Program 1. Coins

Years ago, Australia used the same currency units as did Great Britain. These units were called pounds, shillings, and pence. There were 12 pence in a shilling, and 20 shillings in a pound. Your user will input some number of pence, and your program will determine how to give them back the same amount of money in pounds, shillings, and pence. Think of it like taking change to the bank and getting it converted. All variables should be declared as integer. You will need to use integer division and modulus in this program.

Ans: Explanation

1. tkinter module is imported into the code for it is the standard GUI toolkit in Python.

2. Declaring constants: Constants like PENCE_PER_SHILLING and SHILLINGS_PER_POUND are declared to represent conversion rates between different currency units.

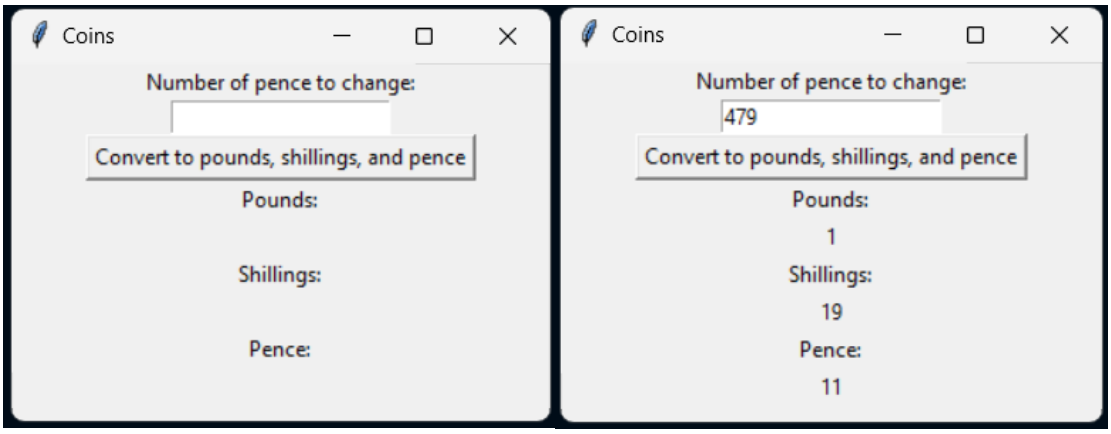
3. Converting Function: A function called convert_pence() is used to convert pence inputted by the user into pounds, shillings and left over pence. This function grabs an input from a user, makes some calculations basing on conversion rates, and modifies accordingly GUI.

4. Building Interface Elements: Different interface elements such as labels, entry fields, and buttons are created to enable interaction of the user with the program; they consist of instruction display labels; output display label; an entry field with which users can insert number of pennies; a button that will start conversion process.

5. Main Window Creation: The main window of the GUI application is created using `tk.Tk()`, and its title is set to "Coins".

6. Event Loop: On calling ‘mainloop()’, event loop begins which listens continuously for user inputs and updates GUIs where necessary.

The script employs tkinter to design simple graphical user interface (GUI) for transforming given quantity of pence into pounds shillings pence as per historic Australian or British currency system



```python
import tkinter as tk

# Define constants
PENCE_PER_SHILLING = 12
SHILLINGS_PER_POUND = 20

# Function to convert pence to pounds, shillings, and pence
def convert_pence():
    # Get the input pence from the entry field
    pence_input = int(pence_entry.get())

    # Calculate the pounds, shillings, and remaining pence
    pounds = pence_input // (PENCE_PER_SHILLING * SHILLINGS_PER_POUND)
    remaining_pence = pence_input % (PENCE_PER_SHILLING * SHILLINGS_PER_POUND)
    shillings = remaining_pence // PENCE_PER_SHILLING
    pence = remaining_pence % PENCE_PER_SHILLING

    # Update the output fields with the result
    pounds_output.config(text=str(pounds))
    shillings_output.config(text=str(shillings))
    pence_output.config(text=str(pence))

# Create the main window
root = tk.Tk()
root.title("Coins")

# Create the input field for pence
pence_label = tk.Label(root, text="Number of pence to change:")
pence_label.pack()
pence_entry = tk.Entry(root)
pence_entry.pack()

# Create the "Convert to pounds, shillings, and pence" button
convert_button = tk.Button(root, text="Convert to pounds, shillings, and pence", command=convert_pence)
convert_button.pack()

# Create the output fields for pounds, shillings, and pence
pounds_label = tk.Label(root, text="Pounds:")
pounds_label.pack()
pounds_output = tk.Label(root, text="")
pounds_output.pack()

shillings_label = tk.Label(root, text="Shillings:")
shillings_label.pack()
shillings_output = tk.Label(root, text="")
shillings_output.pack()

pence_label = tk.Label(root, text="Pence:")
pence_label.pack()
pence_output = tk.Label(root, text="")
pence_output.pack()

# Start the main event loop
root.mainloop()
```

Program 2. Ounces

Write a program which asks the user to enter the weight of an item in pounds and ounces and the price of the item. Ounces and Pounds will be stored as integers. You are to write a program which returns the cost per ounce for the item using currency format with two decimal places. There are 16 ounces in a pound.

Ans: Explanation

1. tkinter Module: This is the module that's imported to create the GUI application.

2. Creating the Main Window: It creates main window of the application through `tk.Tk()` and set its title as "Ounces".

3. Defining Functions:

- `calculate_price_per_ounce()`: This function calculates price per ounce with respect to values obtained from user inputs; it also handles exceptions like missing or invalid values such as negative data or non-numerical entries.

4. Creating Input Fields:

- Labels and entry fields are created for inputting the price of an item, weight of an item in pounds, and weight of an item in ounces.

5. Creating the Calculate Button:

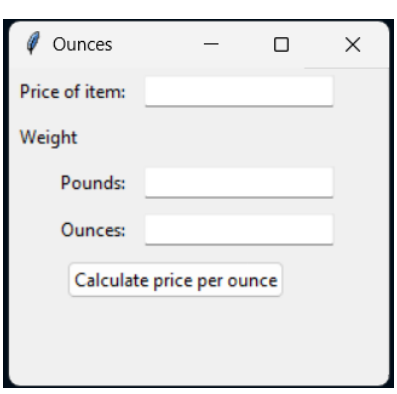
- A button called "Calculate price per ounce" is made by it that on clicking activates the `calculate_price_per_ounce()` function which computes the cost per ounce.

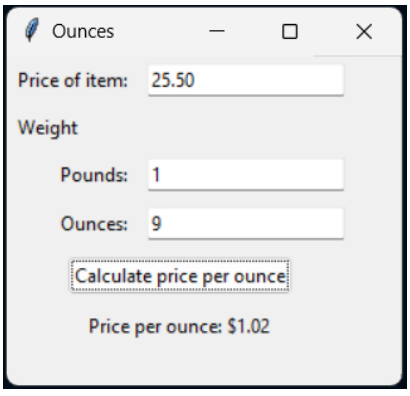
6. Creating the Result Label:

- Result label is created that will display a calculated result i.e., Price per Ounce which initially remains empty before loading it with results after calculation process.

7. Running the Main Event Loop:

- when running this code, there is a start up of main event loop (`mainloop()`) which listens for inputs in form of a user's actions then responds to them accordingly by updating GUIs appropriately.





```python
import tkinter as tk
from tkinter import ttk

# Create the main window
root = tk.Tk()
root.title("Ounces")

# Create functions
def calculate_price_per_ounce():
    """
    Calculate the price per ounce based on the input values.
    Handles error cases where input values are missing or invalid.
    """
    try:
        price = float(price_entry.get())
        pounds = int(pounds_entry.get())
        ounces = int(ounces_entry.get())
    except ValueError:
        result_label.config(text="Invalid input. Please enter valid values.")
        return

    if price <= 0 or (pounds == 0 and ounces == 0):
        result_label.config(text="Invalid input. Price and weight must be positive.")
        return

    total_ounces = pounds * 16 + ounces
    price_per_ounce = price / total_ounces
    result_label.config(text=f"Price per ounce: ${price_per_ounce:.2f}")

# Create input fields
price_label = ttk.Label(root, text="Price of item:")
price_label.grid(row=0, column=0, padx=5, pady=5, sticky="w")
price_entry = ttk.Entry(root)
price_entry.grid(row=0, column=1, padx=5, pady=5)

weight_label = ttk.Label(root, text="Weight")
weight_label.grid(row=1, column=0, padx=5, pady=5, sticky="w")

pounds_label = ttk.Label(root, text="Pounds:")
pounds_label.grid(row=2, column=0, padx=5, pady=5, sticky="e")
pounds_entry = ttk.Entry(root)
pounds_entry.grid(row=2, column=1, padx=5, pady=5)

ounces_label = ttk.Label(root, text="Ounces:")
ounces_label.grid(row=3, column=0, padx=5, pady=5, sticky="e")
ounces_entry = ttk.Entry(root)
ounces_entry.grid(row=3, column=1, padx=5, pady=5)

# Create calculate button
calculate_button = ttk.Button(root, text="Calculate price per ounce", command=calculate_price_per_ounce)
calculate_button.grid(row=4, column=0, columnspan=2, padx=5, pady=5)

# Create result label
result_label = ttk.Label(root, text="")
result_label.grid(row=5, column=0, columnspan=2, padx=5, pady=5)

# Run the main event loop
root.mainloop()
```

Program 3. Averages

Write a program that requests the user for three decimal test scores between 0 and 100. Your program will then find the average of the two **highest** scores. You are to use three **separate IF statements** to determine which score is the smallest. For example, the first IF statement would check if score 1 is less than score 2 and score 1 is less than score 3 That would mean that the smallest score would be score 1, and the average would be calculated using scores 2 and 3. The other IF statements would follow the same process. The average is to be formatted to one decimal place.

Ans: Explanation

1. Creating the Main Window: `tk.Tk()` is used to create a window where the Graphical User Interface will be displayed and its title is set as “Averages”.

2. Defining Functions:

- `calculate_average()` This function calculates the mean of the two highest scores entered by the user, handles exceptional cases where input values are not provided or are not valid; these can include non numeric inputs or scores that are not in between 0 to 100.

3. Creating Input Fields:

- There should be labels and entry fields for three test scores with each corresponding to one test score.

4. Creating the Calculate Button:

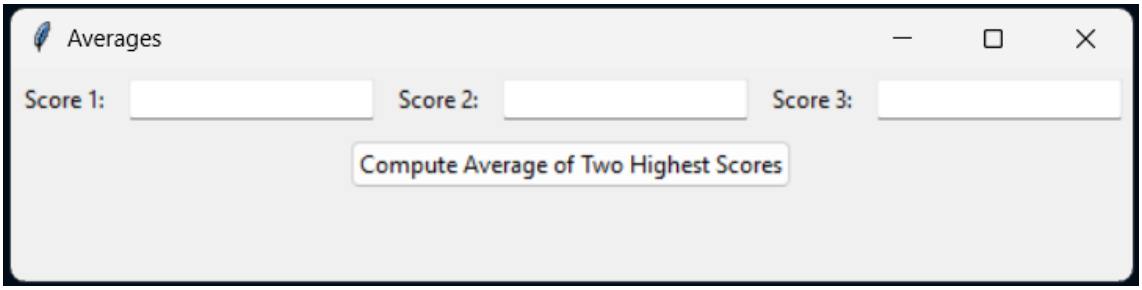
- The button labeled “Compute Average of Two Highest Scores” is made ready. On clicking it, calculate average function takes over formulating sum of two high scores and their divisions.

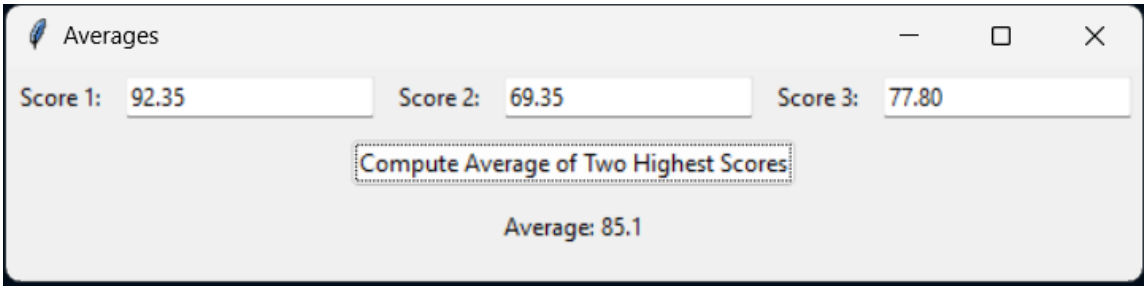
5. Creating the Result Label:

- Create a label that shows calculation output which is average highest numbers obtained in the evaluation process; initially it has no content but it gets filled with result after computation.

6. Running the Main Event Loop:

- Again, start main event loop (’mainloop()’) which waits for user actions and updates GUI accordingly.





```python
import tkinter as tk
from tkinter import ttk

# Create the main window
root = tk.Tk()
root.title("Averages")

# Create functions
def calculate_average():
    """
    Calculate the average of the two highest scores from the input values.
    Handles error cases where input values are missing or invalid.
    """
    try:
        score1 = float(score1_entry.get())
        score2 = float(score2_entry.get())
        score3 = float(score3_entry.get())
    except ValueError:
        result_label.config(text="Invalid input. Please enter valid scores.")
        return

    if score1 < 0 or score1 > 100 or score2 < 0 or score2 > 100 or score3 < 0 or score3 > 100:
        result_label.config(text="Invalid input. Scores must be between 0 and 100.")
        return

    # Find the smallest score using three separate if statements
    smallest_score = score1
    if score2 < score1 and score2 < score3:
        smallest_score = score2
    elif score3 < score1 and score3 < score2:
        smallest_score = score3

    # Calculate the average of the two highest scores
    highest_scores = [score1, score2, score3]
    highest_scores.remove(smallest_score)
    average = sum(highest_scores) / 2

    result_label.config(text=f"Average: {average:.1f}")

# Create input fields
score1_label = ttk.Label(root, text="Score 1:")
score1_label.grid(row=0, column=0, padx=5, pady=5, sticky="w")
score1_entry = ttk.Entry(root)
score1_entry.grid(row=0, column=1, padx=5, pady=5)

score2_label = ttk.Label(root, text="Score 2:")
score2_label.grid(row=0, column=2, padx=5, pady=5, sticky="w")
score2_entry = ttk.Entry(root)
score2_entry.grid(row=0, column=3, padx=5, pady=5)

score3_label = ttk.Label(root, text="Score 3:")
score3_label.grid(row=0, column=4, padx=5, pady=5, sticky="w")
score3_entry = ttk.Entry(root)
score3_entry.grid(row=0, column=5, padx=5, pady=5)

# Create calculate button
calculate_button = ttk.Button(root, text="Compute Average of Two Highest Scores", command=calculate_average)
calculate_button.grid(row=1, column=0, columnspan=6, padx=5, pady=5)

# Create result label
result_label = ttk.Label(root, text="")
result_label.grid(row=2, column=0, columnspan=6, padx=5, pady=5)

# Run the main event loop
root.mainloop()
```

Program 4. Honors

Write a program which allows the user to enter a GPA between 2.0 and 4.0. You are to use an IF statement with ElseIfs for Visual Basic, and IF statement with elifs for Python for the following:

GPA greater than or equal to 3.9: " summa cum laude."

GPA greater than or equal to 3.6 and less than 3.9: " magna cum laude."

GPA greater than or equal to 3.3 and less than 3.6: " cum laude."

GPA greater than or equal to 2.0 and less than 3.3: "."

You are to store the honors in a string variable called strHonors. This will be used to form part of your output. If the student has a GPA greater than or equal to 2.0 but less than 3.3, their output will be "You graduated."

You are to use an IF statement in your output area as well. If the GPA is entered as being less than 2.0 or greater than 4.0, the output should read "Incorrect input. Try again".

Ans: Explanation

1. Determining Honours Based on GPA

- The program defines a function `determine_honors()` to determine honors based on the GPA entered by the user.
- It first converts the GPA input from the entry field to a float.
- Then, it uses an if-elif-else statement to check various conditions:
- If the GPA is less than 2.0 or greater than 4.0, it displays "Incorrect Input. Try again."
- If the GPA is greater than or equal to 3.9, it displays "You graduated summa cum laude."
- If the GPA is greater than or equal to 3.6, it displays "You graduated magna cum laude."
- If the GPA is greater than or equal to 3.3, it displays "You graduated cum laude."
- If the GPA is greater than or equal to 2.0 (and less than 3.3), it displays "You graduated."
- The function 'determine_honors' is used to allow the user to enter their marks and determine if they are eligible for an honours award based on this mark.

2. Building the main window

- The GUI application builds a main window using `tk.Tk()` and gives it the title 'Honors'.

3. Building labels and entry field

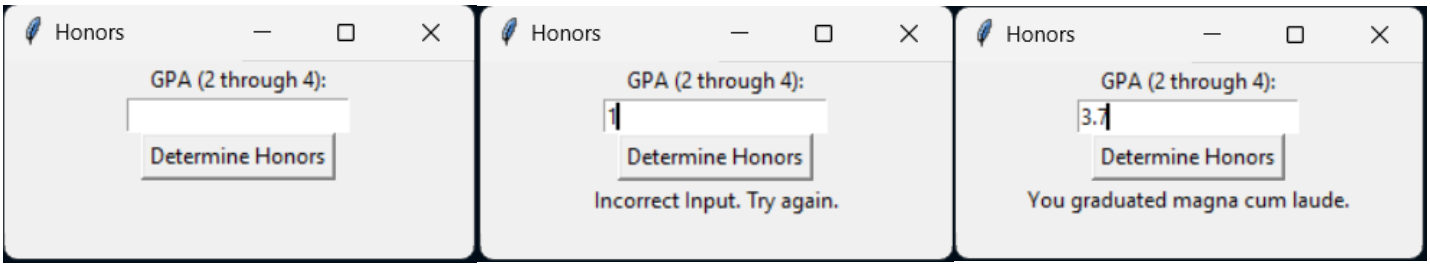
- A label is built which allows one to input their GPA.

4. Creating a "Determine Honours" button

- One creates a button labelled as "Determine Honors". After clicking, it calculates and displays the honours on the basis of student's GPA by calling a function of `determine_honors()`.

5. Displaying results

- A label widget is created in order to display results which are honors that were determined through using marks obtained from entering GPAs




```python
import tkinter as tk

# Function to determine honors based on GPA
def determine_honors():
    gpa = float(gpa_entry.get())
    if gpa < 2.0 or gpa > 4.0:
        result_label.config(text="Incorrect Input. Try again.")
    elif gpa >= 3.9:
        result_label.config(text="You graduated summa cum laude.")
    elif gpa >= 3.6:
        result_label.config(text="You graduated magna cum laude.")
    elif gpa >= 3.3:
        result_label.config(text="You graduated cum laude.")
    elif gpa >= 2.0:
        result_label.config(text="You graduated.")

# Create the main window
root = tk.Tk()
root.title("Honors")

# Create labels and entry field
gpa_label = tk.Label(root, text="GPA (2 through 4):")
gpa_label.pack()

gpa_entry = tk.Entry(root)
gpa_entry.pack()

# Create the Determine Honors button
determine_button = tk.Button(root, text="Determine Honors", command=determine_honors)
determine_button.pack()

# Create a label to display the result
result_label = tk.Label(root, text="")
result_label.pack()

# Run the main event loop
root.mainloop()
```

Program 5. Forecast

The color of the beacon light atop Boston's old John Hancock building forecasts the weather according to the following rhyme:

Steady blue, clear view
Flashing blue, clouds due
Steady red, rain ahead
Flashing red, snow instead

Write a program which requests a color (Blue or Red) and a mode (Steady or Flashing) as input and displays the weather forecast. You are to use an IF statement with such as If color = "B" and mode = "S" Then … in Visual Basic. The same idea will work in Python. Use the ToUpper function in Visual Basic and the upper() function in Python to correct for the user entering lower case letters. In Python, at the end of each input statement add a period and upper(). Here is an example for Python:

Status = input("Enter V for veteran or O for other: ").upper()

Ans: Explanation

1. Determining Forecast Based on Input:

- The program defines a function `determine_forecast()` to determine the weather forecast based on the color and mode entered by the user.
- It converts the input color and mode to uppercase using `.upper()` to ensure consistency.
- It then uses if-elif-else statements to check various conditions:
- If the color is "B" (blue) and the mode is "S" (steady), it displays "Steady blue, clear view."
- If the color is "B" and the mode is "F" (flashing), it displays "Flashing blue, clouds due."
- If the color is "R" (red) and the mode is "S", it displays "Steady red, rain ahead."
- If the color is "R" and the mode is "F", it displays "Flashing red, snow instead."
- If the input does not match any of these conditions, it displays "Invalid input. Please enter B or R for color and S or F for mode."

2. Creating the Main Window: The main window for the GUI application is created using `tk.Tk()` and titled "Forecast".

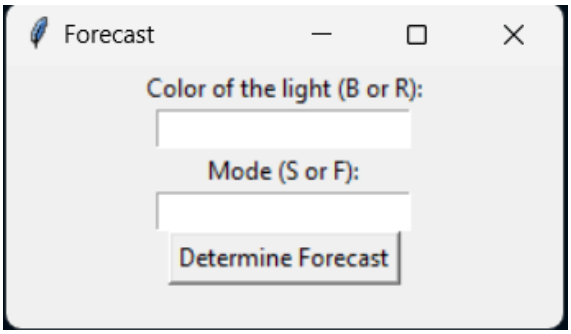
3. Creating Labels and Entry Fields: Labels and entry fields are created to input the color of the light (B or R) and the mode (S or F).

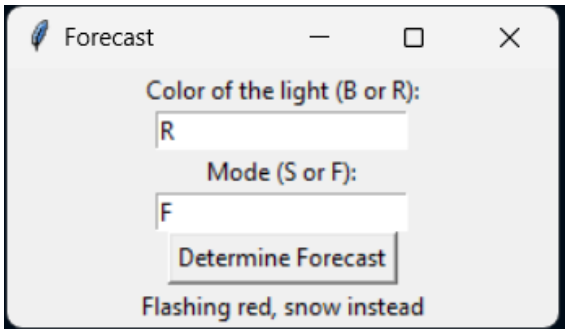
4. Creating the "Determine Forecast" Button: A button labeled "Determine Forecast" is created. When clicked, it triggers the `determine_forecast()` function to calculate and display the weather forecast.

5. Displaying the Forecast: A label is created to display the forecast, i.e., the weather forecast determined based on the user's input.

6. Running the Main Event Loop: The main event loop (`mainloop()`) is started, which listens for user inputs and updates the GUI accordingly.

Overall, this program provides a simple GUI interface for users to input the color and mode of the beacon light atop Boston's old John Hancock building and determine the weather forecast based on the provided rhyme.






```python
import tkinter as tk

# Function to determine the forecast
def determine_forecast():
    color = color_entry.get().upper()
    mode = mode_entry.get().upper()

    if color == "B" and mode == "S":
        forecast_label.config(text="Steady blue, clear view")
    elif color == "B" and mode == "F":
        forecast_label.config(text="Flashing blue, clouds due")
    elif color == "R" and mode == "S":
        forecast_label.config(text="Steady red, rain ahead")
    elif color == "R" and mode == "F":
        forecast_label.config(text="Flashing red, snow instead")
    else:
        forecast_label.config(text="Invalid input. Please enter B or R for color and S or F for mode.")

# Create the main window
root = tk.Tk()
root.title("Forecast")

# Create labels and entry fields
color_label = tk.Label(root, text="Color of the light (B or R):")
color_label.pack()

color_entry = tk.Entry(root)
color_entry.pack()

mode_label = tk.Label(root, text="Mode (S or F):")
mode_label.pack()

mode_entry = tk.Entry(root)
mode_entry.pack()

# Create the Determine Forecast button
determine_button = tk.Button(root, text="Determine Forecast", command=determine_forecast)
determine_button.pack()

# Create a label to display the forecast
forecast_label = tk.Label(root, text="")
forecast_label.pack()

# Run the main event loop
root.mainloop()
```