Most of my code uses material not included in the booklet. Here are some examples:

Code Snippet: canvas.delete("all")
Description:
This code snippet is used to delete all objects from a canvas in a graphical user interface (GUI) application. It is often used to clear the canvas before drawing new objects or refreshing the display.

Demonstration:

```
import tkinter as tk

# Create a window
window = tk.Tk()
window.title("Canvas Demo")

# Create a canvas
canvas = tk.Canvas(window, width=400, height=400)
canvas.pack()

# Draw some objects on the canvas
rectangle = canvas.create_rectangle(50, 50, 200, 200, fill="blue")
text = canvas.create_text(225, 225, text="Hello, World!", fill="white")

# Wait for a moment before deleting
window.after(2000)

# Delete all objects from the canvas
canvas.delete("all")

# Capture a screenshot or manually observe the canvas

# Close the window
window.mainloop()
```

Usage of command=lambda
Description:
The command=lambda is a commonly used construct in Python that allows you to assign a function or method to a button or widget in a GUI application. It enables you to define a callback function or method that gets executed when the associated button or widget is clicked or triggered.

Code snippet:

```
button = tk.Button(window, text="Click Me", command=lambda: handle_click())
```

Demonstration:
In this example, we have created a button widget with the label "Click Me". The command parameter is assigned a lambda function, which is an anonymous function defined on the spot. In this case, the lambda function is lambda: handle_click(), which calls the handle_click() function when the button is clicked.

Here's a step-by-step demonstration of what happens when you run this code:

1. The GUI window is created, and the button "Click Me" is displayed.
2. You click the button.
3. The handle_click() function is executed.
4. Any code inside the handle_click() function is run. You can customize this function to perform any desired action or implement specific functionality.

Please note that you will need to define and implement the handle_click() function separately in your code to perform the desired actions when the button is clicked. The lambda keyword allows you to create a quick inline function without explicitly defining a named function for simple scenarios.

Remember to replace handle_click() with the appropriate function or method name based on your specific use case.

Usage of canvas.create_window
Description:
The canvas.create_window method is used in graphical user interface (GUI) applications to create a window or container within a canvas. It allows you to embed other widgets, such as buttons, labels, or entry fields, inside the canvas. This can be useful for creating custom user interfaces or complex layouts where you need more control over the positioning and interaction of the widgets.

Code Snippet:
```
button_frame = tk.Frame(window)
button = tk.Button(button_frame, text="Click Me")
canvas.create_window(x, y, window=button_frame, anchor=tk.NW)
```

Demonstration:
In this example, we have created a frame button_frame and a button button within that frame. The canvas.create_window method is then used to place the button_frame container at a specific position (x, y) on the canvas, with the anchor set to the northwest (tk.NW) corner of the container.

Here's a step-by-step demonstration of what happens when you run this code:

1. The GUI window is created.
2. The button_frame is created as a container to hold the button.
3. The button button is created inside the button_frame.
4. The canvas.create_window method is called to place the button_frame on the canvas at the specified position (x, y).
5. The button_frame and the button button are rendered within the canvas at the designated position.

Please note that you will need to customize the x and y coordinates based on your desired positioning on the canvas. Additionally, you can add other widgets or elements to the button_frame container before placing it on the canvas using canvas.create_window.

Make sure you have the necessary import statements and a GUI window set up before using the canvas.create_window method.

Usage of canvas.create_image
Description:
The canvas.create_image method is used in graphical user interface (GUI) applications to display an image on a canvas. It allows you to load an image file from your system or use an existing image object and place it at a specific position on the canvas.

Code Snippet:

```
image = tk.PhotoImage(file="image.png")
canvas.create_image(x, y, image=image, anchor=tk.NW)
```

Demonstration:
In this example, we have loaded an image file named "image.png" using the tk.PhotoImage class and assigned it to the variable image. The canvas.create_image method is then used to display the image on the canvas at the specified position (x, y), with the anchor set to the northwest (tk.NW) corner of the image.

Here's a step-by-step demonstration of what happens when you run this code:

1. The GUI window is created.
2. The image file "image.png" is loaded using tk.PhotoImage and assigned to the variable image.
3. The canvas.create_image method is called to place the image on the canvas at the specified position (x, y).
4. The image is rendered within the canvas at the designated position.

Please note that you need to replace "image.png" with the actual path to your image file. Make sure the image file exists at the specified location. Additionally, customize the x and y coordinates based on your desired positioning on the canvas.

I recommend using PNGs for when creating your images.

Remember to have the necessary import statements, a GUI window set up, and a canvas created before using the canvas.create_image method.

Usage of canvas.create_text
Description:
The canvas.create_text method is used in graphical user interface (GUI) applications to display text on a canvas. It allows you to specify the position, font, color, and other formatting options for the text displayed on the canvas.

Code Snippet:
canvas.create_text(x, y, text="Hello, World!", font=("Arial", 12), fill="black", anchor=tk.NW)
Demonstration:
In this example, we are using the canvas.create_text method to display the text "Hello, World!" on the canvas at the specified position (x, y). We have also set the font to "Arial" with a size of 12 points, the text color to black, and the anchor to the northwest (tk.NW) corner of the text.

Here's a step-by-step demonstration of what happens when you run this code:

1. The GUI window is created.
2. The canvas.create_text method is called to place the text "Hello, World!" on the canvas at the specified position (x, y).
3. The text is rendered within the canvas using the specified font, color, and anchor.

Please note that you can customize the x and y coordinates based on your desired positioning on the canvas. Additionally, you can adjust the font, size, and color values to fit your specific requirements.

Ensure that you have the necessary import statements, a GUI window set up, and a canvas created before using the canvas.create_text method.

Code snippet:

```
    # Create numberpad for math input
    buttons = [
        ("1", 0, 0),
        ("2", 1, 0),
        ("3", 2, 0),
        ("Backspace", 3, 0),
        ("4", 0, 1),
        ("5", 1, 1),
```

```
        ("6", 2, 1),
        ("Enter", 3, 1),
        ("7", 0, 2),
        ("8", 1, 2),
        ("9", 2, 2),
        ("0", 3, 2)
    ]

    for i, (label, col, row) in enumerate(buttons):
        if label == "Backspace":
            button = Button(window, image=button_images[3], width=100, height=100,
command=lambda: answer_entry.delete(len(answer_entry.get()) - 1))
        elif label == "Enter":
            button = Button(window, image=button_images[7], bd=0, command=lambda:
check_answer(question, answer_entry.get(), level))
        else:
            button = Button(window, image=button_images[i], width=100, height=100,
command=lambda num=i: numberpad_clicked(num))

        button_window = canvas.create_window(375 + col * 100, 375 + row * 100, anchor='nw',
window=button)
```

Explanation of Code:
The given code is creating a number pad for math input in a graphical user interface (GUI) application. It defines a list of tuples called `buttons`, where each tuple represents a button on the number pad. Each tuple contains the label of the button, the column position, and the row position on the number pad.

The code then iterates over the `buttons` list using `enumerate`, which provides both the index (`i`) and the corresponding tuple (`(label, col, row)`) from the list. For each button, it creates a `Button` widget, customizing it based on the label.

If the label is "Backspace", it creates a button with an image and a command that deletes the last character in the answer entry field (`answer_entry`). If the label is "Enter", it creates a button with an image and a command that checks the answer against a question. For other labels, it creates buttons with images and a command that triggers a `numberpad_clicked` function with the corresponding number as an argument.

Finally, the `canvas.create_window` method is used to place each button at a specific position on the canvas based on its column and row values.

Explanation of Tuples:

In Python, a tuple is an ordered, immutable collection of elements enclosed in parentheses `()`. Each element in a tuple can be of any data type (such as strings, numbers, or other objects) and can have different meanings. In the given code, the tuples represent the buttons on the number pad and store the label, column position, and row position for each button.

For example, the tuple `("1", 0, 0)` represents the button labeled "1" with a column position of 0 and a row position of 0 on the number pad.

Explanation of `enumerate`:
`enumerate` is a built-in Python function used to iterate over a sequence (such as a list, tuple, or string) while also keeping track of the index of each element. It returns pairs of the index and corresponding element.

In the given code, `enumerate(buttons)` is used to iterate over the `buttons` list and obtain both the index (`i`) and the corresponding button tuple (`(label, col, row)`) in each iteration. This allows convenient access to the index and tuple elements simultaneously.

How the Code Works:
1. The `buttons` list is defined, representing the buttons on the number pad.
2. The code iterates over the `buttons` list using `enumerate`, unpacking each tuple into variables `i`, `label`, `col`, and `row`.
3. Depending on the label of the button, a specific `Button` widget is created with appropriate properties and commands.
   - If the label is "Backspace", a button is created with an image and a command that deletes the last character in the `answer_entry` field.
   - If the label is "Enter", a button is created with an image, no border (`bd=0`), and a command that checks the answer against a question.
   - For other labels, buttons are created with images and a command that triggers the `numberpad_clicked` function with the corresponding number as an argument.
4. The `canvas.create_window` method is used to place each button on the canvas at a specific position based on its column and row values.

By using tuples to store the button information, `enumerate` to iterate over the buttons with their indices, and conditional statements to handle different button labels, the code efficiently creates and positions the buttons on the number pad for math input in the GUI application.

Usage of canvas.move
Description:
The canvas.move method is used in graphical user interface (GUI) applications to move a graphical object or item on a canvas. It allows you to specify the object's unique identifier (ID) and the amount by which you want to move it in the x and y directions.

Code Snippet:
canvas.move(object_id, delta_x, delta_y)

Demonstration:
In this example, we assume that you have already created a canvas and drawn an object on it. The object_id refers to the unique identifier assigned to the object you want to move. delta_x and delta_y represent the distances you want to move the object in the x and y directions, respectively.

Here's a step-by-step demonstration of what happens when you run this code:

1. The GUI window is created.
2. The canvas is created.
3. An object (such as a shape or image) is drawn on the canvas and assigned a unique identifier (object_id).
4. The canvas.move method is called with the object's ID and the desired delta values.
5. The object is moved by the specified amount in the x and y directions on the canvas.

Please note that you need to replace object_id with the actual ID of the object you want to move. Additionally, customize the delta_x and delta_y values based on the desired movement distance.

The canvas.move method is useful for animating objects, implementing drag-and-drop functionality, or dynamically updating the position of objects on the canvas in response to user interaction or program logic.

Ensure that you have the necessary import statements, a GUI window set up, and a canvas created before using the canvas.move method.

Usage of canvas.coords
Description:
The canvas.coords method is used in graphical user interface (GUI) applications to retrieve or modify the coordinates of a graphical object on a canvas. It allows you to access or update the position of an object by providing its unique identifier (ID) and the new coordinates.

Code Snippet:

```
# Get the current position of the attack image
    x, y = canvas.coords(attack_image)
```

Demonstration:
In this example, we assume that you have already created a canvas and drawn an object on it. The object_id refers to the unique identifier assigned to the object whose coordinates you want to access or update.

To access the current coordinates of the object, you can use canvas.coords as shown in the first line of the code snippet. It returns a list of the current coordinates of the object.

To update the coordinates of the object, you can use canvas.coords as shown in the second line of the code snippet. You need to provide the object_id and the new coordinates as a list.

Here's a step-by-step demonstration of what happens when you run this code:

1. The GUI window is created.
2. The canvas is created.
3. An object (such as a shape or image) is drawn on the canvas and assigned a unique identifier (object_id).
4. The canvas.coords method is called with the object_id to retrieve the current coordinates of the object. The result is stored in the current_coords variable.
5. The canvas.coords method is called again with the object_id and the new coordinates (new_coords). This updates the position of the object on the canvas.

Please note that you need to replace object_id with the actual ID of the object you want to access or update. Additionally, customize the new_coords values based on the desired new coordinates.

The canvas.coords method is useful for retrieving the current position of an object, modifying the object's position, or animating objects by changing their coordinates over time.

Ensure that you have the necessary import statements, a GUI window set up, and a canvas created before using the canvas.coords method.

Usage of canvas.after
Description:
The canvas.after method is used in graphical user interface (GUI) applications to schedule the execution of a function or code snippet after a specified delay. It allows you to create timed events, animations, or delay the execution of certain actions in your GUI application.

Code Snippet:
canvas.after(delay, function_name, additional_arguments)
Demonstration:
In this example, delay represents the time delay in milliseconds before the function_name is executed. The additional_arguments parameter allows you to pass additional arguments to the function if needed.

Here's a step-by-step demonstration of what happens when you run this code:

1. The GUI window is created.
2. The canvas is created.
3. The canvas.after method is called with the specified delay value, function_name, and any additional arguments (*args) you want to pass to the function.

4. After the specified delay, the function_name is executed, along with the provided arguments, if any.

Please note that you need to replace delay with the desired delay time in milliseconds. The function_name should be replaced with the name of the function you want to execute after the delay. If there are additional arguments you want to pass to the function, you can provide them as additional parameters after function_name.

The canvas.after method is commonly used for creating animations, timed events, or delaying the execution of specific actions in GUI applications. It allows you to schedule code to run after a certain amount of time, providing a way to introduce delays or create timed sequences of actions.

Ensure that you have the necessary import statements, a GUI window set up, and a canvas created before using the canvas.after method.

Usage of window.resizable
Description:
The window.resizable method is used in graphical user interface (GUI) applications to control whether the application window can be resized by the user. It allows you to set the resizable behavior of the application window in both the horizontal (width) and vertical (height) directions.

Code Snippet:
window.resizable(width=True, height=True)
Demonstration:
In this example, the width and height parameters control the resizable behavior of the window. By passing True, the window will be resizable in the corresponding direction. By passing False, the window will be fixed and non-resizable in that direction.

Here's a step-by-step demonstration of what happens when you run this code:

1. The GUI window is created.
2. The window.resizable method is called with the desired resizable behavior for the width and height.
    - If width=True and height=True, the window can be resized by the user in both the horizontal and vertical directions.
    - If width=False and height=True, the window cannot be resized horizontally but can be resized vertically.
    - If width=True and height=False, the window can be resized horizontally but cannot be resized vertically.
    - If width=False and height=False, the window is fixed and non-resizable in both the horizontal and vertical directions.

Please note that you can customize the values of width and height based on your desired resizable behavior.

The window.resizable method provides control over the window's resizing behavior, allowing you to enforce fixed dimensions or allow users to adjust the size of the application window according to their preferences.

Ensure that you have the necessary import statements and a GUI window created before using the window.resizable method.

Usage of ttk.Style and style.configure
Description:
The ttk.Style class in the tkinter library is used to define and customize the appearance of widgets in graphical user interface (GUI) applications. The style.configure method is used to configure the visual properties of a specific widget style.

Code Snippet:

```
style = ttk.Style()
style.theme_use('clam')
style.configure("blue.Horizontal.TProgressbar", foreground='blue', background='blue')
```

Demonstration:
In this example, we create an instance of the ttk.Style class and assign it to the variable style. We then specify the theme to be used by calling the theme_use method on the style object and passing the theme name as a parameter.

After that, we use the style.configure method to configure the visual properties of a specific widget style. In this case, we configure a style named "blue.Horizontal.TProgressbar" to have a blue foreground and blue background color.

Here's a step-by-step demonstration of what happens when you run this code:

The GUI window is created.
An instance of the ttk.Style class is created and assigned to the variable style.
The theme_use method is called on the style object to set the theme to 'clam'.
The style.configure method is called to configure the visual properties of the "blue.Horizontal.TProgressbar" style. The foreground color and background color are set to blue.
Please note that you can customize the style name, theme, and visual properties (foreground color, background color, font, etc.) based on your requirements.

The ttk.Style class and style.configure method allow you to create and modify styles for various tkinter widgets, providing control over the appearance and visual properties of your GUI application.

Ensure that you have the necessary import statements and a GUI window created before using the ttk.Style and style.configure methods.

Usage of Canvas and canvas.pack
Description:
The Canvas class in the tkinter library is used to create a rectangular area on which you can draw and display graphical elements, such as shapes, lines, images, and text. The canvas.pack method is used to add the canvas to the GUI window and configure its layout behavior.

Code Snippet:
```
canvas = Canvas(window, width=1024, height=768)
canvas.pack(fill='both', expand=True)
```
Demonstration:
In this example, we create an instance of the Canvas class called canvas with a width of 1024 pixels and a height of 768 pixels. We pass the window object (representing the GUI window) as the first parameter to associate the canvas with the window.

Next, we use the canvas.pack method to add the canvas to the window and configure its layout behavior. In this case, we set the fill option to 'both', which means the canvas will fill both horizontally and vertically to occupy any available space. We also set the expand option to True, allowing the canvas to expand along with its parent container if more space is available.

Here's a step-by-step demonstration of what happens when you run this code:

1. The GUI window is created.
2. An instance of the Canvas class is created and assigned to the variable canvas.
3. The canvas is associated with the window by passing it as the first parameter to the Canvas constructor.
4. The canvas is added to the GUI window using the canvas.pack method.
5. The canvas fills the available space both horizontally and vertically and expands if more space is available.

Please note that you can customize the width and height of the canvas based on your requirements. Additionally, you can adjust the fill and expand options in the canvas.pack method to achieve the desired layout behavior.

The Canvas class provides a versatile drawing area on which you can create and display various graphical elements. The canvas.pack method allows you to add the canvas to your GUI window and control its layout behavior.

Ensure that you have the necessary import statements and a GUI window created before using the Canvas class and canvas.pack method.