Step-by-Step Guide: Programming a Python Script with Console Launch

This step-by-step guide will help you learn how to program a Python script that can be launched from the console. We will gradually build the script, adding new features and functionality at each step. The final code will include a graphical user interface (GUI) with various features such as BMI calculation, data logging, data storage in CSV format, data visualization, and more.

Step 1: Simple "Hello, World!"

Let's start with a simple "Hello, World!" program to ensure that everything is set up correctly.

1. Create a new Python file called `script.py`.
2. Open the file in a text editor or an integrated development environment (IDE) of your choice.
3. Add the following code to the file:

```
print("Hello, World!")
```

4. Save the file.

To run the script:

1. Open the console or terminal.
2. Navigate to the directory where the `script.py` file is located.
3. Enter the following command:

```
python3 script.py
```

4. You should see the output `Hello, World!` in the console.

Congratulations! You've successfully created and executed a simple Python script. Let's move on to the next step.

Step 2: "Hello, World!" with Name as an Argument

Next, let's modify the script to accept a name as an argument and customize the greeting message.

1. Open the `script.py` file.
2. Replace the existing code with the following:

```
import sys

if len(sys.argv) > 1:
    name = sys.argv[1]
```

```python
else:
    name = "World"

print(f"Hello, {name}!")
```

3. Save the file.

To run the script with a name:

1. Open the console or terminal.
2. Navigate to the directory where the `script.py` file is located.
3. Enter the following command:

```
python3 script.py Alice
```

4. You should see the output `Hello, Alice!` in the console.

If you don't provide a name as an argument, the script will use "World" as the default name.

Great job! You've enhanced the script to accept an argument and customized the greeting message. Let's continue to the next step.

Step 3: BMI Calculator with Height and Weight

In this step, we'll add functionality to calculate the body mass index (BMI) based on user-provided height and weight values.

1. Open the `script.py` file.
2. Replace the existing code with the following:

```python
def calculate_bmi(height=0, weight=0):
    if height == 0:
        height = float(input("Enter height in meters: "))
    if weight == 0:
        weight = float(input("Enter weight in kilograms: "))

    bmi = weight / (height ** 2)
    return bmi

bmi = calculate_bmi()
print(f"Your BMI is: {bmi:.2f}")
```

3. Save the file.

To run the script and calculate BMI:

1. Open the console or terminal.
2. Navigate to the directory where the `script.py` file is located.

3. Enter the following command:

```
python script.py Alice
```

4. Follow the prompts to enter the height and weight values.
5. You should see the output `Hello, Alice!` followed by the calculated BMI.

The script will ask for the height and weight values if they are not provided as command-line arguments. The BMI will be calculated and displayed with two decimal places.

Fantastic progress! You've implemented a BMI calculator in your script. Let's move on to the next step.

Step 4: Script with a GUI Interface for Height and Weight Input

In this step, we'll enhance the script by adding a graphical user interface (GUI) to input the height and weight values.

1. Install the required dependencies:

```
pip3 install tkinter
```

2. Open the `script.py` file.
3. Replace the existing code with the following:

```
import tkinter as tk

def calculate_bmi(height=0, weight=0):
    if height == 0:
    # If height and weight are not provided, retrieve them from the GUI
entry fields
        height = float(height_entry.get())
        weight = float(weight_entry.get())
    # Calculate BMI
    bmi = weight / (height ** 2)
    result_label.config(text=f"Patient's BMI is: {bmi:.2f}")
    return bmi

window = tk.Tk()
window.title("BMI Calculator")

height_label = tk.Label(window, text="Enter height in meters:")
height_label.pack()
height_entry = tk.Entry(window)
height_entry.pack()

weight_label = tk.Label(window, text="Enter weight in kilograms:")
weight_label.pack()
weight_entry = tk.Entry(window)
```

```
weight_entry.pack()

result_label = tk.Label(window)
result_label.pack()

calculate_button = tk.Button(window, text="Calculate BMI",
command=calculate_bmi)
calculate_button.pack()

window.mainloop()
```

4. Save the file.

To run the script with the GUI interface:

1. Open the console or terminal.
2. Navigate to the directory where the `script.py` file is located.
3. Enter the following command:

```
python3 script.py
```

4. The GUI window titled "BMI Calculator" will appear.
5. Enter the height and weight values in the corresponding entry fields.
6. Click the "Calculate BMI" button.
7. The calculated BMI will be displayed below the button.

You now have a graphical interface for entering the height and weight values. The BMI calculation and result display are handled through the GUI. Well done!

Let's proceed to the next step.

Step 5: Adding a Logger Component to Log What's Happening

In this step, we'll introduce logging to the script to track and log various events and messages.

1. Open the `script.py` file.
2. Add the following code at the beginning of the file, before the `import` statements:

```
import logging

# Configure logger
logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(levelname)s
- %(message)s")
logger = logging.getLogger("EDC")
```

3. Replace the existing `calculate_bmi` function with the following code:

```
def calculate_bmi(height=0, weight=0):
    if height == 0:
    # If height and weight are not provided, retrieve them from the GUI
entry fields
        height = float(height_entry.get())
        weight = float(weight_entry.get())
    # Calculate BMI
    bmi = weight / (height ** 2)
    result_label.config(text=f"Patient's BMI is: {bmi:.2f}")

    logger.info(f"Height: {height} m, Weight: {weight} kg")
    logger.info(f"BMI: {bmi:.2f}")
    return bmi
# ...
```

4. Save the file.

To run the script with logging:

1. Open the console or terminal.
2. Navigate to the directory where the `script.py` file is located.
3. Enter the following command:

```
python3 script.py
```

4. Check the console output for logged messages.

You've successfully added a logger component to the script, and now you can track and log important events and messages during execution.

Let's proceed to the next step.

Step 6: GUI Can Save Height, Weight, and Cardiac Event to a CSV

In this step, we'll add a the variable cardiac event and add functionality to save the height, weight, and cardiac event data entered through the GUI to a CSV file.

1. Open the `script.py` file.
2. Add the following `import` statement at the beginning of the file, after the other `import` statements:

```
import csv
```

3. Add the following code after the `weight_entry.pack()` line:

```
```

```
# ...
weight_entry.pack()

cardiac_event_var = tk.StringVar()
cardiac_event_var.set("No") # Set the default value to "No"
cardiac_event_checkbox = tk.Checkbutton(window, text="Cardiac Event
(Yes/No)", variable=cardiac_event_var, onvalue="Yes", offvalue="No")
cardiac_event_checkbox.pack()

# ...
```

4. Replace the existing `calculate_bmi` function with the following code:

```
def calculate_bmi(height=0, weight=0):
    if height == 0:
    # If height and weight are not provided, retrieve them from the GUI
entry fields
        height = float(height_entry.get())
        weight = float(weight_entry.get())
    # Calculate BMI
    bmi = weight / (height ** 2)
    result_label.config(text=f"Patient's BMI is: {bmi:.2f}")

    cardiac_event = cardiac_event_var.get()
    logger.info(f"Height: {height} m, Weight: {weight} kg")
    logger.info(f"BMI: {bmi:.2f}, Cardiac Event: {cardiac_event}")

    # Save patient data
    with open("data.csv", "a", newline="") as file:
        writer = csv.writer(file)
        writer.writerow([height, weight, bmi, cardiac_event])
    return bmi
# ...
```

5. Save the file.

To run the script and save data to a CSV file:

1. Open the console or terminal.
2. Navigate to the directory where the `script.py` file is located.
3. Enter the following command:

```
python3 script.py
```

4. Use the GUI to enter height, weight, and cardiac event data.
5. Click the "Calculate BMI" button.

6. The calculated BMI will be displayed, and the data will be saved to a CSV file named `data.csv` in the same directory.

Congratulations! You've added the functionality to save patient data to a CSV file. The data will include height, weight, BMI, and cardiac event information.

Let's move on to the next step.

Step 7: GUI Can Display Rows of CSV

In this step, we'll add functionality to display all the rows from the CSV file in the GUI.

1. Open the `script.py` file.
2. Add the following code after the `calculate_bmi` function:

```
def calculate_bmi():
    # ...

def display_data():
    # Read data from the CSV file
    with open("data.csv", "r") as file:
        reader = csv.reader(file)
        data = list(reader)

    # Create a new window to display the data
    display_window = tk.Toplevel(window)
    display_window.title("Records")

    # Create a label for each row of data and pack them in the window
    for row in data:
        row_label = tk.Label(display_window, text=row)
        row_label.pack()

window = tk.Tk()
# ...
```

3. Modify the `window` definition code as follows:

```
window = tk.Tk()
window.title("Electronic Data Capture")

# ...
```

4. Add the following code after the `calculate_button.pack()` line:

```
# ...
calculate_button.pack()
```

```
display_button = tk.Button(window, text="Records", command=display_data)
display_button.pack()

# ...
```

5. Save the file.

To run the script and display the data:

1. Open the console or terminal.
2. Navigate to the directory where the `script.py` file is located.
3. Enter the following command:

```
python3 script.py
```

4. Use the GUI to enter height, weight, and cardiac event data and save it.
5. Click the "Records" button.
6. A new window titled "Records" will open, showing all the rows from the `data.csv` file.

You've successfully implemented the feature to display all the rows from the CSV file in a separate window of the GUI.

Great work! Let's proceed to the next step.

Step 8: GUI Can Delete Rows from CSV

In this step, we'll add functionality to delete rows from the CSV file using the GUI.

1. Open the `script.py` file.
2. Replace the existing `display_data` function with the following code:

```
def display_data():
    # Read data from the CSV file
    with open("data.csv", "r") as file:
        reader = csv.reader(file)
        data = list(reader)

    # Create a new window to display the data
    display_window = tk.Toplevel(window)
    display_window.title("Data Display")

    # Create a label for each row of data, a delete button, and pack them
in the window
    for i, row in enumerate(data):
        row_label = tk.Label(display_window, text=row)
        row_label.pack()

        delete_button = tk.Button(display_window, text="Delete",
command=lambda i=i: delete_row(i, display_window))
```

```
        delete_button.pack()

def delete_row(index, display_window):
    # Read data from the CSV file
    with open("data.csv", "r") as file:
        reader = csv.reader(file)
        data = list(reader)

    # Delete the selected row
    del data[index]

    # Write the updated data back to the CSV file
    with open("data.csv", "w", newline="") as file:
        writer = csv.writer(file)
        writer.writerows(data)

    display_window.destroy()
    display_data()

window = tk.Tk()
# ...
```

3. Save the file.

To run the script and delete rows from the CSV file:

1. Open the console or terminal.
2. Navigate to the directory where the `script.py` file is located.
3. Enter the following command:

```
python3 script.py
```

4. Use the GUI to enter height, weight, and cardiac event data and save it.
5. Click the "Records" button.
6. A new window titled "Records" will open, showing all the rows from the `data.csv` file, with a "Delete" button next to each row.
7. Click the "Delete" button next to the row you want to delete.
8. The selected row will be removed from the `data.csv` file, and the window will update to reflect the changes.

Well done! You've added the functionality to delete rows from the CSV file using the GUI.

Let's proceed to the next step.

Step 9: Adding Raise Error Tests to Make the Code More Stable

In this step, we'll enhance the code by adding error handling and raise error tests to make it more stable.

1. Open the `script.py` file.

2. Replace the existing `delete_row` function with the following code:

```python
def calculate_bmi(height=0, weight=0):
    if height == 0:
    # If height and weight are not provided, retrieve them from the GUI entry fields
        height = float(height_entry.get())
        weight = float(weight_entry.get())
    # Calculate BMI
    if height <= 0 or weight <= 0:
        raise ValueError("Height and weight must be positive values.")
    bmi = weight / (height ** 2)
    result_label.config(text=f"Patient's BMI is: {bmi:.2f}")

    cardiac_event = cardiac_event_var.get()
    logger.info(f"Height: {height} m, Weight: {weight} kg")
    logger.info(f"BMI: {bmi:.2f}, Cardiac Event: {cardiac_event}")

    # Save patient data
    with open("data.csv", "a", newline="") as file:
        writer = csv.writer(file)
        writer.writerow([height, weight, bmi, cardiac_event])
    return bmi
# ...
```

3. Replace the existing `delete_row` function with the following code:

```python
def delete_row(index, display_window):
    # Read data from the CSV file
    with open("data.csv", "r") as file:
        reader = csv.reader(file)
        data = list(reader)

    # Delete the selected row
    if index < 0 or index >= len(data):
        raise IndexError("Invalid row index.")
    del data[index]

    # Write the updated data back to the CSV file
    with open("data.csv", "w", newline="") as file:
        writer = csv.writer(file)
        writer.writerows(data)

    display_window.destroy()
    display_data()
# ...
```

4. Save the file.

To run the script and test error handling:

1. Open the console or terminal.
2. Navigate to the directory where the `script.py` file is located.
3. Enter the following command:

```
python3 script.py
```

4. Use the GUI to enter height, weight, and cardiac event data and save it.
5. Click the "Calculate BMI" button.
6. If you enter invalid height or weight values (e.g., negative values), an error message will be displayed in the console.

Great job! You've added error handling and raise error tests to improve the stability of the code.

Let's move on to the next step.

Step 10: GUI Can Visualize Data Points

In this step, we'll add features that will allow us to visualize the CSV file's data points and color them in accordance with cardiac events.

1. Install the required dependencies:

```
pip3 install matplotlib
```

2. Open the `script.py` file.
3. Add the following `import` statement at the beginning of the file, after the other `import` statements:

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

4. Add the following code after the `display_button.pack()` line:

```
# ...
display_button.pack()

visualize_button = tk.Button(window, text="Visualize Data",
command=visualize_data)
visualize_button.pack()

# ...
```

5. Add following `visualize_data` function:

```
def visualize_data():
    # Read data from the CSV file
    with open("data.csv", "r") as file:
        reader = csv.reader(file)
        data = list(reader)

    # Extract the necessary data for visualization
    heights = []
    weights = []
    cardiac_events = []
    bmis = []

    for row in data:
        if len(row) != 4:
            raise ValueError("Invalid data format in CSV file.")
        height = float(row[0])
        weight = float(row[1])
        bmi = float(row[2])
        cardiac_event = row[3]

        heights.append(height)
        weights.append(weight)
        cardiac_events.append(cardiac_event)
        bmis.append(bmi)

    # Group the BMI values by CardiacEvent
    grouped_data = {}
    for event, bmi_value, height_value, weight_value in zip(cardiac_events,
bmis, heights, weights):
        if event not in grouped_data:
            grouped_data[event] = {'BMI': [], 'Height': [], 'Weight': []}
        grouped_data[event]['BMI'].append(bmi_value)
        grouped_data[event]['Height'].append(height_value)
        grouped_data[event]['Weight'].append(weight_value)

    # Create a figure with two subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

    # Boxplot for BMI
    ax1.boxplot([grouped_data['No']['BMI'], grouped_data['Yes']['BMI']],
labels=['No', 'Yes'])
    ax1.set_xlabel('Cardiac Event')
    ax1.set_ylabel('BMI [kg/m²]')
    ax1.set_title('Body mass index (BMI) Distribution by Cardiac Event')

    # Scatterplot for height and weight
    ax2.scatter(grouped_data['No']['Height'], grouped_data['No']['Weight'],
c='blue', label='No')
    ax2.scatter(grouped_data['Yes']['Height'], grouped_data['Yes']
['Weight'], c='red', label='Yes')
    ax2.set_xlabel('Height [m]')
    ax2.set_ylabel('Weight [kg]')
    ax2.set_title('Relationship between Height and Weight')
    ax2.legend()

    # Adjust spacing between subplots
    plt.tight_layout()

    # Show the plot
    plt.show()

window = tk.Tk()
# ...
```

4. Save the file.

To run the script and visualize the data:

1. Open the console or terminal.
2. Navigate to the directory where the `script.py` file is located.
3. Enter the following command:

```
python3 script.py
```

4. Use the GUI to enter height, weight, and cardiac event data and save it.
5. Click the "Visualize Data" button.
6. A figure with two subplots will appear, visualizing the BMI distribution by cardiac event and the relationship between height and weight.

Fantastic work! You've implemented data visualization functionality in your script.

Let's move on to the final step.

Step 11: Adding a Button to Generate a Random Patient

In this final step, we'll add functionality to generate a random patient with random height and weight values.

1. Install the required dependency:

```
pip3 install requests
```

2. Open the `script.py` file.
3. Add the following `import` statement at the beginning of the file, after the other `import` statements:

```
import requests
import random
```

4. Add the following code after the `display_button.pack()` line:

```
# ...
visualize_button.pack()

randompatient_button = tk.Button(window, text="Generate one Random
Patient", command=create_random_patient)
randompatient_button.pack()

# ...
```

5. Add following `create_random_patient` function:

```
def create_random_patient():
    # Generate random height and weight values
    height = float(get_random_height())
    weight = float(get_random_weight())
    height_entry.delete(0, tk.END)
    height_entry.insert(0, height)
    weight_entry.delete(0, tk.END)
    weight_entry.insert(0, weight)

    logger.info(f"Generated random patient: Height: {height} m, Weight:
{weight} kg")

    # Calculate BMI for the generated patient
    bmi = weight / (height ** 2)
    result_label.config(text=f"Patient's BMI is: {bmi:.2f}")

    # Determine the cardiac event for the generated BMI
    event = get_cardiac_event_distribution(bmi)
    # Set the cardiac event value in the checkbox
    cardiac_event_var.set(event)

    logger.info(f"Height: {height} m, Weight: {weight} kg")
    logger.info(f"BMI: {bmi:.2f}, Cardiac Event: {event}")

    # Save the generated patient data
    with open("data.csv", "a", newline="") as file:
        writer = csv.writer(file)
        writer.writerow([height, weight, bmi, event])


def get_random_height():
    # Retrieve a random height value from an API
    response =
requests.get("http://www.randomnumberapi.com/api/v1.0/random", params=
{"min": 130, "max": 210, "count": 1})
    if response.status_code != 200:
        raise requests.RequestException("Failed to retrieve random height.")
    data = response.json()
    if not data:
        raise ValueError("Empty response received for random height.")
    height = float(data[0])
    if height < 130:
        raise ValueError("Invalid random height value received.")
    logger.info(f"Random height generated: {height} cm")
    return height / 100

def get_random_weight():
    # Retrieve a random weight value from an API
    response = requests.get("http://www.randomnumberapi.com/api/v1.0/random", params={"min": 40, "max": 140, "count": 1})
    if response.status_code != 200:
        raise requests.RequestException("Failed to retrieve random weight.")
    data = response.json()
    if not data:
        raise ValueError("Empty response received for random weight.")
    weight = data[0]
    if weight < 40:
        raise ValueError("Invalid random weight value received.")
    logger.info(f"Random weight generated: {weight} kg")
    return weight

def get_cardiac_event_distribution(bmi):
    if bmi < 9 or bmi > 83:
        raise ValueError("BMI value should be between 9 and 83.")
    if bmi <= 22:
        return "No"
    if bmi <= 38:
        risk = (bmi - 22) / (38 - 22)
        return 'Yes' if random.random() < risk else 'No'
    return "Yes"

window = tk.Tk()
# ...
```

4. Save the file.

To run the script and generate a random patient:

1. Open the console or terminal.
2. Navigate to the directory where the `script.py` file is located.
3. Enter the following command:

```
python3 script.py
```

4. Click the "Generate one Random Patient" button.
5. The height and weight fields will be filled with random values.
6. The BMI will be calculated, and the cardiac event checkbox will be set accordingly.
7. Click the "Calculate BMI" button to display the calculated BMI and save the patient data.

Amazing work! You've added the functionality to generate a random patient with random height and weight values.

Congratulations! You have successfully created a step-by-step guide to learning programming a Python script that can be launched from the console. The final code includes features such as a GUI interface, BMI calculation, data logging, data storage in CSV format, data visualization, error handling, and random patient generation.

Feel free to explore and enhance the script further to suit your needs and expand your Python programming skills. Happy coding!