(simple\_list)

**Result:**

 **Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Define the coordinates and dimensions of the rectangle

x = 1 # x-coordinate of the lower-left corner

y = 1 # y-coordinate of the lower-left corner

phi = 1.61803398875 # Golden ratio

# Calculate the width and height based on the golden ratio

width = 2

height = width / phi # Height is calculated to maintain the golden ratio

# Create rectangles

rectangle1 = patches.Rectangle((x, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle1)

rectangle2 = patches.Rectangle((x + 3, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle2)

rectangle3 = patches.Rectangle((x + 6, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle3)

rectangle4 = patches.Rectangle((x + 9, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle4)

# Create connected lines between rectangles

y\_l = 1.6

ax.plot([3, 4], [y\_l, y\_l], color='black', linewidth=4)

ax.plot([6, 7], [y\_l, y\_l], color='black', linewidth=4)

ax.plot([9, 10], [y\_l, y\_l], color='black', linewidth=4)

# Define a function to add shapes to the plot

triangle = patches.Polygon(

[(1.3, 1.2), (2, 2), (2.7, 1.2)],

facecolor='none', edgecolor='black', linewidth=4)

ax.add\_patch(triangle)

circle = patches.Circle((5, 1.6), radius=0.3, facecolor='black', edgecolor='black', linewidth=8)

ax.add\_patch(circle)

rhombus = patches.Polygon(

[(8, 1.3), (7.5, 1.6), (8, 1.9), (8.5, 1.6)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus)

line\_length = 0.4

arrow = patches.FancyArrow(11, 1.25, 0, line\_length, head\_width=0.3, head\_length=0.3, fc='black', ec='black', linewidth=5)

ax.add\_patch(arrow)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

plt.show()

# **Example 2: Visualizing a List of Integers**

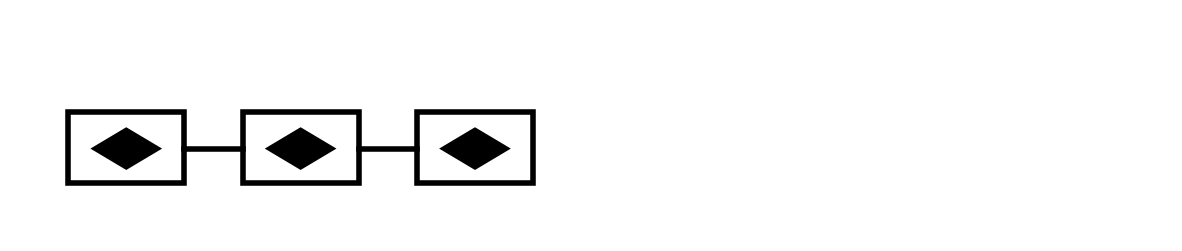
Objective: Learn to visualize a list containing multiple elements of the same type.

Problem: Create a list of integers [1, 2, 3] and visualize it.

integer\_list = [1, 2, 3]

List(integer\_list)

**Result:**

 **Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Define the coordinates and dimensions of the rectangle

x = 1 # x-coordinate of the lower-left corner

y = 1 # y-coordinate of the lower-left corner

phi = 1.61803398875 # Golden ratio

# Calculate the width and height based on the golden ratio

width = 2

height = width / phi # Height is calculated to maintain the golden ratio

# Create rectangles

rectangle1 = patches.Rectangle((x, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle1)

rectangle2 = patches.Rectangle((x + 3, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle2)

rectangle3 = patches.Rectangle((x + 6, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle3)

# Create connected lines between rectangles

y\_l = 1.6

ax.plot([3, 4], [y\_l, y\_l], color='black', linewidth=4)

ax.plot([6, 7], [y\_l, y\_l], color='black', linewidth=4)

# Define a function to add shapes to the plot

rhombus1 = patches.Polygon(

[(2, 1.3), (1.5, 1.6), (2, 1.9), (2.5, 1.6)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus1)

rhombus2 = patches.Polygon(

[(5, 1.3), (4.5, 1.6), (5, 1.9), (5.5, 1.6)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus2)

rhombus3 = patches.Polygon(

[(8, 1.3), (7.5, 1.6), (8, 1.9), (8.5, 1.6)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus3)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

# List of shapes to add

shapes = [1,2,3]

plt.axis("off")

plt.show()

# **Example 3: Visualizing Nested Lists**

Objective: Understand how to represent nested lists.

Problem: Create a list where each element is a nested list with various data types. For example, [[1, "A"], [True, set(), 3.14]].

nested\_list = [[1, "A"], [True, set(), 3.14]]

List(nested\_list)

**Result:**

 **Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Define the coordinates and dimensions of the rectangle

x = 1 # x-coordinate of the lower-left corner

y = 1 # y-coordinate of the lower-left corner

phi = 1.61803398875 # Golden ratio

# Calculate the width and height based on the golden ratio

width = 2

height = width / phi # Height is calculated to maintain the golden ratio

# Create rectangles

rectangle1 = patches.Rectangle((x, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle1)

rectangle2 = patches.Rectangle((x + 3, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle2)

# Create connected lines between rectangles

y\_l = 1.6

ax.plot([3, 4], [y\_l, y\_l], color='black', linewidth=4)

# Define a function to add shapes to the plot

#1-nested list

rectangle1\_1 = patches.Rectangle((x+0.2, y+0.4), width/3, height/3, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle1\_1)

rectangle2\_1 = patches.Rectangle((x+width/3+0.2+0.3, y+0.4), width/3, height/3, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle2\_1)

y\_l = 1.6

ax.plot([x+width/3+0.2, x+width/3+0.2+0.3], [y\_l, y\_l], color='black', linewidth=4)

#2-nested list

rectangle1\_2 = patches.Rectangle((x+width+1+0.2, y+0.4), width/3, height/3, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle1\_2)

rectangle2\_2 = patches.Rectangle((x+width+width/3+1+0.2+0.3, y+0.4), width/3, height/3, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle2\_2)

y\_l = 1.6

ax.plot([x+width+1+0.2+width/3, x+width+1+width/3+0.2+0.3], [y\_l, y\_l], color='black', linewidth=4)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

plt.show()

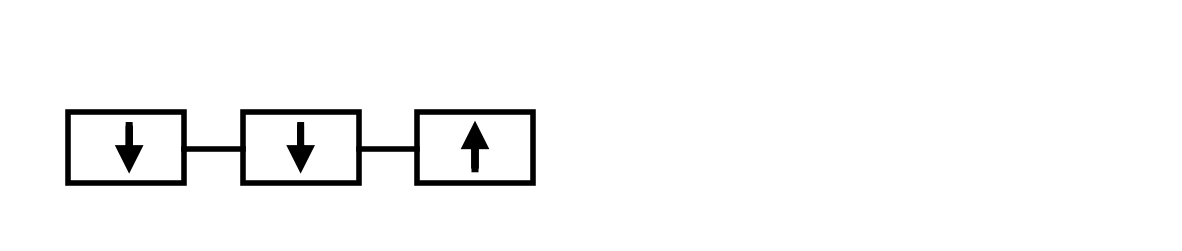
# **Example 4: Visualizing Lists with Conditional Logic**[**¶**](http://localhost:8892/notebooks/Desktop/project/update_last_Python-project/update_List.ipynb#Example-5:-Visualizing-Lists-with-Conditional-Logic)

Objective: Learn to visualize lists that represent the outcomes of conditional logic.

Problem: Represent a list of boolean expressions, like [1 > 2, "A" == "B", 3 <= 3].

conditional\_list = [1 > 2, "A" == "B", 3 <= 3]

List(conditional\_list)

**Result:  
Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Define the coordinates and dimensions of the rectangle

x = 1 # x-coordinate of the lower-left corner

y = 1 # y-coordinate of the lower-left corner

phi = 1.61803398875 # Golden ratio

# Calculate the width and height based on the golden ratio

width = 2

height = width / phi # Height is calculated to maintain the golden ratio

# Create rectangles

rectangle1 = patches.Rectangle((x, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle1)

rectangle2 = patches.Rectangle((x + 3, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle2)

rectangle3 = patches.Rectangle((x + 6, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle3)

# Create connected lines between rectangles

y\_l = 1.6

ax.plot([3, 4], [y\_l, y\_l], color='black', linewidth=4)

ax.plot([6, 7], [y\_l, y\_l], color='black', linewidth=4)

# Define a function to add shapes to the plot

line\_length = 0.4

#1

arrow1 = patches.FancyArrow(2.05, 2, 0, -line\_length, head\_width=0.3, head\_length=0.3, fc='black', ec='black', linewidth=5)

ax.add\_patch(arrow1)

#2

arrow2 = patches.FancyArrow(5, 2, 0, -line\_length, head\_width=0.3, head\_length=0.3, fc='black', ec='black', linewidth=5)

ax.add\_patch(arrow2)

#3

arrow3 = patches.FancyArrow(8, 1.25, 0, line\_length, head\_width=0.3, head\_length=0.3, fc='black', ec='black', linewidth=5)

ax.add\_patch(arrow3)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

plt.show()

# **Example 5: Visualizing append Operation on a List**[**¶**](http://localhost:8892/notebooks/Desktop/project/update_last_Python-project/update_List.ipynb#Example-7:-Visualizing-append-Operation-on-a-List)

Objective: Learn how the append operation adds an element to a specified position in a list (or the last element by default) and visualize this change.

Problem: Start with a list, e.g., [1,”A”], and then append an element to it (e.g., the second element). Visualize the list before and after the append operation.

**# Initial list**

initial\_list = [1, "A"]

List(initial\_list)

**Result:**

 **Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Define the coordinates and dimensions of the rectangle

x = 1 # x-coordinate of the lower-left corner

y = 1 # y-coordinate of the lower-left corner

phi = 1.61803398875 # Golden ratio

# Calculate the width and height based on the golden ratio

width = 2

height = width / phi # Height is calculated to maintain the golden ratio

# Create rectangles

rectangle1 = patches.Rectangle((x, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle1)

rectangle2 = patches.Rectangle((x + 3, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle2)

# Create connected lines between rectangles

y\_l = 1.6

ax.plot([3, 4], [y\_l, y\_l], color='black', linewidth=4)

# Define a function to add shapes to the plot

rhombus1 = patches.Polygon(

[(2, 1.3), (1.5, 1.6), (2, 1.9), (2.5, 1.6)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus1)

circle = patches.Circle((5, 1.6), radius=0.3, facecolor='black', edgecolor='black', linewidth=8)

ax.add\_patch(circle)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

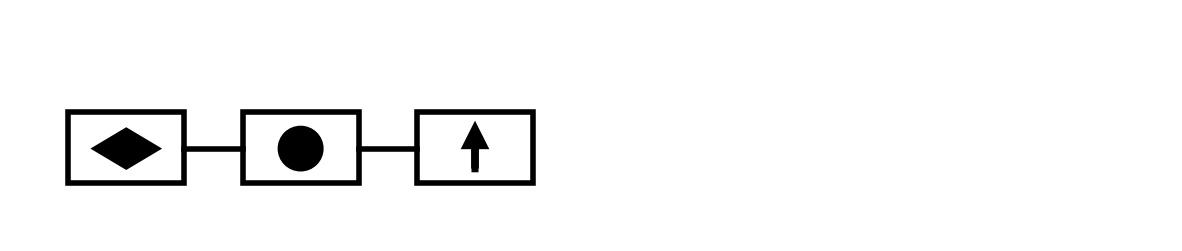
plt.show()

**# Append a boolean value**

initial\_list.append(True)

List(initial\_list)

**Result:**

****

**Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Define the coordinates and dimensions of the rectangle

x = 1 # x-coordinate of the lower-left corner

y = 1 # y-coordinate of the lower-left corner

phi = 1.61803398875 # Golden ratio

# Calculate the width and height based on the golden ratio

width = 2

height = width / phi # Height is calculated to maintain the golden ratio

# Create rectangles

rectangle1 = patches.Rectangle((x, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle1)

rectangle2 = patches.Rectangle((x + 3, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle2)

rectangle3 = patches.Rectangle((x + 6, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle3)

# Create connected lines between rectangles

y\_l = 1.6

ax.plot([3, 4], [y\_l, y\_l], color='black', linewidth=4)

ax.plot([6, 7], [y\_l, y\_l], color='black', linewidth=4)

# Define a function to add shapes to the plot

rhombus1 = patches.Polygon(

[(2, 1.3), (1.5, 1.6), (2, 1.9), (2.5, 1.6)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus1)

circle = patches.Circle((5, 1.6), radius=0.3, facecolor='black', edgecolor='black', linewidth=8)

ax.add\_patch(circle)

arrow3 = patches.FancyArrow(8, 1.25, 0, line\_length, head\_width=0.3, head\_length=0.3, fc='black', ec='black', linewidth=5)

ax.add\_patch(arrow3)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 3)

plt.axis("off")

# Show the plot

plt.show()

# **Example 6: Visualizing pop Operation on a List**[**¶**](http://localhost:8892/notebooks/Desktop/project/update_last_Python-project/update_List.ipynb#Example-8:-Visualizing-pop-Operation-on-a-List)

Objective: Learn how the pop operation removes an element from a specified position in a list (or the last element by default) and visualize this change.

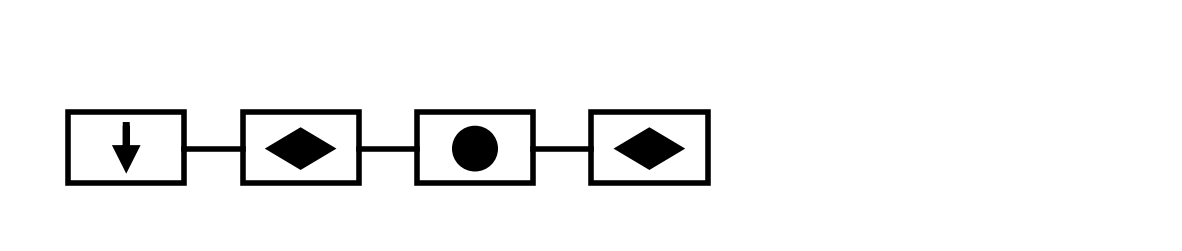
Problem: Start with a list, e.g., [True, 3.14, "B", 2], and then pop an element from it (e.g., the second element). Visualize the list before and after the pop operation.

# Initial list

initial\_list = [True, 3.14, "B", 2]

List(initial\_list)

**Result:**



**Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Define the coordinates and dimensions of the rectangle

x = 1 # x-coordinate of the lower-left corner

y = 1 # y-coordinate of the lower-left corner

phi = 1.61803398875 # Golden ratio

# Calculate the width and height based on the golden ratio

width = 2

height = width / phi # Height is calculated to maintain the golden ratio

# Create rectangles

rectangle1 = patches.Rectangle((x, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle1)

rectangle2 = patches.Rectangle((x + 3, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle2)

rectangle3 = patches.Rectangle((x + 6, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle3)

rectangle4 = patches.Rectangle((x + 9, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle4)

# Create connected lines between rectangles

y\_l = 1.6

ax.plot([3, 4], [y\_l, y\_l], color='black', linewidth=4)

ax.plot([6, 7], [y\_l, y\_l], color='black', linewidth=4)

ax.plot([9, 10], [y\_l, y\_l], color='black', linewidth=4)

# Define a function to add shapes to the plot

arrow = patches.FancyArrow(2, 2, 0, -line\_length, head\_width=0.3, head\_length=0.3, fc='black', ec='black', linewidth=5)

ax.add\_patch(arrow)

rhombus1 = patches.Polygon(

[(5, 1.3), (4.5, 1.6), (5, 1.9), (5.5, 1.6)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus1)

circle = patches.Circle((8, 1.6), radius=0.3, facecolor='black', edgecolor='black', linewidth=8)

ax.add\_patch(circle)

rhombus2 = patches.Polygon(

[(11, 1.3), (10.5, 1.6), (11, 1.9), (11.5, 1.6)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus2)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

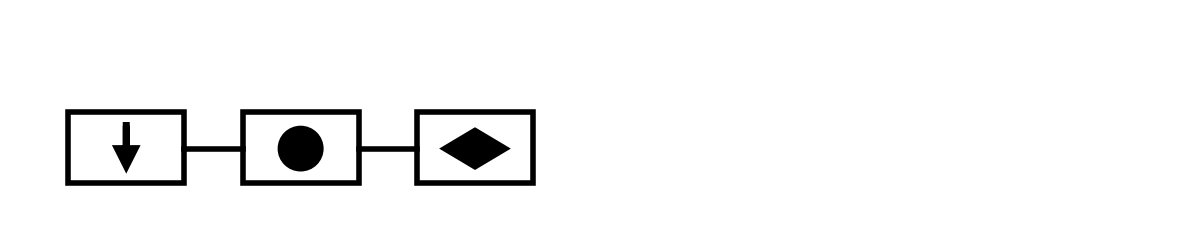
# Show the plot

plt.show()

**Pop the second element (3.14)**

initial\_list.pop(1)

List(initial\_list)

**Result:**

**Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Define the coordinates and dimensions of the rectangle

x = 1 # x-coordinate of the lower-left corner

y = 1 # y-coordinate of the lower-left corner

phi = 1.61803398875 # Golden ratio

# Calculate the width and height based on the golden ratio

width = 2

height = width / phi # Height is calculated to maintain the golden ratio

# Create rectangles

rectangle1 = patches.Rectangle((x, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle1)

rectangle2 = patches.Rectangle((x + 3, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle2)

rectangle3 = patches.Rectangle((x + 6, y), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle3)

# Create connected lines between rectangles

y\_l = 1.6

ax.plot([3, 4], [y\_l, y\_l], color='black', linewidth=4)

ax.plot([6, 7], [y\_l, y\_l], color='black', linewidth=4)

# Define a function to add shapes to the plot

arrow = patches.FancyArrow(2, 2, 0, -line\_length, head\_width=0.3, head\_length=0.3, fc='black', ec='black', linewidth=5)

ax.add\_patch(arrow)

circle = patches.Circle((5, 1.6), radius=0.3, facecolor='black', edgecolor='black', linewidth=8)

ax.add\_patch(circle)

rhombus2 = patches.Polygon(

[(8, 1.3), (7.5, 1.6), (8, 1.9), (8.5, 1.6)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus2)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

# Show the plot

plt.show()

# **Problems for Tuple**

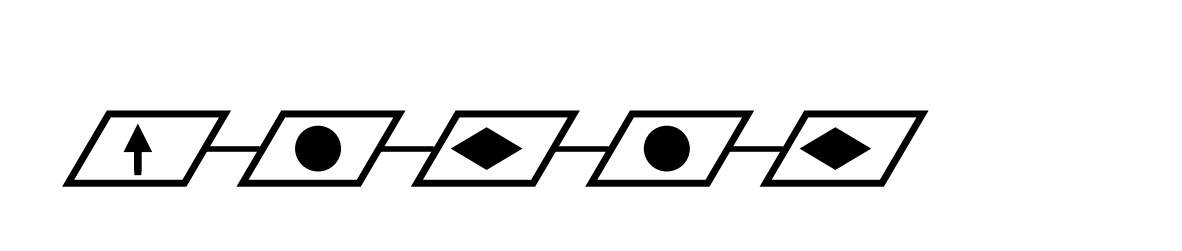
**Example 7: Visualizing Mixed Data Types in a Tuple**

Objective: Understand how to represent mixed data types in a tuple.

Problem: Create a tuple with mixed data types: (True, "A", 4, "B", 4.0).

mixed\_tuple = (True, "A", 4, "B", 4.0)

Tuple(mixed\_tuple) # Wrap the tuple in a list to use your existing List class for visualization.

**Result:  
Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Create paral-s

par1 = patches.Polygon(

[(1,1), (1.7,2.2), (3.7,2.2), (3,1)],

facecolor='None', edgecolor='black', linewidth=5)

ax.add\_patch(par1)

par2 = patches.Polygon(

[(4, 1), (4.7, 2.2), (6.7, 2.2), (6, 1)],

facecolor='None', edgecolor='black', linewidth=5)

ax.add\_patch(par2)

par3 = patches.Polygon(

[(7, 1), (7.7, 2.2), (9.7, 2.2), (9, 1)],

facecolor='None', edgecolor='black', linewidth=5)

ax.add\_patch(par3)

par4 = patches.Polygon(

[(10, 1), (10.7, 2.2), (12.7, 2.2), (12, 1)],

facecolor='None', edgecolor='black', linewidth=5)

ax.add\_patch(par4)

par5 = patches.Polygon(

[(13, 1), (13.7, 2.2), (15.7, 2.2), (15, 1)],

facecolor='None', edgecolor='black', linewidth=5)

ax.add\_patch(par5)

# Create connected lines between paral-s

y\_l = 1.6

ax.plot([3.4, 4.25], [y\_l, y\_l], color='black', linewidth=4)

ax.plot([6.4, 7.25], [y\_l, y\_l], color='black', linewidth=4)

ax.plot([9.4, 10.25], [y\_l, y\_l], color='black', linewidth=4)

ax.plot([12.4, 13.25], [y\_l, y\_l], color='black', linewidth=4)

# Define a function to add shapes to the plot

line\_length = 0.4

arrow = patches.FancyArrow(2.2, 1.2, 0, line\_length, head\_width=0.3, head\_length=0.3, fc='black', ec='black', linewidth=5)

ax.add\_patch(arrow)

circle1 = patches.Circle((5.3, 1.6), radius=0.3, facecolor='black', edgecolor='black', linewidth=8)

ax.add\_patch(circle1)

rhombus1 = patches.Polygon(

[(8.2, 1.3), (7.7, 1.6), (8.2, 1.9), (8.7, 1.6)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus1)

circle2 = patches.Circle((11.3, 1.6), radius=0.3, facecolor='black', edgecolor='black', linewidth=8)

ax.add\_patch(circle2)

rhombus2 = patches.Polygon(

[(14.2, 1.3), (13.7, 1.6), (14.2, 1.9), (14.7, 1.6)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus2)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

# Show the plot

plt.show()

# **Problems for Set**

**Example 8: Visualizing Mixed Data Types in a Set**

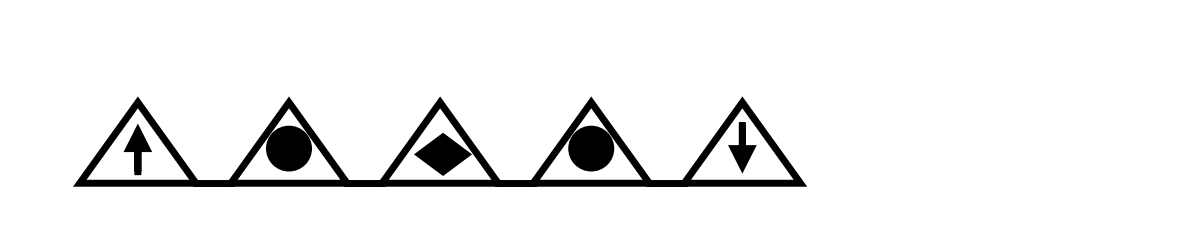
Objective: Understand how to represent mixed data types in a set.

Problem: Create a set with mixed data types: {True, "A", 4, "B", False}.

mixed\_set = {True, "A", 4, "B", False}

Set(mixed\_set)

**Result:**

**  
Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Create triangles

triangle1 = patches.Polygon(

[(1.2, 1), (2.2, 2.4), (3.2, 1)],

facecolor='none', edgecolor='black', linewidth=5)

ax.add\_patch(triangle1)

triangle2 = patches.Polygon(

[(3.8, 1), (4.8, 2.4), (5.8, 1)],

facecolor='none', edgecolor='black', linewidth=5)

ax.add\_patch(triangle2)

triangle3 = patches.Polygon(

[(6.4, 1), (7.4, 2.4), (8.4, 1)],

facecolor='none', edgecolor='black', linewidth=5)

ax.add\_patch(triangle3)

triangle4 = patches.Polygon(

[(9, 1), (10, 2.4), (11, 1)],

facecolor='none', edgecolor='black', linewidth=5)

ax.add\_patch(triangle4)

triangle5 = patches.Polygon(

[(11.6, 1), (12.6, 2.4), (13.6, 1)],

facecolor='none', edgecolor='black', linewidth=5)

ax.add\_patch(triangle5)

# Create connected lines between triangles

y\_l = 1

ax.plot([3.2, 3.8], [y\_l, y\_l], color='black', linewidth=5)

ax.plot([5.8, 6.4], [y\_l, y\_l], color='black', linewidth=5)

ax.plot([8.4, 9], [y\_l, y\_l], color='black', linewidth=5)

ax.plot([11, 11.6], [y\_l, y\_l], color='black', linewidth=5)

# Define a function to add shapes to the plot

line\_length = 0.4

arrow1 = patches.FancyArrow(2.2, 1.2, 0, line\_length, head\_width=0.3, head\_length=0.3, fc='black', ec='black', linewidth=5)

ax.add\_patch(arrow1)

circle1 = patches.Circle((4.8, 1.6), radius=0.3, facecolor='black', edgecolor='black', linewidth=8)

ax.add\_patch(circle1)

rhombus1 = patches.Polygon(

[(7.45, 1.2), (7.05, 1.5), (7.45, 1.8), (7.85, 1.5)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus1)

circle2 = patches.Circle((10, 1.6), radius=0.3, facecolor='black', edgecolor='black', linewidth=8)

ax.add\_patch(circle2)

arrow2 = patches.FancyArrow(12.6, 2, 0, -line\_length, head\_width=0.3, head\_length=0.3, fc='black', ec='black', linewidth=5)

ax.add\_patch(arrow2)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

# Show the plot

plt.show()

# **Problems for String and Char**

**Char Type Example:**

Objective: Create a variable with the **char** type and display it.

char = 'a'

String(char)

**Result:**

**  
Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Create circles

circle1 = patches.Circle((1, 1), radius=0.8, facecolor='black', edgecolor='black', linewidth=8)

ax.add\_patch(circle1)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

# Show the plot

plt.show()

**Using a Single-Character String in a Larger String:**

Objective: Use a single-character string within a larger string.

initial = "Hello, "

name = "Alice"

greeting = initial + name + "!"

String(greeting)

**Result:**

**  
Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Create circles

circle1 = patches.Circle((1, 1), radius=0.6, facecolor='black', edgecolor='black', linewidth=8)

ax.add\_patch(circle1)

circle2 = patches.Circle((2.9, 1), radius=0.6, facecolor='black', edgecolor='black', linewidth=8)

ax.add\_patch(circle2)

# Create connected lines between circles

y\_l = 1

ax.plot([1.5, 2.3], [y\_l, y\_l], color='black', linewidth=5)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

# Show the plot

plt.show()

# **Problems for Primitive types**

**Numeric Values Example:**

Objective: Create numeric variables **(integers,complex and floats)** and display them.

integer\_value = 42

float\_value = 3.14

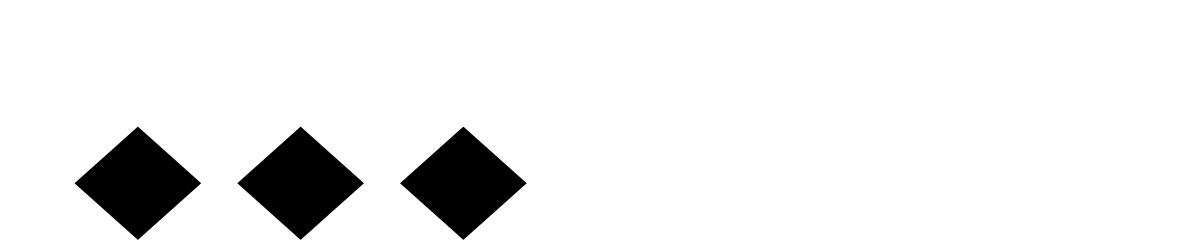
complex\_value = 5j+2

Numeric(integer\_value)

Numeric(float\_value)

Numeric(complex\_value)

**Result:**

**  
Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

# Create rombs

romb1 = patches.Polygon(

[(1.2, 1), (2.2, 1.9), (3.2, 1),(2.2,0.1)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(romb1)

romb2 = patches.Polygon(

[(4, 1), (5, 1.9), (6, 1),(5,0.1)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(romb2)

romb3 = patches.Polygon(

[(6.8, 1), (7.8, 1.9), (8.8, 1),(7.8,0.1)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(romb3)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

# Show the plot

plt.show()

**Boolean Values Example:**

Objective: Create boolean variables **(True and False)** and display them.

true\_value = True

false\_value = False

Boolean(true\_value)

Boolean(false\_value)

**Result:**

**  
Code:**

import matplotlib.pyplot as plt

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

line\_length=1.5

#1

arrow1 = patches.FancyArrow(2.05, 0.3, 0, line\_length, head\_width=0.5, head\_length=0.5, fc='black', ec='black', linewidth=8)

ax.add\_patch(arrow1)

#2

arrow2 = patches.FancyArrow(5, 2.3, 0, -line\_length, head\_width=0.5, head\_length=0.5, fc='black', ec='black', linewidth=8)

ax.add\_patch(arrow2)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

plt.show()

**None Type Example:**

Objective: Create a variable with the **None** type and display it.

none\_value = None

None(none\_value)

**Result:**

**  
Code:**

import matplotlib.pyplot as plt

from matplotlib.patches import Circle

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

circle = Circle((2.5,1.5), radius=1.3, facecolor='None', linewidth=7, edgecolor="black")

ax.add\_patch(circle)

ax.plot([0.9, 4.1], [0.3, 2.7], color='black', linewidth=7)

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

plt.show()

**Dictionary Example**

Objective: Create a dictionary and display them.

dictionary={'A':{"ABS","DFS"},'B':"ASDFGHJ",4:[1,2,3]}

**Result:  
Code:**

import matplotlib.pyplot as plt

from matplotlib.patches import Circle

from matplotlib.lines import Line2D

import matplotlib.patches as patches

# Create a figure and axis

fig, ax = plt.subplots(figsize=(15, 3))

x = 1.5

y = 2

# Shapes of keys##############

#1-element

circle = Circle((x, y), 0.3, linewidth=1, edgecolor='black', facecolor='black')

ax.add\_patch(circle)

#2-element

rhombus1 = patches.Polygon(

[(1.5, 0.5), (1.2, 0.8), (1.5, 1.1), (1.8, 0.8)],

facecolor='black', edgecolor='black', linewidth=5)

ax.add\_patch(rhombus1)

#3-element

triangle1 = patches.Polygon(

[(1.2, 2.9), (1.5, 3.4), (1.8, 2.9)],

facecolor='None', edgecolor='black', linewidth=5)

ax.add\_patch(triangle1)

####################################################

#Shapes of values###############

#1-element

circle2 = Circle((x+2, y), 0.3, linewidth=1, edgecolor='black', facecolor='black')

ax.add\_patch(circle2)

circle3 = Circle((x+2.8, y), 0.3, linewidth=1, edgecolor='black', facecolor='black')

ax.add\_patch(circle3)

ax.plot([3.5, 4], [y, y], color='black', linewidth=5)

######

#2-element

width=0.8

height=0.7

rectangle1 = patches.Rectangle((3.2, 0.5), width, height, linewidth=4, edgecolor='black', facecolor='none')

ax.add\_patch(rectangle1)

#3-element

par1 = patches.Polygon(

[(3.2, 2.9), (3.4, 3.4), (4.2, 3.4), (4, 2.9)],

facecolor='None', edgecolor='black', linewidth=5)

ax.add\_patch(par1)

######

#############################

# Line of keys and teeth###############

#1-element

line1 = Line2D([x + 0.2, x + 1.5], [y,y], color='black', linewidth=5)

ax.add\_line(line1)

t1 = Line2D([x + 1.5, x + 1.5], [y - 0.25, y], color='black', linewidth=5)

t2 = Line2D([x + 0.4, x + 0.4], [y - 0.25, y], color='black', linewidth=5)

t3 = Line2D([x + 0.8, x + 0.8], [y - 0.25, y], color='black', linewidth=5)

ax.add\_line(t1)

ax.add\_line(t2)

ax.add\_line(t3)

####

#2-element

y2=0.8

line2 = Line2D([x + 0.2, x + 1.5], [y2, y2], color='black', linewidth=5)

ax.add\_line(line2)

t1\_1 = Line2D([x + 1.5, x + 1.5], [y2 - 0.25, y2], color='black', linewidth=5)

t2\_1 = Line2D([x + 0.4, x + 0.4], [y2 - 0.25, y2], color='black', linewidth=5)

t3\_1 = Line2D([x + 0.8, x + 0.8], [y2 - 0.25, y2], color='black', linewidth=5)

ax.add\_line(t1\_1)

ax.add\_line(t2\_1)

ax.add\_line(t3\_1)

###

#3-element

y3=3.2

line2 = Line2D([x + 0.2, x + 1.5], [y3, y3], color='black', linewidth=5)

ax.add\_line(line2)

t1\_1 = Line2D([x + 1.5, x + 1.5], [y3 - 0.25, y3], color='black', linewidth=5)

t2\_1 = Line2D([x + 0.4, x + 0.4], [y3 - 0.25, y3], color='black', linewidth=5)

t3\_1 = Line2D([x + 0.8, x + 0.8], [y3 - 0.25, y3], color='black', linewidth=5)

ax.add\_line(t1\_1)

ax.add\_line(t2\_1)

ax.add\_line(t3\_1)

#############################################

# Set the x and y limits

ax.set\_xlim(0, 20)

ax.set\_ylim(0, 4)

plt.axis("off")

plt.show()