CODE LAB-II

Programming Skills Portfolio

Asif Yousaf

Table Of Contents

Chapter 1 - Python Refresher

Chapter 2 - Graphical User Interface

Chapter 3 - Graphical User Interface(Cont.)

Chapter 4 - File Handling and Regular Expressions

Chapter 5 - Object Oriented Programming

Chapter 6 - Python Standard Library Modules

Chapter 1 - Python Refresher

Exercises, Bonus And Further

Chapter 1 Ex 1 – User Input Output

This code demonstrates a simple program that interacts with the user, capturing their name and age, and providing personalized messages based on the input. Below, you'll find the Python code, its corresponding output, and additional comments explaining each step.

Code

```python
# Chapter 1 Exercise 1 - User Input Output

# Printing a Welcome msg.
print("Hello, user!")

# Asking the user to enter their name and convert it to title case.
user_name = str(input("what is your name?\n"))
name_title = user_name.title()

# Asking the user to enter their age.
user_age = int(input("what is your age?\n"))

# Printing a message using User name.
print(f"It is good to meet you, {name_title}")

# Calculating and printing the length of the user name
name_len = len(name_title)
print(f"The Length of your name is:\n{name_len}")

# Calculating and printing the age after one year
age_next_year = user_age + 1
print(f"You will be {age_next_year} in a year.")
```

Output

```
Hello, user!
what is your name?
Asif
what is your age?
19
It is good to meet you, Asif
The Length of your name is:
4
You will be 20 in a year.
```

Chapter 1 Ex 2 - Math's

This code demonstrates a straightforward mathematical program that interacts with the user. It prompts the user to input two integer numbers, performs basic arithmetic operations (addition, subtraction, multiplication, division, and modulo), and outputs the results. Below, you'll find the Python code, its corresponding output, and additional comments explaining each step.

Code

```python
# Chapter 1 Exercise 2 - Maths

# Asking the User two int numbers
User_input_1 = int(input("Enter A Number = "))
User_input_2 = int(input("Enter A Number = "))

# Performing basic arithmetic operations
Sum = User_input_1 + User_input_2  #Addition of the two numbers
subtraction = User_input_1 - User_input_2  #Subtraaction of the two numbers
Product = User_input_1 * User_input_2  #Product of the two numbers
Quotient = User_input_1 / User_input_2  #Quotient of the two numbers
Remainder = User_input_1 % User_input_2  #Remainder of the two numbers

#Printing to console
print(f"The Sum of the two numbers is {Sum}")
print(f"The Difference between the two numbers is {subtraction}")
print(f"The Product of the two numbers is {Product}")
print(f"The Quotient od two number is {Quotient}")
print(f"The Remainder of these two number is {Remainder}")
```

Output

```
Enter A Number = 22
Enter A Number = 33
The Sum of the two numbers is 55
The Difference between the two numbers is -11
The Product of the two numbers is 726
The Quotient od two number is 0.6666666666666666
The Remainder of these two number is 22
```

Chapter 1 Ex 3 - Is it Triangle

This code prompts the user to enter the lengths of three sides of a triangle. It then checks the triangle inequality to determine whether the provided side lengths form a valid triangle. If valid, the code proceeds to classify the triangle as Equilateral, Isosceles, or Scalene based on the side lengths. Below, you'll find the Python code, its corresponding output, and additional comments explaining each step.

Code

```python
# Chapter 1 Exercise 3 - Is it Triangle

# Asking user for lengths of the three sides
side_a = float(input("Enter the length of the first side: "))
side_b = float(input("Enter the length of the second side: "))
side_c = float(input("Enter the length of the third side: "))

# Checking the triangle inequality to determine if its a valid triangle
if side_a + side_b >= side_c and side_a + side_c >= side_b and side_b + side_c >= side_a:
    print("It is a valid triangle")
else:
    print("It is not a valid triangle")

# Checking all sides if equal it is Equilateral
if side_a == side_b == side_c:
    print(" The triangle is Equilateral")

    # Checking all sides if any one is equal it is Isosceles
elif side_a == side_b or side_a == side_c or side_b == side_c:
    print("The triangle is Isosceles")

    # if none of the side is equal then it is scalene
else:
    print("The triangle is scalene")
```

Output

```
Enter the length of the first side: 15
Enter the length of the second side: 25
Enter the length of the third side: 34
It is a valid triangle
The triangle is scalene
```

Chapter 1 Ex 4 - Largest Number

This code prompts the user to enter three numbers, compares them, and identifies the largest among them. The Python code, along with its corresponding output and additional comments explaining each step, is presented below:

Code

```python
# Chapter 1 Exercise 4 - Largest Number

# Prompt the user to enter three numbers and store them in variables
num1 = int(input("Enter a number: "))
num2 = int(input("Enter a number: "))
num3 = int(input("Enter a number: "))

# Checking which number is the largest among the three using if-elif-else statements
if num1 > num2 and num1 > num3:
    print(f"The Number {num1} is the biggest")  # Print the result if num1 is the largest

elif num2 > num1 and num2 > num3:
    print(f"The Number {num2} is the biggest")  # Print the result if num2 is the largest

elif num3 > num1 and num3 > num2:
    print(f"The Number {num3} is the biggest")  # Print the result if num3 is the largest

else:
    print("Wrong input")  # Print an error message if the inputs are not valid

# End marker
```

Output

```
Enter a number: 10
Enter a number: 15
Enter a number: 25
The Number 25 is the biggest
```

Chapter 1 Ex 5 - Continue

This code presents a simple program utilizing a while loop that interacts with the user, prompting them to continue or exit by entering 'Y' or any other key, respectively. The loop keeps track of how many times it executes, and the final count is printed as output. Below, you'll find the Python code, its corresponding output, and additional comments explaining each step.

Code

```python
# Chapter 1 Exercise 5 - Continue

# Counter for loop
count = 0

# using while loop
while True:
    # Ask the user if they want to continue
    user_input = input("Do you want to continue? Enter 'Y' for yes, any other key to exit: ").upper()

    # Check if the user wants to continue
    if user_input == 'Y':
        # Adding the loop count
        count += 1
    else:
        # Exiting the loop if user enters anything else
        break

# Output the number of times the loop was executed
print(f"The loop was executed {count} times.")

# End marker
```

Output

```
Do you want to continue? Enter 'Y' for yes, any other key to exit: y
Do you want to continue? Enter 'Y' for yes, any other key to exit: y
Do you want to continue? Enter 'Y' for yes, any other key to exit: y
Do you want to continue? Enter 'Y' for yes, any other key to exit: y
Do you want to continue? Enter 'Y' for yes, any other key to exit: y
Do you want to continue? Enter 'Y' for yes, any other key to exit: n
The loop was executed 5 times.
```

Chapter 1 Ex 6 - Fizz Buzz

This code demonstrates the classic Fizz Buzz problem. It iterates through numbers from 1 to 100 and prints "Fizz" for multiples of 3, "Buzz" for multiples of 5, "Fizz Buzz" for numbers that are multiples of both 3 and 5, and the number itself otherwise. Below, you'll find the Python code, its corresponding output, and additional comments explaining each step.

Code

```python
# Chapter 1 Exercise 6 - FizzBuzz

#  Loop to start numbers from 1 to 100
for number in range(1, 101):

    # Checking if the number is a multiple of both 3 and 5
    if number % 3 == 0 and number % 5 == 0:
        print("FizzBuzz")

    # Checking if the number is a multiple of 3
    elif number % 3 == 0:
        print("Fizz")

    # Checking if the number is a multiple of 3
    elif number % 5 == 0:
        print("Buzz")

    # if not the multiple of '3' and '5' just prints the number
    else:
        print(number)

# End marker
```

Output

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
Fizz
22
23
Fizz
Buzz
26
Fizz
28
29
FizzBuzz
```

```
FizzBuzz
31
32
Fizz
34
Buzz
Fizz
37
38
Fizz
Buzz
41
Fizz
43
44
FizzBuzz
46
47
Fizz
49
Buzz
Fizz
52
53
Fizz
Buzz
56
Fizz
58
59
FizzBuzz
```

```
61
62
Fizz
64
Buzz
Fizz
67
68
Fizz
Buzz
71
Fizz
73
74
FizzBuzz
76
77
Fizz
79
Buzz
Fizz
82
83
Fizz
Buzz
86
Fizz
88
89
FizzBuzz
```

```
91
92
Fizz
94
Buzz
Fizz
97
98
Fizz
Buzz
```

Chapter 1 Ex 7 - Even Numbers

This code generates and prints even numbers between 1 and 100. The program uses a loop to iterate through numbers from 1 to 100. Odd numbers are skipped using the 'continue' statement, and only even numbers are printed. Finally, an end marker indicates the completion of the program.

Code

```python
# Chapter 1 Exercise 7 - Even Numbers

# Loop for numbers from 1 to 100
for number in range(1, 101):
    # Skiping odd numbers and continue
    if number % 2:
        continue
    # Print the even number
    print(number)

# End marker
```

Output

```
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
64
66
68
70
72
74
76
78
80
82
84
86
88
90
92
94
96
98
100
```

Chapter 1 Ex 8 - Print Pattern

This code presents a simple Python program that generates a numeric pattern. The program utilizes nested loops to control the number of rows and columns, creating a pattern where each row displays sequential numbers. The Python code, its corresponding output, and additional comments detailing each step are provided below.

Code

```python
# Chapter 1 Exercise 8 - Print Pattern

# Loop through a range to control the number of rows in the pattern
for i in range(1, 5):
    # Loop through a range to control the number of columns in each row
    for j in range(1, i + 1):
        print(j, end=" ")  # Print the value of 'j' followed by a space, without moving to a new line
    print()  # Move to a new line after printing each row

    # End marker
```

Output

```
1
1 2
1 2 3
1 2 3 4
```

Chapter 1 Ex 9 - Integer List

This code demonstrates a program that works with an integer list. It creates a list, outputs its original form, finds the highest and lowest values, sorts the list in ascending and descending order, appends two elements, and prints the final list.

Code

```python
# Chapter 1 Exercise 9 - Integer List

# Creating an integer list with 10 values
integer_list = [5, 79, 8, 1, 7, 3, 12, 4, 9, 6]

# Output the list using a for loop
print("Original List:")
for i in integer_list:
    print(i, end=" ")
print()

# the highest and lowest value
print("Highest Value is:", max(integer_list))
print("Lowest Value is:", min(integer_list))

# Sorting the elements in ascending order
integer_list.sort()
print("Sorted List Ascending:", integer_list)

# Sorting the elements in descending order
integer_list.sort(reverse=True)
print("Sorted List Descending:", integer_list)

# Appending two elements
integer_list.append(91)
integer_list.append(0)

# Printin the list after appending
print("List after Appending:", integer_list)

# End marker
```

Output

```
Highest Value is: 79
Lowest Value is: 1
Sorted List Ascending: [1, 3, 4, 5, 6, 7, 8, 9, 12, 79]
Sorted List Descending: [79, 12, 9, 8, 7, 6, 5, 4, 3, 1]
List after Appending: [79, 12, 9, 8, 7, 6, 5, 4, 3, 1, 91, 0]
```

Chapter 1 Ex 10 - Film Dictionary

This code creates a simple film directory using a Python dictionary. The dictionary contains details such as the film title, director, release year, genre, and rating. The program then displays these film details using a loop, showcasing how to iterate through key-value pairs. Below, you'll find the Python code, its corresponding output, and additional comments explaining each step.

Code

```python
# Chapter 1 Exercise 10 - Film Directory

# Creating a dictionary for a film
film_detail = {
    "Title": "Inception",
    "Director": "Christopher Nolan",
    "Year": 2010,
    "Genre": "Science Fiction",
    "Rating": 8.8
}

# Display film details using a loop
print("Film Details:")

# Using items() to directly access key-value pairs in the loop
for key, value in film_detail.items():
    print(f"{key}: {value}")

# End marker
```

Output

```
Film Details:
Title: Inception
Director: Christopher Nolan
Year: 2010
Genre: Science Fiction
Rating: 8.8
```

Chapter 1 Ex 11 - Year Tuples

This code explores the manipulation of a tuple containing various years. It demonstrates the retrieval of a specific value, the reversal of the entire tuple, counting occurrences of a particular year, obtaining the index of a specified value, and determining the length of the tuple. Below, you'll find the Python code, its corresponding output, and additional comments explaining each step.

Code

```python
# Chapter 1 Exercise 11 - Year Tuples

# Given tuple values
year = (2017, 2003, 2011, 2005, 1987, 2009, 2020, 2018, 2009)

# Accessing the value at index -3
value_index_3 = year[-3]
print(f"Value at index -3: {value_index_3}")

# Reversing the tuple using slicing [start:stop:step]
reversed_year = year[::-1]

print(f"Original Tuple: {year}")
print(f"Reversed Tuple: {reversed_year}")

# Counting the number of times 2009 is in the tuple using a loop
count_2009 = 0
for value in year:
    if value == 2009:
        count_2009 += 1

# Get the index value of 2018
value_index = year.index(2018)
print(f"Index value of 2018: {value_index}")

# Find the length of the given tuple
tuple_length = len(year)
print(f"Length of the tuple: {tuple_length}")

# End marker
```

Output

```
Value at index -3: 2020
Original Tuple: (2017, 2003, 2011, 2005, 1987, 2009, 2020, 2018, 2009)
Reversed Tuple: (2009, 2018, 2020, 2009, 1987, 2005, 2011, 2003, 2017)
Index value of 2018: 7
Length of the tuple: 9
```

Chapter 1 Ex 12 - Area Function

This code calculates the area of geometric shapes (square, circle, triangle). It uses functions for each shape, a menu for user interaction, and the math module for circle calculations. Below is the Python code, its output, and comments explaining each step.

Code

```python
# Chapter 1 Exercise 12 - Area Fucntion

# Using math to do operations
import math

def calculate_square_area():
    # Function to calculate the area of a square
    side_length = float(input("Enter the side length of the square: "))
    area = side_length ** 2
    print(f"The area of the square is: {area}")

def calculate_circle_area():
    # Function to calculate the area of a circle
    radius = float(input("Enter the radius of the circle: "))
    area = math.pi * radius ** 2
    print(f"The area of the circle is: {area}")

def calculate_triangle_area():
    # Function to calculate the area of a triangle
    base_length = float(input("Enter the base length of the triangle: "))
    height = float(input("Enter the height of the triangle: "))
    area = 0.5 * base_length * height
    print(f"The area of the triangle is: {area}")

def main():
    # Main program to display a menu and calculate area based on user input
    while True:
        print("Menu:")
        print("1: Calculate the area of a square")
        print("2: Calculate the area of a circle")
        print("3: Calculate the area of a triangle")
        print("4: Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == '1':
            calculate_square_area()
        elif choice == '2':
            calculate_circle_area()
        elif choice == '3':
            calculate_triangle_area()
        elif choice == '4':
            print("Exiting the program. Goodbye!")
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 4.")

# Call the main function directly
main()
```

Output

```
Menu:
1: Calculate the area of a square
2: Calculate the area of a circle
3: Calculate the area of a triangle
4: Exit
Enter your choice (1-4): 1
Enter the side length of the square: 40
The area of the square is: 1600.0
Menu:
1: Calculate the area of a square
2: Calculate the area of a circle
3: Calculate the area of a triangle
4: Exit
Enter your choice (1-4): 4
Exiting the program. Goodbye!
```

Chapter 1 Ex 13 - Product of list items

This code showcases a simple program that calculates the product of values in a given list. The provided Python code, its corresponding output, and explanatory comments are presented below.

Code

```python
# Chapter 1 Exercise 13 - Product of list items

# making a random list
my_list = [2, 3, 5, 7, 11,66]

def calculate_product(input_list):
    # Initialize the product to 1 so can multiply
    product = 1

    # to Calculate the product of values in the list
    for value in input_list:
        product *= value

    # Return the calculated product
    return product

# Printing the result
print("Product of the list values:", calculate_product(my_list))

# End marker
```

Output

```
Product of the list values: 152460
```

Chapter 1 Bonus Ex A - Multiplication Tables

This code generates multiplication tables for numbers 1 through 10. It utilizes nested loops to iterate through each multiplication table and displays the results. Below, you'll find the Python code, its corresponding output, and additional comments explaining each step.

Code

```python
# Chapter 1 Bonus Exercise A - Multiplication Tables

# Outer loop for tables from 1 to 10
for i in range(1, 11):
    print(f"Multiplication table of : {i}")

    # Inner loop for multiplying numbers from 1 to 10
    for j in range(1, 11):
        result = i * j
        print(f"{i} x {j} = {result}")

    # Print a separator line between tables
    print("-" * 20)

# End marker
```

Output

```
Multiplication table of : 1
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10
…
Multiplication table of : 10
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100
```

Chapter 1 Bonus Ex B - Locations List

This code demonstrates the manipulation of a list of locations. The original list is displayed, sorted in different ways, reversed, and then sorted in place. Below, you'll find the Python code, its corresponding output, and additional comments explaining each step.

Code

```python
# Chapter 1 Bonus Exercise B - Location List

# List of locations
locations = ['dubai', 'paris', 'switzerland', 'London', 'amsterdam', 'New York']

# Displaying the original list and its length
print(f"Original List: \n{locations}")
print(f"\nLength of the list: {len(locations)}")

# Sorting the list in alphabetical order using casefold for case insensitivity
sorted_locations = sorted(locations, key=str.casefold)
print(f"\nSorted List: \n{sorted_locations}")
print(f"Original list: \n{locations}")

# Reverse alphabetical sort and displaying the sorted list
sorted_locations = sorted(locations, key=str.casefold, reverse=True)
print(f"\nSorted Reverse List: \n{sorted_locations}")

# Reversing the original list and displaying it
locations.reverse()
print(f"\nReversed List: \n{locations}")

# Alphabetical sort using list.sort() method and displaying the sorted list
locations.sort(key=str.casefold)
print(f"\nAlphabetical order: \n{locations}")

# Reverse alphabetical sort using list.sort() method and displaying the sorted list
locations.sort(reverse=True, key=str.casefold)
print(f"\nReverse alphabetical:\n{locations}")

# End marker
```

Output

```
Original List:
['dubai', 'paris', 'switzerland', 'London', 'amsterdam', 'New York']

Length of the list: 6

Sorted List:
['amsterdam', 'dubai', 'London', 'New York', 'paris', 'switzerland']
Original list:
['dubai', 'paris', 'switzerland', 'London', 'amsterdam', 'New York']

Sorted Reverse List:
['switzerland', 'paris', 'New York', 'London', 'dubai', 'amsterdam']

Reversed List:
['New York', 'amsterdam', 'London', 'switzerland', 'paris', 'dubai']

Alphabetical order:
['amsterdam', 'dubai', 'London', 'New York', 'paris', 'switzerland']

Reverse alphabetical:
['switzerland', 'paris', 'New York', 'London', 'dubai', 'amsterdam']
```

Chapter 1 Bonus Ex C - Calculator Function

This code implements a simple calculator using a loop to allow the user to perform multiple calculations. while True:

Code

```python
# Chapter 1 Bonus Exercise C - Calculator Function

# A loop for the Calculator
while True:
    try:
        # Asking user to choose an operation
        operation_choice = int(input("Calculator Menu:\n1. Add\n2. Subtract\n3. Multiply\n4. Divide\n5. Modulus\n"))
    except ValueError:
        # Error for non-integer input
        print("Invalid choice. Please enter a number between 1 and 5.")
        continue

    if operation_choice not in [1, 2, 3, 4, 5]:
        # If the user inputs an invalid choice ask again
        print("Invalid choice. Please enter a number between 1 and 5.")
        continue

    # Getting values for calculation
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    # Doing Addition
    if operation_choice == 1:
        result = num1 + num2
        operation_name = "Addition"

    # Doing Subtraction
    elif operation_choice == 2:
        result = num1 - num2
        operation_name = "Subtraction"

    # Doing Multiplication
    elif operation_choice == 3:
        result = num1 * num2
        operation_name = "Multiplication"

    # Doing Division
    elif operation_choice == 4:
        if num2 != 0:
            result = num1 / num2
            operation_name = "Division"
        else:
            print("Cannot divide by zero.")
            continue

    # Doing Modulus
    elif operation_choice == 5:
        if num2 == 0:
            result = num1 % num2
            operation_name = "Modulus"
        else:
            print("Cannot calculate modulus with zero divisor.")
            continue
    # Printing result
    print(f"The result of {operation_name} is {result}")

    # Asking user if wants to perfrom another
    another_calculation = input("Do you want to perform another calculation? (y/n): ").lower()
    if another_calculation.lower() == 'n':
        print("Exiting the calculator. 4Goodbye!")
        break

# End marker
```

Output

```
Calculator Menu:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulus
1
Enter the first number: 4
Enter the second number: 4
The result of Addition is 8.0
Do you want to perform another calculation? (y/n): y
Calculator Menu:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulus
2
Enter the first number: 2
Enter the second number: 2
The result of Subtraction is 0.0
Do you want to perform another calculation? (y/n): n
Exiting the calculator. 4Goodbye!
```

Chapter 1 Further Ex I - Count seconds

This code presents a program that prompts the user to enter the number of days and calculates the corresponding number of seconds. The program ensures valid input by checking if the user's input consists of digits. Below, you'll find the Python code, its corresponding output, and additional comments detailing each step.

Code

```python
# Chapter 1 Further Exercise I - Count seconds

while True:
    # Getting user input for the number of days
    user_day_input = input("Enter the number of days: ")

    # Checking if the input is of digits
    if user_day_input.isdigit():
        # Converting the input to an integer
        user_day = int(user_day_input)

        # Calculating the number of seconds if the input is a valid integer
        day = user_day * 24
        minutes = day * 60
        seconds = minutes * 60

        # Displaying the result
        print(f"The seconds in {user_day} days is: {seconds} seconds")

        # Exiting the loop
        break
    else:
        # Informing the user about invalid input and prompting to try again
        print("Invalid input. Please enter a valid integer for the number of days. Try again.")

# End marker
```

Output

```
Enter the number of days: 12
The seconds in 12 days is: 1036800 seconds
```

Chapter 1 Further Ex II - Sum of Digits in number

This code presents a program that calculates the sum of digits in a given number. It defines a function, sum of digits, which takes an integer as input, converts it to a string to iterate through each digit, and computes the sum. The user is prompted to input a number, and the program displays the result – the sum of digits in the entered number.

Code

```python
# Chapter 1 Further Exercise II - Sum of Digits in a number

# Function to calculate the sum of digits
def sum_of_digits(number):
    # Converting the number to a string to iterate through each digit
    str_number = str(number)

    # Initialize sum
    digit_sum = 0

    # Iterate through each digit and add it to the sum
    for digit in str_number:
        digit_sum += int(digit)

    return digit_sum

# Inputing from the user
user_number = int(input("Enter a number: "))

# Calculate the sum of digits using the function
result = sum_of_digits(user_number)

# Displaying the result
print(f"The sum of digits in {user_number} is: {result}")
```

Output

```
Enter a number: 123
The sum of digits in 123 is: 6
```

Chapter 1 Further Ex III - Arrows

This code showcases a simple program that generates a pattern using asterisks. The pattern consists of rows where the number of asterisks in each row increases incrementally, creating a visually appealing pyramid-like structure. The number of rows in the pattern is set to 5, but this can be easily adjusted.

Code

```python
# Chapter 1 Further Exercise II - Sum of Digits in a number

# Define the number of rows for the pattern
num = 5

# Iterate through each row
for i in range(num):
    # Print spaces before the asterisks for the current row
    for j in range(0, num - i - 1):
        print(" ", end="")

    # Print asterisks for the current row
    for j in range(0, 2 * i + 1):
        print("*", end="")

    # Move to the next line after printing the row
    print()

# End marker
```

Output

```
    *
   ***
  *****
 *******
*********
```

Chapter 1 Further Ex IV - Count items

This code demonstrates a program that counts the occurrences of each staff member in a given list. It uses a dictionary to store the count of each staff member and iterates through the list, updating the counts accordingly. The final output displays the count for each staff member. Below, you'll find the Python code, its corresponding output, and additional comments explaining each step.

Code

```python
# Chapter 1 Further Exercise IV - Count items

# List of staff members
staff = ["Arshiya", "Usman", "Iftikhar", "Usman", "Rafia", "Mary", "Anmol",
         "Zainab", "Iftikhar", "Arshiya", "Rafia", "Jake"]

# Dictionary to store the count of each staff member
staff_count = {}

# To Go through each staff member in the list
for item in staff:
    # Checking if the current staff member already exists in the dictionary.
    if item in staff_count:
        # If the staff member exists increment their count by 1
        staff_count[item] += 1
    else:
        # If the staff member does not exist add them to the dictionary with a count of 1
        staff_count[item] = 1

# Printing the counts for each staff member
for key, value in staff_count.items():
    print(f"{key}: {value} times")

# End marker
```

Output

```
Arshiya: 2 times
Usman: 2 times
Iftikhar: 2 times
Rafia: 2 times
Mary: 1 times
Anmol: 1 times
Zainab: 1 times
Jake: 1 times
```

Chapter 1 Further Ex V - Lamda function

This code utilizes lambda functions to sort a list of tuples representing marks for various courses. The tuples consist of the course name and the corresponding mark.

Code

```python
# Chapter 1 Further Exercise V - Lamda function

# Listing of tuples representing marks for various courses
marks = [("CodeLab I", 67), ("web Development", 75), ("CodeLabII", 74),
         ("Smartphone Apps", 68), ("Games Development", 70), ("Responsive web", 65)]

# Sorting low to high based on marks using lambda function
low_to_high = sorted(marks, key=lambda x: x[1])

# Sorting high to low based on marks using lambda function
high_to_low = sorted(marks, key=lambda x: x[1], reverse=True)

# Displaying the results
print("Low to High:", low_to_high)
print("High to Low:", high_to_low)

# End marker
```

Output

```
Low to High: [('Responsive web', 65), ('CodeLab I', 67), ('Smartphone Apps', 68), ('Games Development', 70), ('CodeLabII', 74), ('web Development', 75)]
High to Low: [('web Development', 75), ('CodeLabII', 74), ('Games Development', 70), ('Smartphone Apps', 68), ('CodeLab I', 67), ('Responsive web', 65)]
```

Chapter 2 - Graphical User Interface

Exercises, Bonus And Further

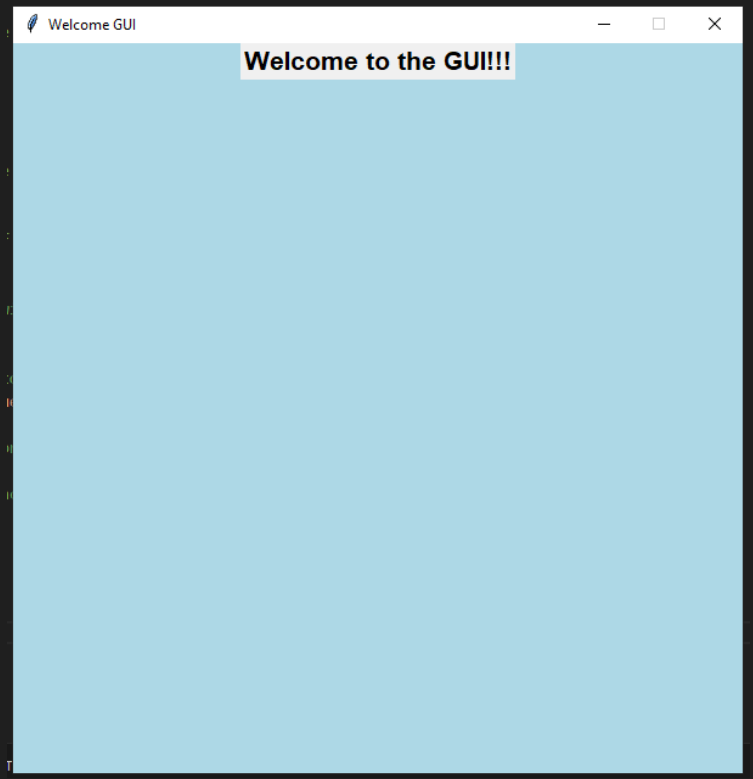
Chapter 2 Ex 1 - Welcome

This code demonstrates a simple graphical user interface (GUI) program using the tkinter module. The program creates a window with a welcoming message. Below, you'll find the Python code, its corresponding output, and additional comments explaining each step.

Code

```python
# Chapter 2  Exercise 1 - Welcome

# Import the tkinter module
from tkinter import *

# Creating a root window
root = Tk()

# Setting the title for the window
root.title('Welcome GUI')

# Setting the dimensions of the window
root.geometry('600x600')

# Disable resizing of the window
root.resizable(0, 0)

# Adding background color to the window
root.configure(bg="lightblue")

# Creating a custom font for the label
welcome_label = Label(root, text="Welcome to the GUI!!!", font=("Arial", 16, "bold"))
# to display label using pack
welcome_label.pack()

# Run the main event loop
root.mainloop()

#End marker
```

Output



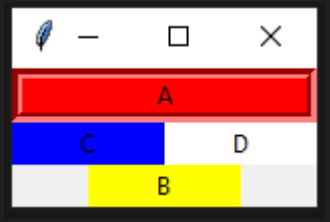
Chapter 2 Ex 2 - Managing Layout

This code showcases the use of the Tkinter library to create a simple GUI with four labels. Each label is styled differently and arranged at different positions within the main window.

Code

```python
# Chapter 2  Exercise 2 - Managing Layout

# using the tkinter library
from tkinter import *

# Creating the main window
root = Tk()

# Setting the title of the window
root.title("Four Labels Example")

# Creating Label 1 with style
label1 = Label(root, text="A", bg="red", width=10, bd=5, relief="groove")
label1.pack(side=TOP, fill=X, expand=1)  # Using pack to display and arrange Label 1 at the top

# Creating Label 2 with style
label2 = Label(root, text="B", bg="yellow", width=10)
label2.pack(side=BOTTOM)  # Using pack to display and arrange Label 2 at the bottom

# Creating Label 3 with style
label3 = Label(root, text="C", bg="blue", width=10)
label3.pack(side=LEFT, fill=Y, expand=1)  # Using pack to display and arrange Label 3 on the left

# Creating Label 4 with style
label4 = Label(root, text="D", bg="white", width=10)
label4.pack(side=RIGHT)  #  Using pack to display and arrange Label 4 on the right

# Starting the Tkinter window
root.mainloop()

# End marker
```

Output



Chapter 2 Ex 2 b - Square Grid

This code showcases a simple graphical user interface (GUI) program using the tkinter library. The program creates a window with two frames, each containing labeled elements.

Code

```python
# Chapter 2  Exercise 2 b - Square Grid

# using the tkinter library
from tkinter import *

# Create the main window
root = Tk()

# Setting the title of the window
root.title("Labels Inside Frames")

# Creating left frame and displaying it with pack
left_frame = Frame(root, bd=5, relief="groove")
left_frame.pack(side=LEFT, expand=True, fill=BOTH)

# Creating right frame and displaying it with pack
right_frame = Frame(root, bd=5, relief="groove")
right_frame.pack(side=RIGHT, expand=True, fill=BOTH)

# Creating labels A and B inside the left frame with style
label_a = Label(left_frame, text="A", bg="#22263d",fg="white", width=10, bd=5 )
label_b = Label(left_frame, text="B", bg="white", width=10, bd=5 )

# Using Pack labels A and B in the left frame
label_a.pack(side=TOP, expand=True, fill=BOTH)
label_b.pack(side=BOTTOM, expand=True, fill=BOTH)

# Creating labels C and D inside the right frame
label_c = Label(right_frame, text="C", bg="white", width=10, bd=5)
label_d = Label(right_frame, text="D", bg="#22263d",fg="white", width=10, bd=5, )

# Packing labels C and D in the right frame
label_c.pack(side=TOP, expand=True, fill=BOTH)
label_d.pack(side=BOTTOM, expand=True, fill=BOTH)

# Start the Tkinter event loop
root.mainloop()

# End marker
```

Output

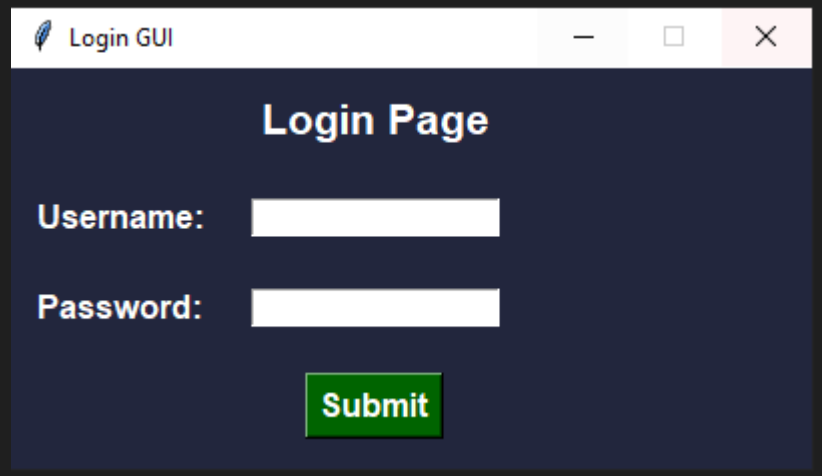
Chapter 2 Ex 3 - Login page

GUI-based login page using tkinter with welcome message, username/password entry, submit button, and result label.

Code

```python
# Chapter 2  Exercise 3 - Login Page

# using the tkinter library
from tkinter import *

# Creating a root window
root = Tk()

# Setting the title for the window
root.title('Login GUI')

# Setting the dimensions of the window
root.geometry('400x200')

# Disable resizing of the window
root.resizable(0, 0)

# Adding background color to the window
root.configure(bg="#22263d")

# Creating a custom font for the labels
font_style = ("Arial", 12, "bold")

# Creating and placing a label for welcome message
welcome_label = Label(root, text="Login Page", font=("Arial", 16, "bold"), bg="#22263d", fg="white")
welcome_label.grid(row=0, column=1, pady=11, padx=1)

# Creating and placing labels and entry widgets in separate rows
label_login = Label(root, text="Username:", font=font_style, bg="#22263d", fg="white")
label_password = Label(root, text="Password:", font=font_style, bg="#22263d", fg="white")

# Displaing the login and password text
label_login.grid(row=1, column=0, pady=10, padx=10)
label_password.grid(row=2, column=0, pady=10, padx=10)

# Creating and placing labels and entry widgets in separate rows
entry_login = Entry(root)
entry_password = Entry(root, show="*")

# Displaing the login and password field
entry_login.grid(row=1, column=1, pady=10, padx=10)
entry_password.grid(row=2, column=1, pady=10, padx=10)

# Creating and placing a button
btn_submit = Button(root, text="Submit", bg="darkgreen", fg="white",font=("Arial", 12, "bold"))
btn_submit.grid(row=3, column=1, pady=10)

# Creating and placing a label to display results
result_label = Label(root, text="", font=font_style, bg="#22263d", fg="white")
result_label.grid(row=4, column=1, pady=10)

# Run the main event loop
root.mainloop()

# End Marker
```

Output



Chapter 2 Ex 4 – Registration page

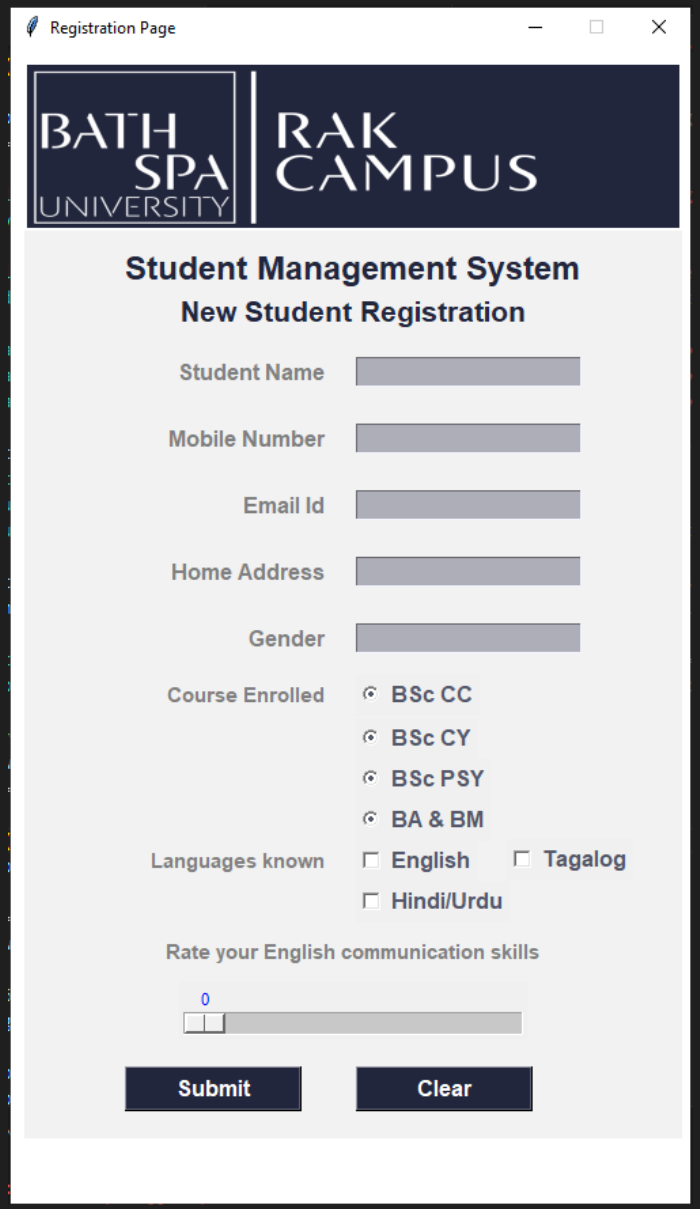
This Python code utilizes Tkinter to create a student registration GUI. The form captures essential details: name, mobile number, email, address, gender, enrolled course, languages, and English communication skills. The breakdown includes code and output

Code

```python
# Chapter 2  Exercise 4 - Registration Page

# using the tkinter library
from tkinter import *

# Creating a root window
root = Tk()

# Setting the title for the window
root.title('Registration Page')

# Setting the size of the window
root.geometry('500x850')  # Smaller window size

# Disable resizing of the window
root.resizable(0, 0)

# Adding background color to the window
root.configure(bg="white")

# Creating a frame for the entire content with adjusted width
content_frame = Frame(root, width=480, height=600, bg="#f2f2f2")  # Adjusted width and height
content_frame.grid(padx=10, pady=10)

# Creating a label for the image (banner)
banner = PhotoImage(file='Chapter 2 -Grpahical User Interface\images\RAK.png', width=480, height=120)  # Adjusted width
banner_lbl = Label(content_frame, image=banner, bg="white")
banner_lbl.grid(row=0, column=0, columnspan=2)

# Creating the main heading
head1 = Label(content_frame, font=('Roboto', 18, 'bold'), text="Student Management System", fg='#22263d', bg="#f2f2f2")
head1.grid(row=1, column=0, columnspan=2, pady=(10, 0))

# Creating the sub heading
head2 = Label(content_frame, font=('Roboto', 16, 'bold'), text="New Student Registration", fg='#22263d', bg="#f2f2f2")
head2.grid(row=2, column=0, columnspan=2, pady=(0, 5))

# Labels and Entry Widgets with adjusted layout
label_login = Label(content_frame, font=('Roboto', 12, 'bold'), text="Student Name", fg="gray", bg="#f2f2f2")
entry_name = Entry(content_frame, width=18, bg='#ADAEB7', font=('Roboto', 12))

label_password = Label(content_frame, font=('Roboto', 12, 'bold'), text="Mobile Number", fg="gray", bg="#f2f2f2")
entry_mobile = Entry(content_frame, width=18, bg='#ADAEB7', font=('Roboto', 12))

label_mail = Label(content_frame, font=('Roboto', 12, 'bold'), text="Email Id", fg="gray", bg="#f2f2f2")
entry_email = Entry(content_frame, width=18, bg='#ADAEB7', font=('Roboto', 12))

label_add = Label(content_frame, font=('Roboto', 12, 'bold'), text="Home Address", fg="gray", bg="#f2f2f2")
entry_home_address = Entry(content_frame, width=18, bg='#ADAEB7', font=('Roboto', 12))

label_gender = Label(content_frame, font=('Roboto', 12, 'bold'), text="Gender", fg="gray", bg="#f2f2f2")
entry_gender = Entry(content_frame, width=18, bg='#ADAEB7', font=('Roboto', 12))

label_course = Label(content_frame, font=('Roboto', 11, 'bold'), text="Course Enrolled", fg="gray", bg="#f2f2f2")
label_radiobtn = Radiobutton(content_frame, text="BSc CC", fg='#545869', font=('Roboto', 12, 'bold'))

label_radiobtn = Radiobutton(content_frame, text="BSc CC", fg='#545869', font=('Roboto', 12, 'bold'))

label_radiobtn1 = Radiobutton(content_frame, text="BSc CY", fg='#545869', font=('Roboto', 12, 'bold'))
label_radiobtn2 = Radiobutton(content_frame, text="BSc PSY", fg='#545869', font=('Roboto', 12, 'bold'))
label_radiobtn3 = Radiobutton(content_frame, text="BA & BM", fg='#545869', font=('Roboto', 12, 'bold'))

label_lang = Label(content_frame, font=('Roboto', 11, 'bold'), text="Languages known", fg="gray", bg="#f2f2f2")
label_box = Checkbutton(content_frame, text='English', fg='#545869', font=('Roboto', 12, 'bold'))
label_box1 = Checkbutton(content_frame, text='Tagalog', fg='#545869', font=('Roboto', 12, 'bold'))
label_box2 = Checkbutton(content_frame, text='Hindi/Urdu', fg='#545869', font=('Roboto', 12, 'bold'))

label_rate = Label(content_frame, font=('Roboto', 11, 'bold'), text="Rate your English communication skills", fg="gray",
scale = Scale(content_frame, from_=0, to=100, orient="horizontal", length=250, fg='blue')

submit_button = Button(content_frame, text="Submit", bg="#22263d", fg="white", width=12, font=('Roboto', 12, 'bold'))
clear_button = Button(content_frame, text="Clear", bg="#22263d", fg="white", width=12, font=('Roboto', 12, 'bold'))

# Displaying the forum text and entry widgets on the same row
label_login.grid(row=3, column=0, pady=12, padx=10, sticky=E)
entry_name.grid(row=3, column=1, pady=12, padx=10, sticky=W)

label_password.grid(row=4, column=0, pady=12, padx=10, sticky=E)
entry_mobile.grid(row=4, column=1, pady=12, padx=10, sticky=W)

label_mail.grid(row=5, column=0, pady=12, padx=10, sticky=E)
entry_email.grid(row=5, column=1, pady=12, padx=10, sticky=W)

label_add.grid(row=6, column=0, pady=12, padx=10, sticky=E)
entry_home_address.grid(row=6, column=1, pady=12, padx=10, sticky=W)

label_gender.grid(row=7, column=0, pady=12, padx=10, sticky=E)
entry_gender.grid(row=7, column=1, pady=12, padx=10, sticky=W)

label_course.grid(row=8, column=0, pady=5, padx=10, sticky=E)
label_radiobtn.grid(row=8, column=1, padx=10, sticky=W)

label_radiobtn1.grid(row=9, column=1, padx=10, sticky=W)
label_radiobtn2.grid(row=10, column=1, padx=10, sticky=W)
label_radiobtn3.grid(row=11, column=1, padx=10, sticky=W)

label_lang.grid(row=12, column=0, padx=10, sticky=E)
label_box.grid(row=12, column=1, padx=10, sticky=W)

label_box1.place(x=355, y=572)
label_box2.grid(row=13, column=1, padx=10, sticky=W)

label_rate.grid(row=14, column=0, columnspan=2, pady=(10, 0))
scale.grid(row=15, column=0, columnspan=2, pady=(10, 0))

submit_button.grid(row=16, column=0, pady=20, padx=(15, 30), sticky=E)
clear_button.grid(row=16, column=1, pady=20, padx=(10, 50), sticky=W)

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

Chapter 2 Ex 4 – Registration page

Output



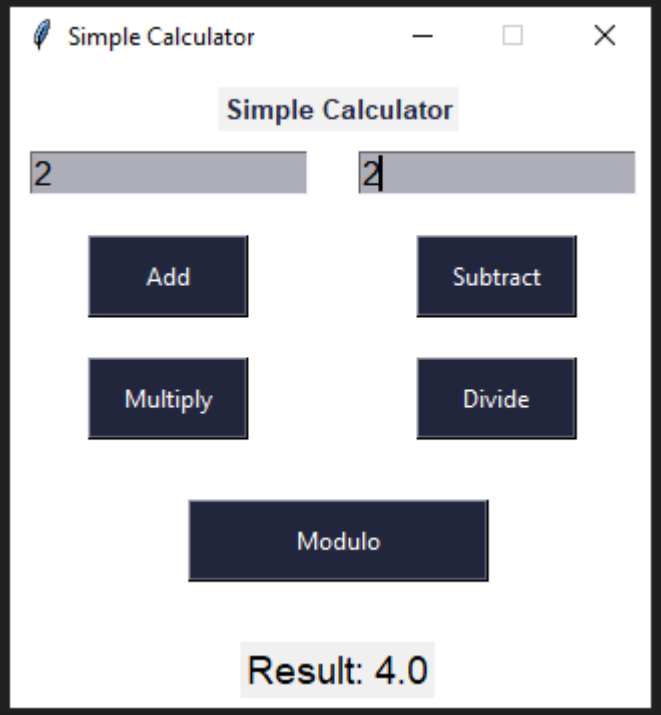
Chapter 2 Ex 5 - Calculator

This Python code uses tkinter to create a basic calculator with GUI. Users can perform addition, subtraction, multiplication, division, and modulo operations. The result is displayed on the interface, with comments explaining each function.

Code

```python
# Chapter 2  Exercise 5 - Calculator

# using the tkinter library
from tkinter import *

# Function to perform addition
def add():
    num1 = entry_num1.get()
    num2 = entry_num2.get()

    # Checking if both inputs are valid numbers
    if num1.isdigit() and num2.isdigit():
        result = float(num1) + float(num2)
        result_label.config(text=f"Result: {result}")
    else:
        result_label.config(text="Invalid input")

# Function to perform subtraction
def subtract():
    num1 = entry_num1.get()
    num2 = entry_num2.get()

    # Checking if both inputs are valid numbers
    if num1.isdigit() and num2.isdigit():
        result = float(num1) - float(num2)
        result_label.config(text=f"Result: {result}")
    else:
        result_label.config(text="Invalid input")

# Function to perform multiplication
def multiply():
    num1 = entry_num1.get()
    num2 = entry_num2.get()

    # Checking if both inputs are valid numbers
    if num1.isdigit() and num2.isdigit():
        result = float(num1) * float(num2)
        result_label.config(text=f"Result: {result}")
    else:
        result_label.config(text="Invalid input")

# Function to perform division
def divide():
    num1 = entry_num1.get()
    num2 = entry_num2.get()

    # Check if both inputs are valid numbers
    if num1.isdigit() and num2.isdigit():
        # Checking if the divisor is not zero
        if float(num2) != 0:
            result = float(num1) / float(num2)
            result_label.config(text=f"Result: {result}")
        else:
            result_label.config(text="Cannot divide by zero")
    else:
        result_label.config(text="Invalid input")

# Function to perform modulo division
def modulo():
    num1 = entry_num1.get()
    num2 = entry_num2.get()

    # Checking if both inputs are valid numbers
    if num1.isdigit() and num2.isdigit():
        # Checking if the divisor is not zero
        if float(num2) != 0:
            result = float(num1) % float(num2)
            result_label.config(text=f"Result: {result}")
        else:
            result_label.config(text="Cannot calculate modulo with zero")
    else:
        result_label.config(text="Invalid input")


# Creating the main window
root = Tk()

# Setting the title for the window
root.title("Simple Calculator")

# Setting the size of the window
root.geometry('320x320')

# Disable resizing of the window
root.resizable(0, 0)

# Adding a white background to the main window
root.configure(bg='white')

head1 = Label(root, font=('Roboto', 10, 'bold'), text="Simple Calculator", fg='#22263d', bg="#f2f2f2")
head1.grid(row=0, column=0, columnspan=2, pady=(10, 0))

# Making Entry widgets for numbers
entry_num1 = Entry(root, width=15, font=('Arial', 12), bg='#ADAEB7')
entry_num1.grid(row=1, column=0, padx=10, pady=10)

entry_num2 = Entry(root, width=15, font=('Arial', 12), bg='#ADAEB7')
entry_num2.grid(row=1, column=1, padx=15, pady=10)

# Making Buttons to perform operations
add_button = Button(root, text="Add", command=add, bg='#22263d', fg='white', height=2, width=10)
add_button.grid(row=2, column=0, padx=10, pady=10)

subtract_button = Button(root, text="Subtract", command=subtract, bg='#22263d', fg='white', height=2, width=10)
subtract_button.grid(row=2, column=1, padx=10, pady=10)

multiply_button = Button(root, text="Multiply", command=multiply, bg='#22263d', fg='white', height=2, width=10)
multiply_button.grid(row=3, column=0, padx=10, pady=10)

divide_button = Button(root, text="Divide", command=divide, bg='#22263d', fg='white', height=2, width=10)
divide_button.grid(row=3, column=1, padx=10, pady=10)

modulo_button = Button(root, text="Modulo", command=modulo, bg='#22263d', fg='white', height=2, width=20)
modulo_button.grid(row=4, column=0, columnspan=2, pady=20)

# Making Label to display the result
result_label = Label(root, text="Result: ", font=('Arial', 14))
result_label.grid(row=5, column=0, columnspan=2, pady=10)

# Start the main loop
root.mainloop()

# End marker
```

Output



Chapter 2 Ex A Bonus - Temperature Converter

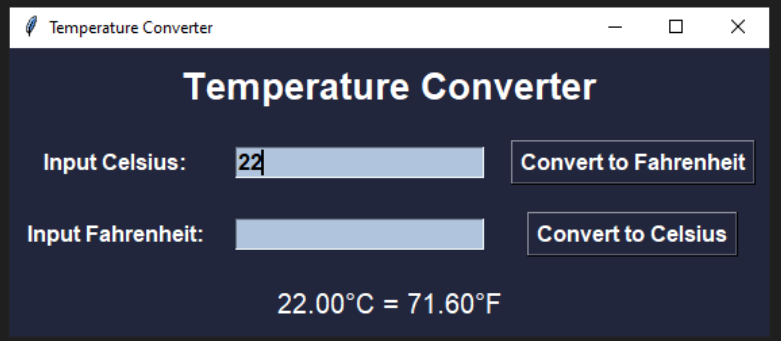
This code uses tkinter for a simple Celsius to Fahrenheit and Fahrenheit to Celsius converter. See the code, output, and comments below

Code

```python
# Chapter 2 Bonus Exercise A - Temperature Converter

# using the tkinter library
from tkinter import *

# Making a Function to convert from Celsius to Fahrenheit
def celsius_to_fahrenheit():
    user_input = celsius_entry.get()
    if user_input.isdigit():
        celsius = float(user_input)
        fahrenheit = (celsius * 9/5) + 32
        result_label.config(text=f"{celsius:.2f}°C = {fahrenheit:.2f}°F", fg="white")
    else:
        result_label.config(text="Invalid input", fg="red")

# Making a Function to convert from Fahrenheit to Celsius
def fahrenheit_to_celsius():
    user_input = fahrenheit_entry.get()
    if user_input.isdigit():
        fahrenheit = float(user_input)
        celsius = (fahrenheit - 32) * 5/9
        result_label.config(text=f"{fahrenheit:.2f}°F = {celsius:.2f}°C", fg="white")
    else:
        result_label.config(text="Invalid input", fg="red")


# Creating the main window
root = Tk()

# Setting the title for the window
root.title("Temperature Converter")

# Adding a white background to the main window
root.configure(bg='#22263d')
```

```python
# Making a heading
head1 = Label(root, font=('Roboto', 20, 'bold'), text="Temperature Converter", bg='#22263d',fg="white")
head1.grid(row=0, column=0, columnspan=3, pady=10,padx=10)

# Text and entry feild for Celsius to Fahrenheit Conversion
celsius_label = Label(root, text="Input Celsius:", font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
celsius_label.grid(row=1, column=0, padx=10, pady=10)

celsius_entry = Entry(root, font=('Roboto', 12,'bold'), bg='#B0C4DE', fg='black')
celsius_entry.grid(row=1, column=1, padx=10, pady=10)

# Button to convert
celsius_to_fahrenheit_button = Button(root, text="Convert to Fahrenheit", command=celsius_to_fahrenheit, font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
celsius_to_fahrenheit_button.grid(row=1, column=2, padx=10, pady=10)

# Text and entry feild Fahrenheit to Celsius Conversion
fahrenheit_label = Label(root, text="Input Fahrenheit:", font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
fahrenheit_label.grid(row=2, column=0, padx=10, pady=10)

fahrenheit_entry = Entry(root, font=('Roboto', 12, 'bold'), bg='#B0C4DE', fg='black')
fahrenheit_entry.grid(row=2, column=1, padx=10, pady=10)

fahrenheit_to_celsius_button = Button(root, text="Convert to Celsius", command=fahrenheit_to_celsius,font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
fahrenheit_to_celsius_button.grid(row=2, column=2, padx=10, pady=10)

# Display the result
result_label = Label(root, text="", font=("Helvetica", 15), bg='#22263d', fg="white")
result_label.grid(row=3, columnspan=3, pady=10)

# Start the Tkinter event loop
root.mainloop()

# End marker
```

Output



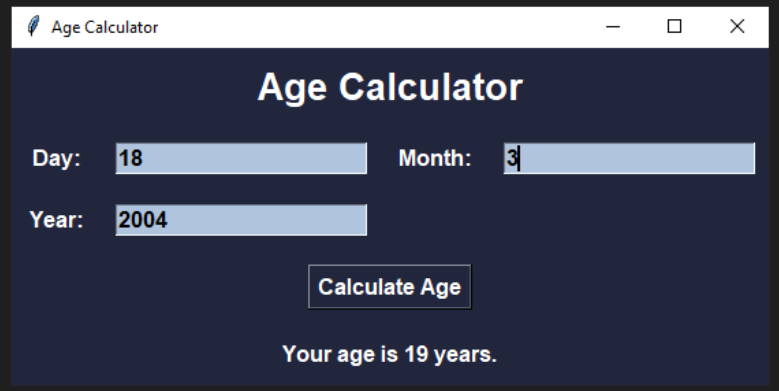
Chapter 2 Ex B Bonus - Age Calculator

This code uses tkinter for a simple Celsius to Fahrenheit and Fahrenheit to Celsius converter. See the code, output, and comments below

Code

```python
# Chapter 2 Bonus Exercise B - Age Calculator

# using the tkinter library and datetime
from tkinter import *
from datetime import datetime

# Function to calculate the age
def calculate_age():
    try:
        # Getting the day, month, and year from the input entries
        dob_day = int(dob_day_entry.get())
        dob_month = int(dob_month_entry.get())
        dob_year = int(dob_year_entry.get())

        # Creating a datetime object for the input date of birth
        dob = datetime(dob_year, dob_month, dob_day)

        # Getting the current date
        today = datetime.today()

        # Calculatingg the age, adjusting for whether the birthday has occurred this year by using '<'
        age = today.year - dob.year - ((today.month, today.day) < (dob.month, dob.day))

        # Display the result
        result_label.config(text=f"Your age is {age} years.", fg="white")
    except ValueError:
        # if the input date is not in the correct format to show error msg
        result_label.config(text="Invalid date format", fg="red")


# Creating the main window
root = Tk()

# Setting the title for the window
root.title("Age Calculator")

# Adding a blue background to the main window
root.configure(bg='#22263d')

# Heading
head1 = Label(root, font=('Roboto', 20, 'bold'), text="Age Calculator", bg='#22263d',fg="white")
head1.grid(row=0, column=0, columnspan=4, pady=10, padx=10)

# Age Calculation
# Label for Day input
dob_day_label = Label(root, text="Day:", font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
dob_day_label.grid(row=1, column=0, padx=10, pady=10)

# Entry field for Day input
dob_day_entry = Entry(root, font=('Roboto', 12,'bold'), bg='#B0C4DE', fg='black')
dob_day_entry.grid(row=1, column=1, padx=10, pady=10)

# Label for Month input
dob_month_label = Label(root, text="Month:", font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
dob_month_label.grid(row=1, column=2, padx=10, pady=10)

# Entry field for Month input
dob_month_entry = Entry(root, font=('Roboto', 12,'bold'), bg='#B0C4DE', fg='black')
dob_month_entry.grid(row=1, column=3, padx=10, pady=10)

# Label for Year input
dob_year_label = Label(root, text="Year:", font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
dob_year_label.grid(row=2, column=0, padx=10, pady=10)

# Entry field for Year input
dob_year_entry = Entry(root, font=('Roboto', 12,'bold'), bg='#B0C4DE', fg='black')
dob_year_entry.grid(row=2, column=1, padx=10, pady=10)

# Button to calculate age
calculate_button = Button(root, text="Calculate Age", command=calculate_age, font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
calculate_button.grid(row=3, column=0, columnspan=4, padx=10, pady=10)

# Result
result_label = Label(root, text="", font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
result_label.grid(row=4, column=0, columnspan=4, padx=10, pady=10)

# Start the Tkinter event loop
root.mainloop()

#End marker
```

Output



Chapter 2 Further Ex I - Count characters

This code demonstrates a program using the tkinter library to count the number of vowels, consonants, special characters, and total letters in a given sentence.

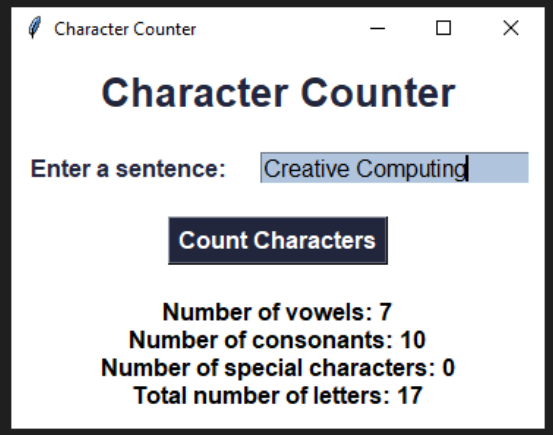
Code

```python
# Chapter 2 Bonus Exercise B - Age Calculator

# using the tkinter library and datetime
from tkinter import *
from datetime import datetime

# Function to calculate the age
def calculate_age():
    try:
        # Getting the day, month, and year from the input entries
        dob_day = int(dob_day_entry.get())
        dob_month = int(dob_month_entry.get())
        dob_year = int(dob_year_entry.get())

        # Creating a datetime object for the input date of birth
        dob = datetime(dob_year, dob_month, dob_day)

        # Getting the current date
        today = datetime.today()

        # Calculatingg the age, adjusting for whether the birthday has occurred this year by using '<'
        age = today.year - dob.year - ((today.month, today.day) < (dob.month, dob.day))

        # Display the result
        result_label.config(text=f"Your age is {age} years.", fg="white")
    except ValueError:
        # if the input date is not in the correct format to show error msg
        result_label.config(text="Invalid date format", fg="red")


# Creating the main window
root = Tk()

# Setting the title for the window
root.title("Age Calculator")

# Adding a blue background to the main window
root.configure(bg='#22263d')

# Heading
head1 = Label(root, font=('Roboto', 20, 'bold'), text="Age Calculator", bg='#22263d',fg="white")
head1.grid(row=0, column=0, columnspan=4, pady=10, padx=10)

# Age Calculation
# Label for Day input
dob_day_label = Label(root, text="Day:", font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
dob_day_label.grid(row=1, column=0, padx=10, pady=10)

# Entry field for Day input
dob_day_entry = Entry(root, font=('Roboto', 12,'bold'), bg='#B0C4DE', fg='black')
dob_day_entry.grid(row=1, column=1, padx=10, pady=10)

# Label for Month input
dob_month_label = Label(root, text="Month:", font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
dob_month_label.grid(row=1, column=2, padx=10, pady=10)

# Entry field for Month input
dob_month_entry = Entry(root, font=('Roboto', 12,'bold'), bg='#B0C4DE', fg='black')
dob_month_entry.grid(row=1, column=3, padx=10, pady=10)
```


```python
# Entry field for Day input
dob_day_entry = Entry(root, font=('Roboto', 12,'bold'), bg='#B0C4DE', fg='black')
dob_day_entry.grid(row=1, column=1, padx=10, pady=10)

# Label for Month input
dob_month_label = Label(root, text="Month:", font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
dob_month_label.grid(row=1, column=2, padx=10, pady=10)

# Entry field for Month input
dob_month_entry = Entry(root, font=('Roboto', 12,'bold'), bg='#B0C4DE', fg='black')
dob_month_entry.grid(row=1, column=3, padx=10, pady=10)
```


```python
# Label for Year input
dob_year_label = Label(root, text="Year:", font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
dob_year_label.grid(row=2, column=0, padx=10, pady=10)

# Entry field for Year input
dob_year_entry = Entry(root, font=('Roboto', 12,'bold'), bg='#B0C4DE', fg='black')
dob_year_entry.grid(row=2, column=1, padx=10, pady=10)

# Button to calculate age
calculate_button = Button(root, text="Calculate Age", command=calculate_age, font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
calculate_button.grid(row=3, column=0, columnspan=4, padx=10, pady=10)

# Result
result_label = Label(root, text="", font=('Roboto', 12,'bold'), bg='#22263d', fg='white')
result_label.grid(row=4, column=0, columnspan=4, padx=10, pady=10)

# Start the Tkinter event loop
root.mainloop()

#End marker
```

Output



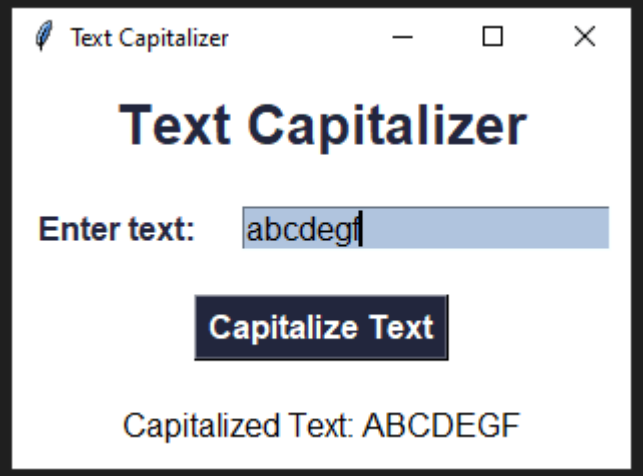
Chapter 2 Further Ex II - Capitalize letters

Tkinter app for capitalizing user text. GUI with entry, capitalization button, and result label. capitalize_text converts input to uppercase, updating the result label. Tkinter event loop starts.

Code

```python
# Chapter 2 Further Exercise II - Capitalize Letters

# using the tkinter library
from tkinter import *

def capitalize_text():
    # Getting the text entered by the user
    user_input = entry.get()

    # Converting the text to uppercase
    capitalized_text = user_input.upper()

    # the result label with the capitalized text
    result_label.config(text=f"Capitalized Text: {capitalized_text}")


# Creating the main window
root = Tk()

# Setting the title for the window
root.title("Text Capitalizer")

# Adding a white background to the main window
root.configure(bg='#FFFFFF')

# Heading
heading_label = Label(root, font=('Roboto', 20, 'bold'), text="Text Capitalizer", bg='#FFFFFF', fg='#22263d')
heading_label.grid(row=0, column=0, columnspan=2, pady=10, padx=10)

# Input label and entry
input_label = Label(root, text="Enter text:", font=('Roboto', 12, 'bold'), bg='#FFFFFF', fg='#22263d')
input_label.grid(row=1, column=0, pady=10, padx=10)

entry = Entry(root, font=('Roboto', 12), bg='#B0C4DE', fg='#000000')
entry.grid(row=1, column=1, pady=10, padx=10)

# Button to capitalize text
capitalize_button = Button(root, text="Capitalize Text", command=capitalize_text, font=('Roboto', 12, 'bold'),
                           bg='#22263d', fg='#FFFFFF')
capitalize_button.grid(row=2, column=0, columnspan=2, pady=10)

# Result
result_label = Label(root, text="", font=('Roboto', 12), bg='#FFFFFF', fg='#000000')
result_label.grid(row=3, column=0, columnspan=2, pady=10, padx=10)

# Start the Tkinter event loop
root.mainloop()

#End marker
```

Output



Chapter 3 - Graphical User Interface

Exercises, Bonus And Further

Chapter 3 Ex 1 - Greeting App

Captures user name and color choice for personalized greeting. Output: Tkinter window with entry fields, dropdown, and display label.

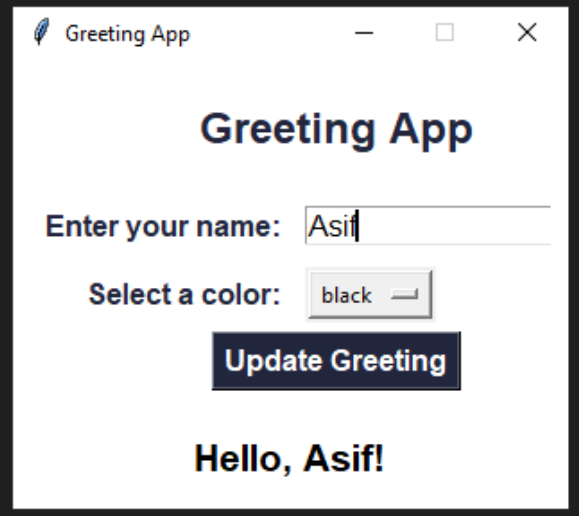
Comments provide step-by-step explanation.

Code

```python
# Chapter 3 Exercise 1 - Greeting App

# using the tkinter library
from tkinter import *

def update_greeting():
    # Getting the name from the entry widget
    name = name_entry.get()

    # Getting the selected color from the dropdown menu
    color = color_var.get()

    # Creating a personalized greeting message
    greeting_text = f"Hello, {name}!"

    # Updating the display label with the greeting message and selected color
    display_label.config(text=greeting_text, fg=color)


# Creating the main window
root = Tk()

# Setting the title of the window
root.title("Greeting App")

# Setting the window size
root.geometry("310x200")

# Disabling window resizable
root.resizable(0, 0)

# Adding a white background color to the window
root.configure(bg="white")

# InputFrame
input_frame = Frame(root, bg='white', width=300, height=150)
input_frame.pack(padx=10, pady=10, fill="both", expand=True)

# Title Label
title_label = Label(input_frame, text="Greeting App", font=('Helvetica', 18, 'bold'), fg='#22263d', bg='white')
title_label.grid(row=0, column=0, columnspan=2, pady=(10, 20))

# Entry for user's name
name_label = Label(input_frame, text="Enter your name:", font=('Roboto', 12, 'bold'), bg='#FFFFFF', fg='#22263d')
name_label.grid(row=1, column=0, padx=5, pady=5, sticky='e')

# Displaying the entry field
name_entry = Entry(input_frame, bg='white', fg='black', font=('Helvetica', 12))
name_entry.grid(row=1, column=1, padx=5, pady=5, sticky='w')

# Dropdown menu for color selection using OptionMenu
color_label = Label(input_frame, text="Select a color:", font=('Roboto', 12, 'bold'), bg='#FFFFFF', fg='#22263d')
color_label.grid(row=2, column=0, padx=5, pady=5, sticky='e')

# Setting the color list
colors = ['black', 'red', 'green', 'blue']  # Color
color_var = StringVar(value=colors[0])  # Starting color

# Using dropdown to display the list of color
color_dropdown = OptionMenu(input_frame, color_var, *colors)
color_dropdown.grid(row=2, column=1, padx=5, pady=5, sticky='w')
```

```python
# Button to update greeting
update_button = Button(input_frame, text="Update Greeting", command=update_greeting,
                       bg='#22263d', fg='white', font=('Helvetica', 12, 'bold'))
update_button.grid(row=3, column=0, columnspan=2, padx=10, sticky='n')

# DisplayFrame
display_frame = Frame(root, bg='white', width=300, height=150)
display_frame.pack(padx=10, pady=10, fill="both", expand=True)

# Label to display personalized greeting
display_label = Label(display_frame, text="", font=('Helvetica', 16), bg='Grey', fg='black')
display_label.pack(padx=5, pady=5)

# Starting the Tkinter event loop
root.mainloop()

# End marker
```

Output



Chapter 3 Ex 2 - Coffee Vending Machine

This code allows users to select coffee type, sugar, and milk, and calculates the cost of the order. It provides feedback on the total cost and change after the user enters the payment amount.

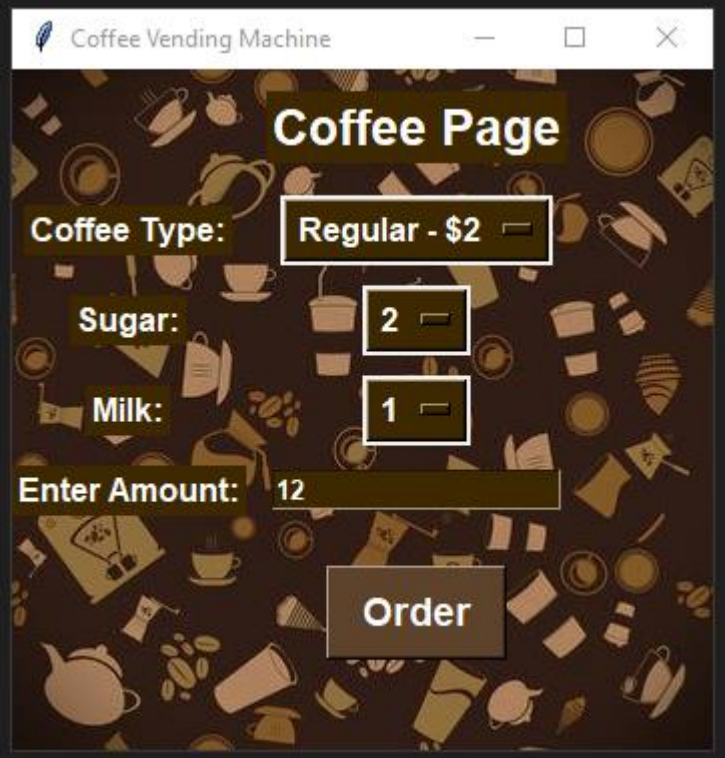
Code

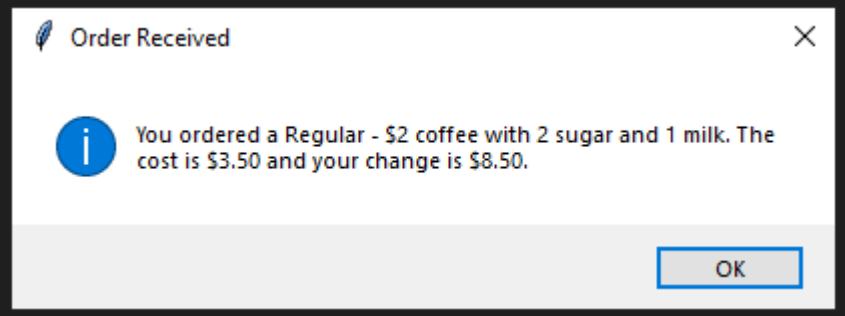
```python
# Chapter 3 Exercise 2 - Coffee Vending Machine

# using the tkinter library
from tkinter import *
from tkinter import messagebox

# Creating main window
root = Tk()

# Setting the title for the window
root.title("Coffee Vending Machine")

# Setting the window size
root.geometry("350x340")

# Adding background image
image_path = PhotoImage(file="Chapter 3 - Graphical User Interface(Cont.)\images\Coffee-bg.png")
bg = Label(root, image=image_path)
bg.place(relheight=1, relwidth=1)

# Function to handle user input
def process_order():
    coffee_type = coffee_var.get()
    sugar = int(sugar_var.get())
    milk = int(milk_var.get())

    try:
        amount_paid = float(amount_paid_entry.get())
        # will Display an error message if the entered amount is not a valid number
    except ValueError:
        messagebox.showerror("Invalid Amount", "Please enter a valid amount.")
        return

    # it will Calculate the cost of coffee
    cost = 2 + (sugar * 0.5) + (milk * 0.5)

    # Checking if the amount paid is sufficient
    if amount_paid >= cost:
        change = amount_paid - cost # substracting the cost
        order_message = f"You ordered a {coffee_type} coffee with {sugar} sugar and {milk} milk. The cost is $
        {cost:.2f} and your change is ${change:.2f}."
        messagebox.showinfo("Order Received", order_message)
    else:
        messagebox.showerror("Insufficient Funds", "You have not paid enough for your order.")

# Variables to store user inputs
coffee_var = StringVar()
sugar_var = StringVar()
milk_var = StringVar()

# Creating and placing a label for the welcome message
welcome_label = Label(root, text="Coffee Page", font=("Roboto", 19, "bold"), bg="#3d2800", fg="white")
welcome_label.grid(row=0, column=1, pady=11, padx=10)

# Creating a drop-down menu for coffee type
coffee_label = Label(root, text="Coffee Type:", bg="#3d2800", font=("Roboto", 12, 'bold'), fg="white")
coffee_label.grid(row=1, column=0, pady=5, padx=5)
coffee_option = OptionMenu(root, coffee_var, "Regular - $2", "Americano - $2", "Latte - $2")
coffee_option.config(font=("Roboto", 12, 'bold'), bg="#3d2800", fg="white")
coffee_option.grid(row=1, column=1, pady=5)

# Creating option menus for sugar and milk
sugar_label = Label(root, text="Sugar:", bg="#3d2800", font=("Roboto", 12, 'bold'), fg="white")
sugar_label.grid(row=2, column=0, pady=5)
sugar_option = OptionMenu(root, sugar_var, "1", "2", "3")
sugar_option.config(font=("Roboto", 12, 'bold'), bg="#3d2800", fg="white")
sugar_option.grid(row=2, column=1, pady=5)
# Creating label for Milk
milk_label = Label(root, text="Milk:", bg="#3d2800", font=("Roboto", 12, 'bold'), fg="white")
milk_label.grid(row=3, column=0, pady=5)

# Creating option menu for selecting Milk quantity
milk_option = OptionMenu(root, milk_var, "1", "2", "3")
milk_option.config(font=("Roboto", 12, 'bold'), bg="#3d2800", fg="white")
milk_option.grid(row=3, column=1, pady=5)


# Creating an entry box for the amount paid
amount_paid_label = Label(root, text="Enter Amount:", bg="#3d2800", font=("Roboto", 12, 'bold'), fg="white")
amount_paid_label.grid(row=4, column=0, pady=5)
amount_paid_entry = Entry(root, font=("Roboto", 10, 'bold'), bg="#3d2800", fg="white")
amount_paid_entry.grid(row=4, column=1, pady=5)

# Creating a button to process the order with some style
order_button = Button(root, text="Order", command=process_order, bg="#5d432c", fg="white",
                      font=("Roboto", 14, 'bold'), padx=10, pady=5)
order_button.grid(row=5, column=1, columnspan=2, pady=20)

# Starting the Tkinter event loop
root.mainloop()

# End marker
```

Chapter 3 Ex 2 - Coffee Vending Machine

This code allows users to select coffee type, sugar, and milk, and calculates the cost of the order. It provides feedback on the total cost and change after the user enters the payment amount.

Output





Chapter 3 Ex 3 – Area Function

Area Calculator using tkinter for circle, square, and rectangle. Entry widgets, Calculate buttons, and result labels provide a user-friendly interface.Functions validate inputs, and ttk module creates a tabbed interface for each shape.

Code

```python
# Chapter 3 Exercise 3 - Area fucntion

# using the tkinter library and messagebox
from tkinter import *
from tkinter import ttk, messagebox # tkk for notebook

# Function to calculate the area of a circle
def calculate_circle_area():
    # Getting the radius value from the entry widget
    radius_str = circle_radius_entry.get()
    # Checking if the input is a valid number
    if radius_str.isdigit():
        # Converting the radius to a float
        radius = float(radius_str)
        # Calculate the area of the circle
        area = 3.14 * radius**2
        # Update the result label with the calculated area
        circle_result_label.config(text=f"Area: {area} square units",
                                   fg="green", font=("Roboto", 12, 'bold'))
    else:
        # Showing an error message if the input is not valid
        messagebox.showerror("Error", "Please enter a valid radius.")

# Function to calculate the area of a square
def calculate_square_area():
    # Getting the side length value from the entry widget
    side_str = square_side_entry.get()
    # Checking if the input is a valid number
    if side_str.isdigit():
        # Converting the side length to a float
        side = float(side_str)
        # Calculate the area of the square
        area = side**2
        # Update the result label with the calculated area
        square_result_label.config(text=f"Area: {area} square units",
                                   fg="#9370DB", font=("Roboto", 12, 'bold'))
    else:
        # Show an error message if the input is not valid
        messagebox.showerror("Error", "Please enter a valid side length.")

def calculate_rectangle_area():
    # Getting the length and width values from the entry widgets
    length_str = rectangle_length_entry.get()
    width_str = rectangle_width_entry.get()
    # Checking if the inputs are valid numbers
    if length_str.isdigit() and width_str.isdigit():
        # Converting the length and width to floats
        length = float(length_str)
        width = float(width_str)
        # Calculate the area of the rectangle
        area = length * width
        # Update the result label with the calculated area
        rectangle_result_label.config(text=f"Area:
        {area} square units", fg="#4169E1", font=("Roboto", 12, 'bold'))
    else:
        # Show an error message if the inputs are not valid
        messagebox.showerror("Error", "Please enter valid length and width.")
# Creating main window
root = Tk()

# Setting the title for the window
root.title("Area Calculator")

# Setting the window size
root.geometry("400x300")

# Creating a notebook (tabbed interface)
notebook = ttk.Notebook(root)

# Tab for Circle
circle_tab = Frame(notebook, bg="#E6E6FA")
notebook.add(circle_tab, text='Circle')

# Widgets for Circle tab
# Label for entering the radius
label_radius = Label(circle_tab, text='Enter Radius:',
                     font=('Roboto', 12, 'bold'), bg="#E6E6FA")
label_radius.pack(pady=5)
# Entry widget for entering the radius
circle_radius_entry = Entry(circle_tab, font=("Helvetica", 12))
circle_radius_entry.pack(pady=5)
# Button to calculate the circle area
button_calculate_circle = Button(circle_tab, text='Calculate',
command=calculate_circle_area, font=('Roboto', 12, 'bold'), bg="#9370DB", fg="white")
button_calculate_circle.pack(pady=10)
# Result label for displaying the circle area
circle_result_label = Label(circle_tab, text='Area:',
                            font=('Roboto', 12, 'bold'), bg="#E6E6FA")
circle_result_label.pack()

# Tab for Square
square_tab = Frame(notebook, bg="#87CEEB")
notebook.add(square_tab, text='Square')

# Widgets for Square tab
# Label for entering the side length
label_square_side = Label(square_tab,
                  text='Enter Side Length:',font=('Roboto', 12, 'bold'), bg="#87CEEB")
label_square_side.pack(pady=5)
# Entry widget for entering the side length
square_side_entry = Entry(square_tab, font=('Roboto', 12, 'bold'))
square_side_entry.pack(pady=5)
# Button to calculate the square area
button_calculate_square = Button(square_tab, text='Calculate',
    command=calculate_square_area,font=('Roboto', 12, 'bold'), bg="#4169E1", fg="white")
button_calculate_square.pack(pady=10)
# Result label for displaying the square area
square_result_label = Label(square_tab, text='Area:',
                            font=('Roboto', 12, 'bold'), bg="#87CEEB")
square_result_label.pack()

# Tab for Rectangle
rectangle_tab = Frame(notebook, bg="#F08080")
notebook.add(rectangle_tab, text='Rectangle')

# Widgets for Rectangle tab
# Label for entering the length
label_rectangle_length = Label(rectangle_tab, text='Enter Length:',
                               font=('Roboto', 12, 'bold'), bg="#F08080")
label_rectangle_length.pack(pady=5)
# Entry widget for entering the length
rectangle_length_entry = Entry(rectangle_tab, font=('Roboto', 12, 'bold'))
rectangle_length_entry.pack(pady=5)
# Label for entering the width
label_rectangle_width = Label(rectangle_tab, text='Enter Width:',
                              font=('Roboto', 12, 'bold'), bg="#F08080")
label_rectangle_width.pack(pady=5)
# Entry widget for entering the width
rectangle_width_entry = Entry(rectangle_tab, font=('Roboto', 12, 'bold'))
rectangle_width_entry.pack(pady=5)
# Button to calculate the rectangle area
button_calculate_rectangle = Button(rectangle_tab, text='Calculate',
command=calculate_rectangle_area, font=('Roboto', 12, 'bold'), bg="#CD5C5C", fg="white")
button_calculate_rectangle.pack(pady=10)
# Result label for displaying the rectangle area
rectangle_result_label = Label(rectangle_tab, text='Area:', font=('Roboto', 12, 'bold'),
                               bg="#F08080")
rectangle_result_label.pack()

# Pack for the notebook
notebook.pack(fill=BOTH, expand=YES)

# Starting the Tkinter event loop
root.mainloop()

# End marker
```

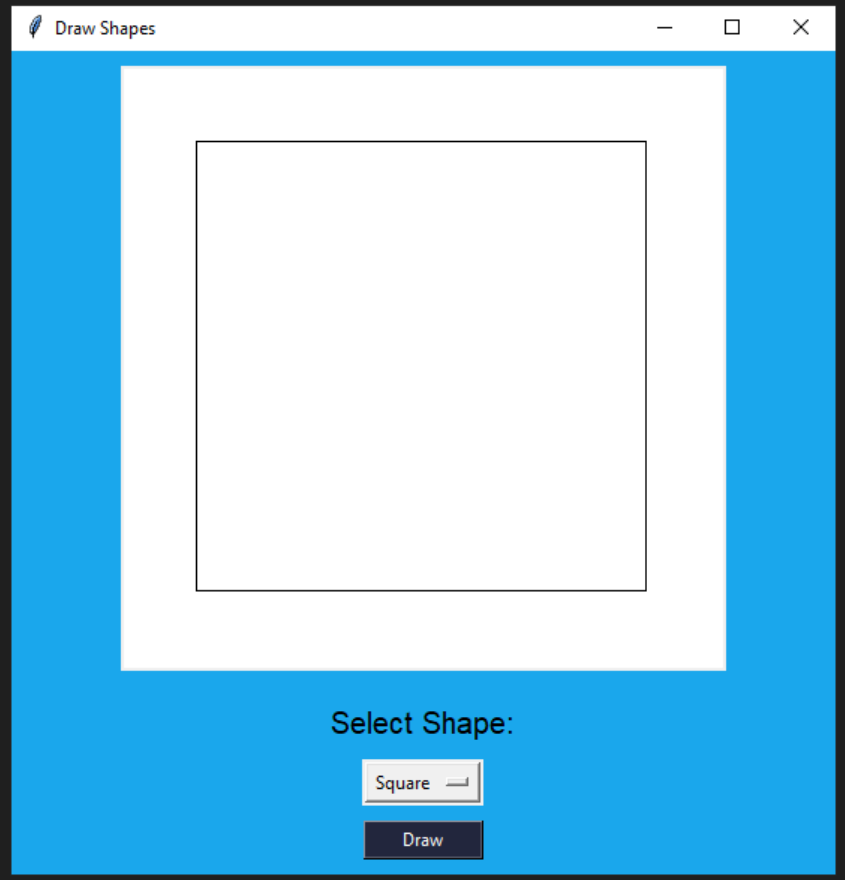
Chapter 3 Ex 4 - Draw Shape

This code demonstrates a program using the tkinter library to draw shapes based on user input. The main window includes a canvas for drawing, a dropdown menu to select a shape, and a button to initiate the drawing.

Code

```python
# Using the tkinter library
from tkinter import *

# Making function
def draw_shape():
    # Getting the selected shape
    shape = shape_var.get()
    # used to clear the canvas before drawing a new shape
    canvas.delete("all")
    # Drawing the selected shape on the canvas
    if shape == "Oval":
        canvas.create_oval(50, 50, 350, 350, outline="black")
    elif shape == "Rectangle":
        canvas.create_rectangle(50, 50, 250, 350, outline="black")
    elif shape == "Square":
        canvas.create_rectangle(50, 50, 350, 350, outline="black")
    elif shape == "Triangle":
        canvas.create_polygon(50, 350, 200, 50, 350, 350, outline="black")

# Creating the main window
root = Tk()

# Setting the window size
root.geometry("550x550")

# Setting the title for the window
root.title("Draw Shapes")

# Setting the background color of the main window
root.configure(bg="#1aa7ec")

# Creating canvas for drawing shapes
canvas = Canvas(root, width=400, height=400, bg="white")
canvas.pack(pady=10)

# Label for selecting the shape
shape_label = Label(root, text="Select Shape:",bg="#1aa7ec",font=("Roboto", 15))  # Creating a l
shape_label.pack(pady=10)  # Pack the label into the main window

# List of shapes for the dropdown menu
shapes = ["Oval", "Rectangle", "Square", "Triangle"]

# Variable to store the selected shape default is the first shape in the list
shape_var = StringVar(value=shapes[0])  # Creating a StringVar and set its initial value to the

# Dropdown menu for selecting the shape
shape_dropdown = OptionMenu(root, shape_var, *shapes)  # Creating an OptionMenu with the shape_v
shape_dropdown.pack()  # Pack the dropdown menu into the main window

# Button to draw the selected shape
draw_button = Button(root, text="Draw", command=draw_shape, width=10, bg="#22263d", fg="white")
draw_button.pack(pady=10)

# Starting the Tkinter event loop
root.mainloop()

# End marker
```

Output



Chapter 3 Bonus - Burger Shack Vendor

Simulates a burger shack vendor program. Users choose burgers, customize, and view transactions. Provides a transaction history with total cost and remaining balance. Python code, console output, and comments are detailed below.

Code

```python
# Chapter 3  Bonus Assessment Exercise - Burger Shack Vendor

# using the time module
import time

# Burger Shack
print(" **WELCOME TO BURGER SHACK** ")

# Defining the prices of each burger
Beef_Burger = 10.00
Chicken_Burger = 8.00
Vegetarian_Burger = 6.50

time.sleep(1)
print("\nThere are 3 delicious burgers to choose from:\n")
time.sleep(2)

# Displaying various options and the pricing for each burger to the user.
print(" Burger                    Costs\n"
" ------                      ------\n"
" 1. Beef Burger            10.00 Dhs\n "
" 2. Chicken Burger          8.00 Dhs\n "
" 3. Vegetarian Burger       6.50 Dhs\n "
"-----------------------------------\n"
)

time.sleep(1)

while True:
    try:
        # Getting the user's input
        money = int(input("Please Enter The Amount of Money: "))
        break
    except ValueError:
        # Displaying an error message if the input is not a valid integer
        print("Sorry, that is not a valid amount. Please enter a whole number (e.g. 10, 100, 1000).")

# Printing line to separate
print("-" * 40)
time.sleep(1)
print("The amount of money you have entered is: " + str(money) + " Dhs")
print("-" * 40)
time.sleep(1)

# To store the items selected by the user
transactions = []

# Function to store the transaction history
def vend(item, price, quantity, total_cost):
    # Calculate the total cost of the purchase
    total_cost = price * quantity
    # Adding the transaction to the history
    transactions.append((item, price, quantity, total_cost))

def get_transaction_history():
    # Returning the transaction history
    return transactions

# Loop until the user has no more money or chooses to exit
while money > 0:
    while True:
        try:
            burger_choice = int(input("\nPlease enter your choice (Once you are finish type 0 to Quit): "))
            if burger_choice in [0, 1, 2, 3]:
                break
            else:
                print("Sorry, that is not a valid option. Please select a number between 1 and 3.")
        except ValueError:
            print("Sorry, that is not a valid option. Please enter a number between 1 and 3.")

    # To Check which burger the user has chosen
    if burger_choice == 0:
        break
    elif burger_choice == 1:
        burger_type = "Beef Burger"
        burger_price = Beef_Burger
    elif burger_choice == 2:
        burger_type = "Chicken Burger"
        burger_price = Chicken_Burger
    elif burger_choice == 3:
        burger_type = "Vegetarian Burger"
        burger_price = Vegetarian_Burger

    # To Check if the user has enough money for the chosen burger
    if money < burger_price:
        print(f"Insufficient funds to purchase {burger_type}")
    else:
        time.sleep(1)
        print(f"\nYou have selected {burger_type}")
        # Loop until the user enters a valid quantity
        while True:
            try:
                # Prompt the user to specify the quantity of burgers they want
                quantity = int(input(f"How many {burger_type}s would you like to purchase? "))
                total_cost = burger_price * quantity

                # Checking if the total cost of the purchase is greater than the user's balance
                if total_cost > money:
                    print("You do not have enough money to make this purchase. Please enter a lower quantity.")
                else:
                    # Calculating the total cost of the purchase
                    money = money - total_cost
                    # Adding the purchase to the transaction history
                    vend(burger_type, burger_price, quantity, total_cost)
                    print(
                        f"Thank you for your purchase! Your remaining balance is: {money} Dhs"
                    )
                    # Printing line to separate
                    print("-" * 40)
                    # Break out of the loop
                    break
            except ValueError:
                print("Sorry, that is not a valid option. Please enter a number.")
```

Chapter 3 Bonus - Burger Shack Vendor

Simulates a burger shack vendor program. Users choose burgers, customize, and view transactions. Provides a transaction history with total cost and remaining balance. Python code, console output, and comments are detailed below.

Code

```python
        # Offer additional items to add to the purchase
        print("\nWould you like to customize your burger with toppings and condiments?")
        print("1. Add Toppings")
        print("2. Add Condiments")
        print("3. Skip customization")

        while True:
            try:
                customization_choice = int(input("Enter your choice (1-3): "))
                if customization_choice in [1, 2, 3]:
                    break
                else:
                    print("Invalid choice. Please select a valid option.")
            except ValueError:
                print("Invalid input. Please enter a number.")

        if customization_choice == 1:
            # Adding toppings
            print("\nAvailable Toppings:")
            print("1. Cheese (1.50 Dhs)")
            print("2. Peanut Butter (1.00 Dhs)")
            print("3. Avocado (2.00 Dhs)")

            while True:
                try:
                    topping_choice = int(input("Select a topping (1-3): "))
                    if 1 <= topping_choice <= 3:
                        toppings = ["Cheese", "Peanut Butter", "Avocado"]
                        selected_topping = toppings[topping_choice - 1]
                        topping_price = {"Cheese": 1.50, "Peanut Butter": 1.00, "Avocado": 2.00}[selected_topping]

                        # Adding the selected topping to the burger
                        money -= topping_price
                        print(f"{selected_topping} added to your burger. Remaining balance: {money} Dhs")
                        vend(selected_topping, topping_price, 1, topping_price)

                        break
                    else:
                        print("Invalid choice. Please select a valid option.")
                except ValueError:
                    print("Invalid input. Please enter a number.")

        elif customization_choice == 2:
            # Adding condiments
            print("\nAvailable Condiments:")
            print("1. Ketchup (0.50 Dhs)")
            print("2. Mayonnaise (0.75 Dhs)")
            print("3. BBQ Sauce (1.00 Dhs)")

            while True:
                try:
                    condiment_choice = int(input("Select a condiment (1-3): "))
                    if 1 <= condiment_choice <= 3:
                        condiments = ["Ketchup", "Mayonnaise", "BBQ Sauce"]
                        selected_condiment = condiments[condiment_choice - 1]
                        condiment_price = {"Ketchup": 0.50, "Mayonnaise": 0.75, "BBQ Sauce": 1.00}[selected_condiment]

                        # Adding the selected condiment to the burger
                        money -= condiment_price
                        print(f"{selected_condiment} added to your burger. Remaining balance: {money} Dhs")
                        vend(selected_condiment, condiment_price, 1, condiment_price)

                        break
                    else:
                        print("Invalid choice. Please select a valid option.")
                except ValueError:
                    print("Invalid input. Please enter a number.")
```

```python
        # Adding sides
        print("\nAvailable Sides:")
        print("1. Fries (3.00 Dhs)")
        print("2. Drink (2.50 Dhs)")

        while True:
            try:
                side_choice = int(input("Select a side (1-2): "))
                if 1 <= side_choice <= 2:
                    sides = ["Fries", "Drink"]
                    selected_side = sides[side_choice - 1]
                    side_price = {"Fries": 3.00, "Drink": 2.50}[selected_side]

                    # Adding the selected side to the order
                    money -= side_price
                    print(f"{selected_side} added to your order. Remaining balance: {money} Dhs")
                    vend(selected_side, side_price, 1, side_price)

                    break
                else:
                    print("Invalid choice. Please select a valid option.")
            except ValueError:
                print("Invalid input. Please enter a number.")

    time.sleep(1)
    # Asking the user if they want to make an additional selection
    additional_selection = input('Would you like to make another purchase? (y/n): ')
    if additional_selection.lower() == 'no' or additional_selection.lower() == 'n':
        break
    else:
      continue

# Displaying the transaction history and total
print("\n-----\033[92mTransaction History\033[0m-----")
history = get_transaction_history()
total = 0

for item, price, quantity, total_cost in history:
    total += total_cost
    print(f"\n{item}: {price} Dhs x {quantity} = {total_cost} Dhs")

print(f"\nTotal: {total} Dhs ")
print(f"\nRemaining balance: {money} Dhs")
print("\nEnjoy your meal at Burger Shack!")

# end marker
```

Chapter 3 Bonus - Burger Shack Vendor

Simulates a burger shack vendor program. Users choose burgers, customize, and view transactions. Provides a transaction history with total cost and remaining balance. Python code, console output, and comments are detailed below.

Output

```
 **WELCOME TO BURGER SHACK**

There are 3 delicious burgers to choose from:

 Burger                    Costs
 ------                    ------
 1. Beef Burger            10.00 Dhs
  2. Chicken Burger          8.00 Dhs
  3. Vegetarian Burger       6.50 Dhs
 ------------------------------------

Please Enter The Amount of Money: 12
---------------------------------------
The amount of money you have entered is: 12 Dhs
---------------------------------------

Please enter your choice (Once you are finish type 0 to Quit): 2

You have selected Chicken Burger
How many Chicken Burgers would you like to purchase? 3
You do not have enough money to make this purchase. Please enter a lower quantity.
How many Chicken Burgers would you like to purchase? 1
Thank you for your purchase! Your remaining balance is: 4.0 Dhs
---------------------------------------

Would you like to customize your burger with toppings and condiments?
1. Add Toppings
2. Add Condiments
3. Skip customization
Enter your choice (1-3): 2

Available Condiments:
1. Ketchup (0.50 Dhs)
2. Mayonnaise (0.75 Dhs)
3. BBQ Sauce (1.00 Dhs)
Select a condiment (1-3): 3
BBQ Sauce added to your burger. Remaining balance: 3.0 Dhs

Available Sides:
1. Fries (3.00 Dhs)
2. Drink (2.50 Dhs)
Select a side (1-2): 2
Drink added to your order. Remaining balance: 0.5 Dhs
Would you like to make another purchase? (y/n): n
```

```
-----Transaction History-----

Chicken Burger: 8.0 Dhs x 1 = 8.0 Dhs

BBQ Sauce: 1.0 Dhs x 1 = 1.0 Dhs

Drink: 2.5 Dhs x 1 = 2.5 Dhs

Total: 11.5 Dhs

Remaining balance: 0.5 Dhs

Enjoy your meal at Burger Shack!
```

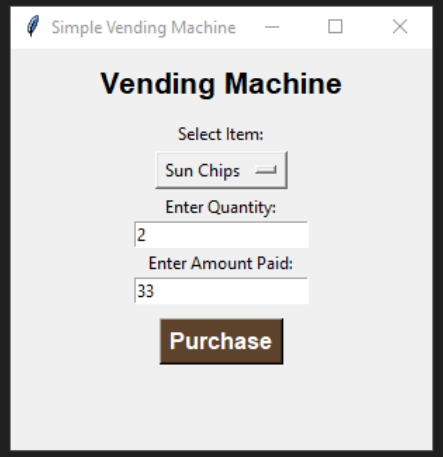
Chapter 3 Further Ex I - Vending Machine

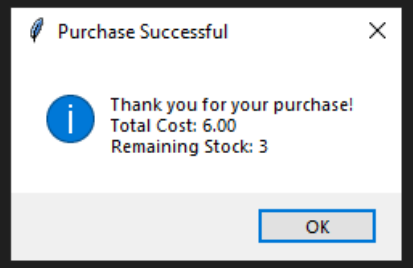
Simple vending machine program using Tkinter and messagebox. Users can select items, enter quantity, and amount paid for a simulated purchase. The program checks stock, calculates the cost, and provides user feedback.

Code

```python
# Chapter 3 Exercise i - Vending machine

# using the tkinter libraryand messagebox
from tkinter import *
from tkinter import messagebox

# Vending machine data
items = ["Snickers Bar", "Sun Chips", "Cold Coffee"]
prices = {"Snickers Bar": 2.00, "Sun Chips": 3.00, "Cold Coffee": 2.50}
stock = {"Snickers Bar": 10, "Sun Chips": 5, "Cold Coffee": 8}

def process_purchase():
    # Getting user input: selected item, quantity, and amount paid
    item = selected_item.get()
    quantity = int(quantity_var.get())
    amount_paid = amount_paid_var.get()

    # Calculates the total cost
    if item in prices:
        price = prices[item]
        current_stock = stock[item]
    else:
        # Show an error message for invalid item selection
        messagebox.showerror("Invalid Item", "Please select a valid item.")
        return

    total_cost = price * quantity

    # Checking if there is enough stock
    if quantity > current_stock:
        messagebox.showerror("Out of Stock", f"We're out of stock for {item}. Please enter a lower quantity.")
        return

    # Checking if the amount paid is sufficient
    if amount_paid >= total_cost:
        remaining_stock = current_stock - quantity
        messagebox.showinfo("Purchase Successful", f"Thank you for your purchase!\nTotal Cost:
                            {total_cost:.2f}\nRemaining Stock: {remaining_stock}")
        # Updating the stock
        stock[item] = remaining_stock
    else:
        messagebox.showerror("Insufficient Funds", "You have not paid enough for your purchase.")

# Creating the main window
root = Tk()

# Setting the title for the window
root.title("Simple Vending Machine")

# Setting the window size
root.geometry("300x300")

# Variables
selected_item = StringVar()

# Default selection
selected_item.set(items[0])
quantity_var = StringVar()
amount_paid_var = DoubleVar(value='')

# Labels
welcome_label = Label(root, text="Vending Machine", font=("Roboto", 16, "bold"))
welcome_label.pack(pady=10)

item_label = Label(root, text="Select Item:")
item_label.pack()

# Option menu for selecting item
item_option = OptionMenu(root, selected_item, *items)
item_option.pack()

quantity_label = Label(root, text="Enter Quantity:")
quantity_label.pack()

# Entry for entering quantity
quantity_entry = Entry(root, textvariable=quantity_var)
quantity_entry.pack()

amount_paid_label = Label(root, text="Enter Amount Paid:")
amount_paid_label.pack()

# Entry for entering amount paid
amount_paid_entry = Entry(root, textvariable=amount_paid_var)
amount_paid_entry.pack()

# Button to process the purchase
purchase_button = Button(root, text="Purchase", command=process_purchase, bg="#5d432c",
                         fg="white", font=("Roboto", 12, 'bold'))
purchase_button.pack(pady=10)

# Starting the Tkinter event loop
root.mainloop()

#End marker
```

Output





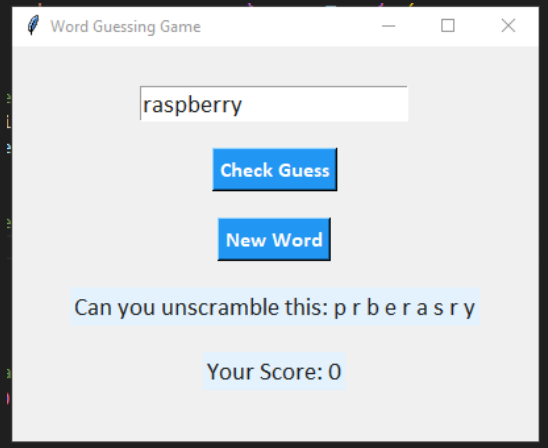
Chapter 3 Further Ex II - Guess Word

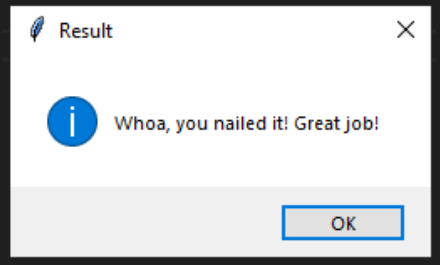
This code uses Tkinter to create a word guessing game. Users unscramble shuffled words, and the program checks their guesses, updates the score, and provides feedback. Code, output, and comments are included for clarity.

Code

```python
# Chapter 3 Exercise ii - Guess word


# using the tkinter library,random and messagebox
from tkinter import *
import random
from tkinter import messagebox

# Global variable to store the current word for the user to guess
current_word = ""

def check_guess():
    # Getting the user's guess from the entry field and convert it to lowercase
    guess = entry.get().lower()

    # Check if the guess is correct
    if guess == current_word.lower():
        messagebox.showinfo('Result', 'Whoa, you nailed it! Great job!')
        update_score()
        new_word()  # Generating a new word for the user to guess
    else:
        # Showing the result message first, then update current_word
        messagebox.showinfo('Result', f"Oops, not quite! The answer is {current_word}.")
        new_word()  # Generating a new word for the user to guess

def update_score():
    # Extracting the current score from the label text and increment it
    current_score = int(score_label["text"].split(":")[-1])
    score_label.config(text=f'Your Score: {current_score + 1}')

def new_word():
    # Declaring current_word as global to update the global variable
    global current_word

    # Selecting a random word from the list
    word = random.choice(words)

    # Shuffling the characters of the word for a bit of fun
    shuffle_word = random.sample(word, len(word))

    # Displaying the shuffled word in the label
    shuffle_label.config(text='Can you unscramble this: ' + ' '.join(shuffle_word))

    current_word = word  # Update the global variable with the new word

# Listing of words for the user to guess
words = ['apple', 'banana', 'cherry', 'date', 'fig', 'grape', 'honeydew', 'kiwi', 'lemon',
         'mango', 'nectarine', 'orange', 'pineapple', 'raspberry', 'strawberry', 'tomato', 'watermelon']

# Creating the main window
root = Tk()

# Setting the title of the window
root.title('Word Guessing Game')

# Setting the window size
root.geometry("400x300")

# Creating a frame within the window with padding
frame = Frame(root, padx=20, pady=20)
frame.pack()

# Creating an entry widget for the user to input their guess
entry = Entry(frame, font=('Calibri', 14), fg='#212121')
entry.pack(pady=10)

# Creating a blue-colored button with a different font for checking the guess
check_button = Button(frame, text='Check Guess', command=check_guess, bg='#2196F3',
                      fg='#FFFFFF', font=('Calibri', 12, 'bold'))
check_button.pack(pady=10)

# Creating a blue-colored button with a different font for getting a new word
new_word_button = Button(frame, text='New Word', command=new_word, bg='#2196F3',
                         fg='#FFFFFF', font=('Calibri', 12, 'bold'))
new_word_button.pack(pady=10)

# Creating a label to display the shuffled word for the user to unscramble
shuffle_label = Label(frame, text='Can you unscramble this: ', font=('Calibri', 14), bg='#E3F2FD', fg='#212121')
shuffle_label.pack(pady=10)

# Creating a label to display the user's current score
score_label = Label(frame, text='Your Score: 0', font=('Calibri', 14), bg='#E3F2FD', fg='#212121')
score_label.pack(pady=10)

# Initializing the current_word variable with a new word
new_word()

# Starting the main event loop
root.mainloop()

# End marker
```

Output





Chapter 4 - File Handling & Regular Expressions

Exercises, Bonus And Further

Chapter 4 Ex 1 - User information

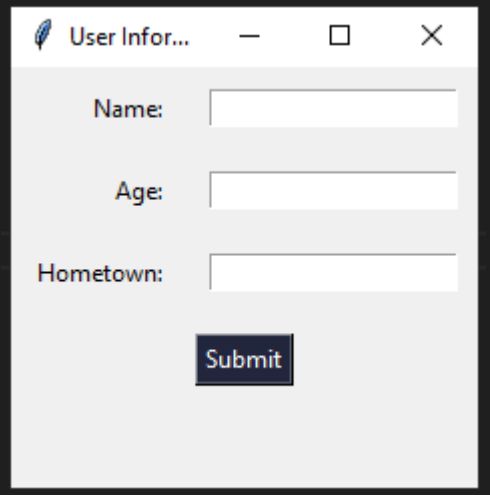
This tkinter-based code captures user info (name, age, hometown), saving it to 'bio.txt'. Functions handle data. See code and output

Code

```python
# Chapter 4 Exercise 1 - User information

# using the tkinter library
from tkinter import *

# Making fucntion to read & write data
def save_to_file(name, age, hometown):
    # Saving user information to the bio.txt file
    with open('bio.txt', 'w') as file:
        file.write(f"Name: {name}\n")
        file.write(f"Age: {age}\n")
        file.write(f"Hometown: {hometown}\n")

def read_from_file():
    try:
        # Reading user information from the bio.txt file
        with open('bio.txt', 'r') as file:
            content = file.read()
            return content
    except FileNotFoundError:
        return "File not found. Please create your file first."

def submit_form():
    # Getting user input from the entry fields
    name = entry_name.get()
    age = entry_age.get()
    hometown = entry_hometown.get()

    # Saving user information to the file
    save_to_file(name, age, hometown)

    # Displaying the content in a label
    result_label.config(text=read_from_file())

# Creating the main window
root = Tk()
```

```python
# Setting the title for the window
root.title("User Information root")

# Creating and place the form elements
label_name = Label(root, text="Name:")   # Name
label_name.grid(row=0, column=0, padx=10, pady=10, sticky=E)
entry_name = Entry(root)
entry_name.grid(row=0, column=1, padx=10, pady=10)

label_age = Label(root, text="Age:")   # Age
label_age.grid(row=1, column=0, padx=10, pady=10, sticky=E)
entry_age = Entry(root)
entry_age.grid(row=1, column=1, padx=10, pady=10)

label_hometown = Label(root, text="Hometown:")  # Hometown
label_hometown.grid(row=2, column=0, padx=10, pady=10, sticky=E)
entry_hometown = Entry(root)
entry_hometown.grid(row=2, column=1, padx=10, pady=10)

# Button to submit the form
submit_button = Button(root, text="Submit", command=submit_form,bg='#22263d',fg='white')
submit_button.grid(row=3, column=0, columnspan=2, pady=10)

# Label to display the result
result_label = Label(root, text="")  # display data
result_label.grid(row=4, column=0, columnspan=2, pady=10)

# Run the main event loop
root.mainloop()

# End Marker
```

Output




```
Name: Asif
Age: 19
Hometown: Dubai
```

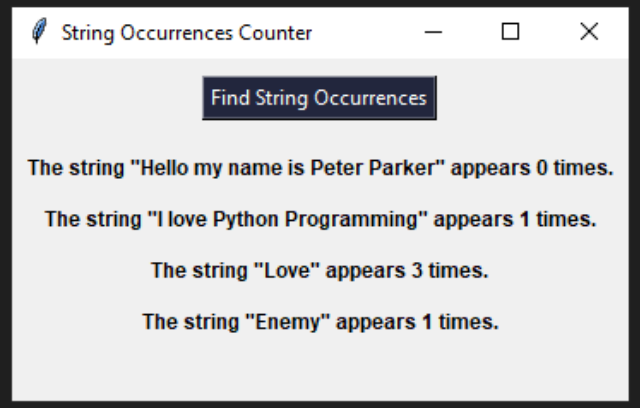
Chapter 4 Ex 2 - CountString

GUI program using tkinter to count occurrences of specified strings in a file.

Code

```python
# Chapter 4 Exercise 2 - CountString

# using the tkinter library
from tkinter import *


def count_string_occurrences(search_string, file_path):
    # Count occurrences of a specified string in a file and return a formatted result.
    # search_string (str): The string to search for.
    # file_path (str): The path to the file containing the text.
    try:
        # to open and read the specified file
        with open('sentences.txt', 'r') as file:
            content = file.read()

            # Counting occurrences of the search string
            count = content.lower().count(search_string.lower())

            # Return the result message
            return f'The string "{search_string}" appears {count} times.\n'
    except FileNotFoundError:
        # to handle the case where the file is not found
        return "File not found. Please make sure the file exists."

def submit_form():
    # Callback function for the submit button.
    # Go through a list of search strings, count their occurrences

    # List of strings to search for
    search_strings = [
        "Hello my name is Peter Parker",
        "I love Python Programming",
        "Love",
        "Enemy"
    ]

    result_text = ""
    # go through the search strings and count occurrences
    for search_string in search_strings:
        result_text += count_string_occurrences(search_string, 'sentences.txt') + '\n'

    # Updating the label with the result text
    result_label.config(text=result_text)

# Creating the main window
root = Tk()

# Setting the window size
root.geometry("360x200")

# Setting the title for the window
root.title("String Occurrences Counter")

# Button to submit the form
submit_button = Button(root, text="Find String Occurrences",
                       command=submit_form,bg='#22263d',fg='white')
submit_button.pack(pady=10)

# Label to display the result
result_label = Label(root, text="",font=("Arial", 9, "bold"))
result_label.pack(pady=10)

# Run the main event loop
root.mainloop()

# End Marker
```

Output



Chapter 4 Ex 3 - Reading to a List

Code reads numbers from numbers.txt, converts them to integers, and outputs the results.

Code

```python
# Chapter 4 Exercise 3 - Reading to a List

# Read numbers from the file and output them as integers

# Open and read the numbers from the file
with open('Chapter 4 - File Handling and Regular Expressions/Exercises/Exercise 3 - Reading to a List/numbers.txt', 'r') as file:
    # Read each line, convert to integer, and store in a list
    numbers_list = [int(line.strip()) for line in file]

# Output the numbers in integer format
print("Numbers in integer format:")
for num in numbers_list:
    print(num)

# End Marker
```

Output

```
Numbers in integer format:
58
19
4
46
52
33
36
96
15
97
37
42
88
83
62
91
79
25
1
10
39
47
16
55
2
69
59
90
43
82
74
80
92
38
12
```

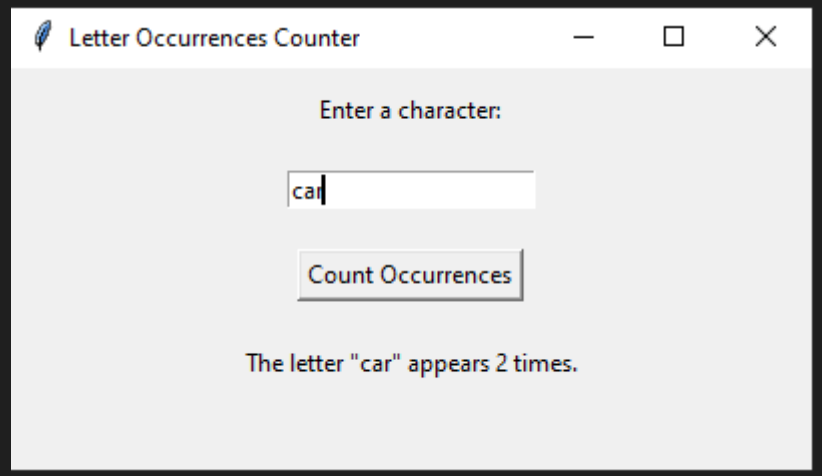
Chapter 4 Ex 4 – Letter Count

GUI program for counting occurrences of a letter in a file.

Code

```python
# Chapter 4 Exercise 3 - letter count

# Importing the tkinter library for GUI
from tkinter import *

# Function to count occurrences of a character in a file
def count_occurrences(character, file_path):
    try:
        # Opening and reading the specified file
        with open('sentences.txt', 'r') as file:
            content = file.read()
            # Counting occurrences of the character (case-insensitive)
            count = content.lower().count(character.lower())
            return f'The letter "{character}" appears {count} times.'
    except FileNotFoundError:
        return "File not found. Please make sure the file exists."

# Function to handle form submission
def submit_form():
    # Getting user input from the entry field
    user_input = entry_character.get()
    # Counting occurrences and updating the result label
    result_text = count_occurrences(user_input, 'sentences.txt')
    result_label.config(text=result_text)

# Creating the main window
root = Tk()

# Setting the window size
root.title("Letter Occurrences Counter")

# Setting the window size
root.geometry("400x200")

# Label and Entry for user input
label_prompt = Label(root, text="Enter a character:")
label_prompt.pack(pady=10)

entry_character = Entry(root)
entry_character.pack(pady=10)

# Button to submit the form
submit_button = Button(root, text="Count Occurrences", command=submit_form)
submit_button.pack(pady=10)

# Label to display the result
result_label = Label(root, text="")
result_label.pack(pady=10)

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

Output



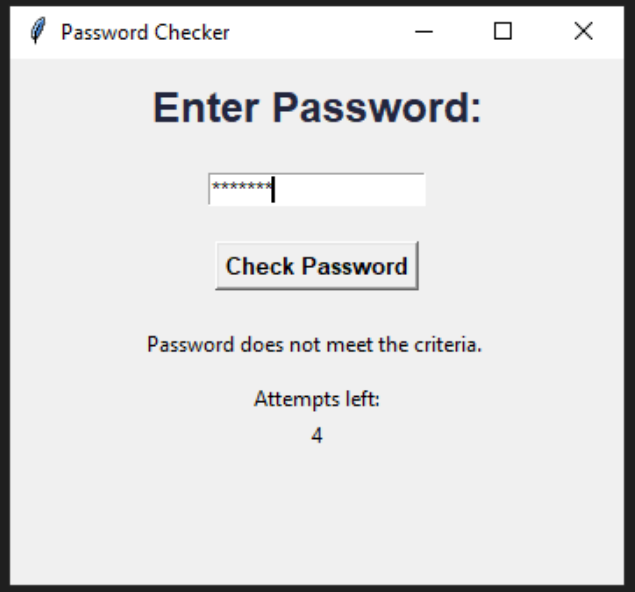
Chapter 4 Ex 5 – Password Check

This code uses tkinter to create a password checking application. Users input a password, its validity is checked based on criteria, and the result is displayed along with remaining attempts.

Code

```python
# Chapter 4 Exercise 5 - Password checker

# Using the tkinter library
from tkinter import *

def check_password(password):
    # Functioning to check the validity of the password
    # Returns a tuple (is_valid, message)

    if not (6 <= len(password) <= 12):
        return False, "Password length must be between 6 and 12 characters.\n"

    # Checking if the password satisfies all the criteria
    has_lowercase = any(char.islower() for char in password)
    has_number = any(char.isdigit() for char in password)
    has_uppercase = any(char.isupper() for char in password)
    has_special_char = any(char in "$#@ " for char in password)

    if has_lowercase and has_number and has_uppercase and has_special_char:
        return True, "Password is valid."
    else:
        return False, "Password does not meet the criteria."

def submit_form():
    # Function to handle form submission
    password = entry_password.get()
    attempts_left = int(attempts_var.get())

    if attempts_left > 0:
        is_valid, message = check_password(password)

        if is_valid:
            result_label.config(text="Password is valid.")
            root.destroy()  # Close the window if the password is valid
        else:
            attempts_left -= 1
            result_text = f"{message} "

            if attempts_left == 0:
                result_text += "\nThe authorities have been alerted!"
                submit_button.config(state=DISABLED)  # Disable the submit button

            result_label.config(text=result_text)
            attempts_var.set(str(attempts_left))
    else:
        result_label.config(text="No attempts left. The authorities have been alerted!")

# Creating the main window
root = Tk()

# Setting the title for the window
root.title("Password Checker")

# Setting the window size
root.geometry("350x300")

# Variables
attempts_var = StringVar()
attempts_var.set("5")

# Label and Entry for password input
label_password = Label(root, text="Enter Password:")
label_password.pack(pady=10)

# to show the password as *
entry_password = Entry(root, show="*")
entry_password.pack(pady=10)

# Button to submit
submit_button = Button(root, text="Check Password", command=submit_form)
submit_button.pack(pady=10)

# Label to display the result and attempts left
result_label = Label(root, text="")
result_label.pack(pady=10)

# Attempts left label
attempts_label = Label(root, text="Attempts left:")
attempts_label.pack()

attempts_display = Label(root, textvariable=attempts_var)
attempts_display.pack()

# Run the main event loop
root.mainloop()

# End marker
```

Output



Chapter 4 Bonus Ex A - Petrol Price

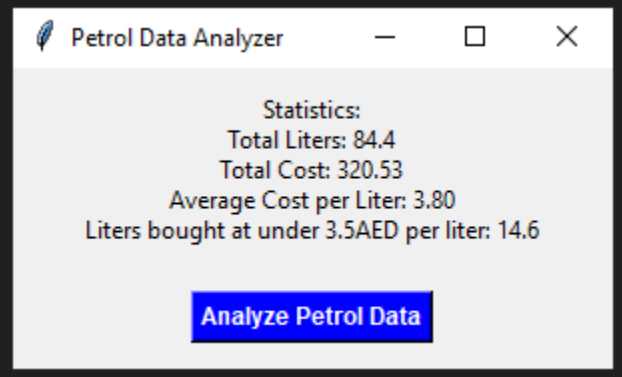
This tkinter program analyzes petrol data from 'petrolPrice.txt' and displays statistics in a GUI window.

Code

```python
# Chapter 4 Exercise A - Petrol Price

# using the tkinter library
from tkinter import *

def analyze_petrol_data():
    try:
        file_path = 'petrolPrice.txt'
        total_liters = 0
        total_cost = 0
        under_3_5_liters = 0

        # Reading the data from the file and process each line
        with open(file_path, 'r') as file:
            # Skip the header line
            next(file)

            for line in file:
                # Spliting the line into columns
                columns = line.strip().split('\t')

                # Extracting liters and cost from the columns
                liters, cost = float(columns[0]), float(columns[1])

                # to add totals
                total_liters += liters
                total_cost += cost

                # Checking if the cost per liter is under 3.5AED
                if liters > 0 and cost / liters < 3.5:
                    under_3_5_liters += liters

        # Calculating average cost per liter
        if total_liters > 0:
            average_cost_per_liter = total_cost / total_liters
        else:
            average_cost_per_liter = 0

        # Display statistics
        result_text = f"Statistics:\nTotal Liters: {total_liters}\nTotal Cost: {total_cost}\n"
        result_text += f"Average Cost per Liter: {average_cost_per_liter:.2f}\n"
        result_text += f"Liters bought at under 3.5AED per liter: {under_3_5_liters}"
        result_label.config(text=result_text)

    except FileNotFoundError:
        result_label.config(text="File not found. Please make sure the file exists.")
```

```python
# Creating the main window
root = Tk()

# Creating a title of page
root.title("Petrol Data Analyzer")

# Setting the window size
root.geometry('300x150')

# Label for displaying the result
result_label = Label(root, text="")
result_label.pack(pady=10)

# Button to analyze the petrol data with a blue color
analyze_button = Button(root, text="Analyze Petrol Data", command=analyze_petrol_data, bg="blue",
                        fg="white",font=('Roboto', 9, 'bold'))
analyze_button.pack(pady=10)

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

Output



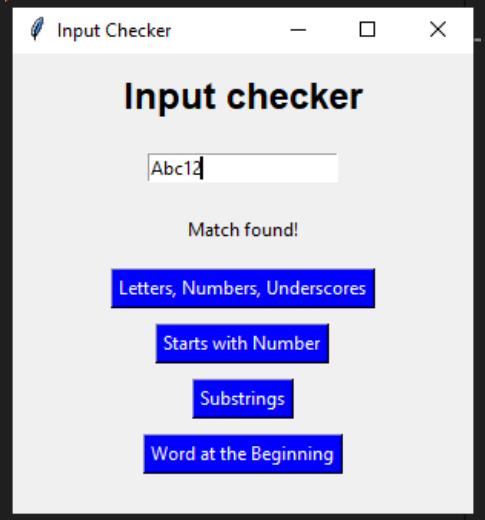
Chapter 4 Further Ex I - Check Input

This tkinter program checks user input for conditions like alphanumeric characters, starting with a number, containing a specific substring, and starting with a particular word, providing feedback for each.

Code

```python
# Chapter 4 Exercise I - Check Input

# using the tkinter library
from tkinter import *

# Function to check if the input contains only letters, numbers, and underscores
def check_letters_numbers_underscores(input_text):
    # Matching a string that contains only upper and lowercase letters, numbers, and underscores
    return "Match found!" if input_text.isalnum() or '_' in input_text else "No match found."

# Function to check if the input starts with a number
def check_start_with_number(input_text):
    # Matching a string that starts with a specific number
    return "Match found!" if input_text[0].isdigit() else "No match found."

# Function to check if the input contains a specific substring
def check_substrings(input_text):
    # Find substrings within a string
    return "Match found!" if "substring" in input_text else "No match found."

# Function to check if the input starts with a specific word
def check_word_at_beginning(input_text):
    # Match a word at the beginning of a string
    return "Match found!" if input_text.startswith("Word") else "No match found."

# Function to handle button click
def on_button_click(check_function, entry_widget, result_label):
    input_text = entry_widget.get()
    result_text = check_function(input_text)
    result_label.config(text=result_text)

# Creating the main window
root = Tk()

root.title("Input Checker")
root.geometry('300x300')

# The heading label
welcome_label = Label(root, text="Input checker", font=("Roboto", 18, "bold"))
welcome_label.pack(pady=10)

# Entry for user input
input_entry = Entry(root)
input_entry.pack(pady=10)

# Result label
result_label = Label(root, text="")
result_label.pack(pady=10)

# Button for "Letters, Numbers, Underscores"
button1 = Button(root, text="Letters, Numbers, Underscores",
 command=lambda: on_button_click(check_letters_numbers_underscores, input_entry, result_label), bg="blue", fg="white")
button1.pack(pady=5)

# Button for "Starts with Number"
button2 = Button(root, text="Starts with Number",
 command=lambda: on_button_click(check_start_with_number, input_entry, result_label), bg="blue", fg="white")
button2.pack(pady=5)

# Button for "Substrings"
button3 = Button(root, text="Substrings",
 command=lambda: on_button_click(check_substrings, input_entry, result_label), bg="blue", fg="white")
button3.pack(pady=5)

# Button for "Word at the Beginning"
button4 = Button(root, text="Word at the Beginning",
 command=lambda: on_button_click(check_word_at_beginning, input_entry, result_label), bg="blue", fg="white")
button4.pack(pady=5)

# Run the main event loop
root.mainloop()

# End marker
```

Output



Chapter 5 - Object Oriented Programming

Exercises, Bonus And Further

Chapter 5 Ex 1 - Woof Woof

This code showcases a simple Tkinter GUI displaying information about two dogs and making the oldest dog woof.It defines a Dog class, retrieves and displays dog information, and creates a GUI to showcase the details.

Code

```python
# Chapter 5 Exercise 1 - Woof Woof

# using the tkinter library and messagebox
from tkinter import *
from tkinter import messagebox


# Defining a simple Dog class
class Dog:
    name = ""
    age = 0
    breed = ""

# Function to get information about a dog
def info(dog):
    return f"{dog.name} is {dog.age} years old and is a {dog.breed}."

# Function to make a dog woof
def woof(dog):
    return f"{dog.name} says: Woof!"

# Creating two dog objects
dog1 = Dog()
dog1.name = "Buddy"
dog1.age = 3
dog1.breed = "Labrador"

dog2 = Dog()
dog2.name = "Max"
dog2.age = 5
dog2.breed = "Golden Retriever"

# Display the information of each dog in the console
print(info(dog1))
print(info(dog2))

# Finding the oldest dog and make it woof
oldest_dog = dog1 if dog1.age > dog2.age else dog2
print(woof(oldest_dog))
```

```python
# Creating a simple Tkinter GUI to display the dog information
root = Tk()
root.title('Dog Information')

# Setting the dimensions of the window
root.geometry('350x150')

# Creating the main heading
head1 = Label(root, font=('Roboto', 22, 'bold'), text="Dog Information", fg='#4CAF50', bg="#f2f2f2")
head1.pack(pady=10)

# Labels to display information about each dog and the woof of the oldest dog
label1 = Label(root, text=info(dog1), font=("Arial", 10, "bold"))
label1.pack()

label2 = Label(root, text=info(dog2), font=("Arial", 10, "bold"))
label2.pack()

oldest_dog_label = Label(root, text=woof(oldest_dog), font=("Arial", 10, "bold"))
oldest_dog_label.pack()

# Run the main event loop
root.mainloop()

# End Marker
```

Output

Chapter 5 Ex 2 - Student Class

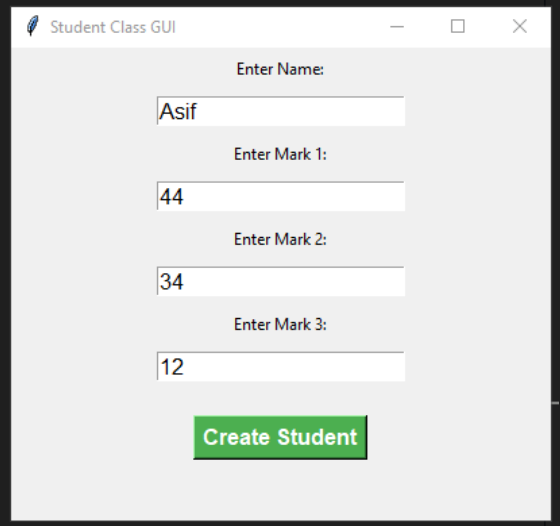
Tkinter GUI app for a Student class. Collects user input (name, marks), calculates average grade, and displays info with a messagebox.

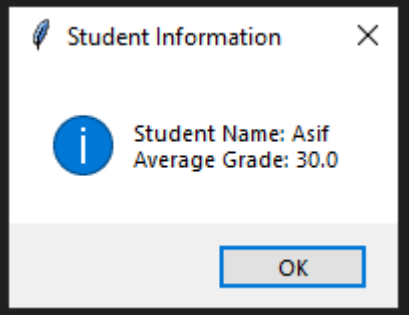
Code

```python
# Chapter 5 Exercise 2 - Student Class

# using the tkinter library and messagebox
from tkinter import *
from tkinter import messagebox

# Defining a class for representing a student
class Student:
    # Method to set data for a student
    def set_data(student, name, mark1, mark2, mark3):
        student.name = name
        student.mark1 = mark1
        student.mark2 = mark2
        student.mark3 = mark3

    # Method to calculate the average grade of a student
    def calc_grade(student):
        # Calculate the average of the three marks
        return (student.mark1 + student.mark2 + student.mark3) / 3

    # Method to display student information
    def display(student):
        # Return a formatted string with student name and average grade
        return f"Student Name: {student.name}\nAverage Grade: {student.calc_grade():}"

# Function to create a student object and display information
def create_student_object():
    # Getting values from entry boxes
    name = name_entry.get()
    mark1 = float(mark1_entry.get())
    mark2 = float(mark2_entry.get())
    mark3 = float(mark3_entry.get())

    # Creating a Student object with the provided values
    student_object = Student()
    student_object.set_data(name, mark1, mark2, mark3)

    # Displaying the student information using a messagebox popup
    messagebox.showinfo("Student Information", student_object.display())

# Creating the main window
root = Tk()
```

```python
# Setting the title for the window
root.title("Student Class GUI")

# Setting the window size
root.geometry("400x350")

# Labels and Entry Boxes
name_label = Label(root, text="Enter Name:") # Asking Name
name_label.pack(pady=5)
name_entry = Entry(root, font=("Helvetica", 12))
name_entry.pack(pady=5)

mark1_label = Label(root, text="Enter Mark 1:") # Asking Mark 1
mark1_label.pack(pady=5)
mark1_entry = Entry(root, font=("Helvetica", 12))
mark1_entry.pack(pady=5)

mark2_label = Label(root, text="Enter Mark 2:") # Asking Mark 2
mark2_label.pack(pady=5)
mark2_entry = Entry(root, font=("Helvetica", 12))
mark2_entry.pack(pady=5)

mark3_label = Label(root, text="Enter Mark 3:") # Asking Mark 3
mark3_label.pack(pady=5)
mark3_entry = Entry(root, font=("Helvetica", 12))
mark3_entry.pack(pady=5)

# Button to create a student object and display information
create_student_button = Button(root, text="Create Student", command=create_student_object,
    bg="#4CAF50", fg="white", font=("Roboto", 12, 'bold'))
create_student_button.pack(pady=20)

# Run the main event loop
root.mainloop()

# End Marker
```

Output





Chapter 5 Ex 3 - Employee Class

Tkinter GUI app for a Student class. Collects user input (name, marks), calculates average grade, and displays info with a messagebox.

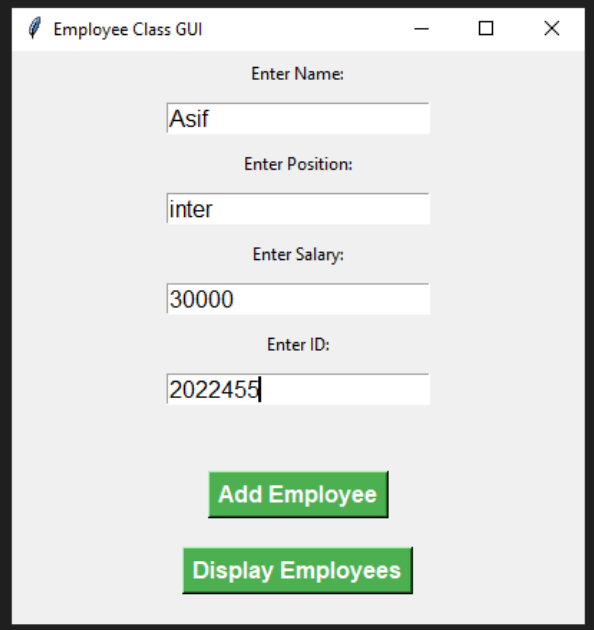
Code

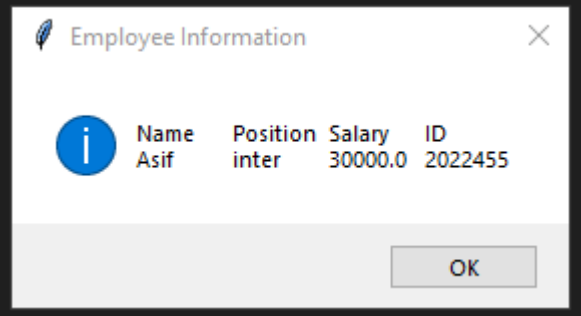
```python
# Chapter 5 Exercise 3 - Employee Class

# using the tkinter library and messagebox
from tkinter import *
from tkinter import messagebox

# Defining a class for representing an employee
class Employee:
    # Method to set data for an employee
    def set_data(employee, name, position, salary, employee_id):
        employee.name = name
        employee.position = position
        employee.salary = salary
        employee.employee_id = employee_id

    # Method to get data for an employee
    def get_data(employee):
        # Return a formatted string with employee data
        return f"{employee.name}\t{employee.position}\t{employee.salary}\t{employee.employee_id}"

# Function to create an employee object and display information
def add_employee():
    # Get values from entry boxes
    name = name_entry.get()
    position = position_entry.get()

    # Handle potential ValueError for invalid input for salary
    try:
        salary = float(salary_entry.get())
    except ValueError:
        # Display an error message in red
        error_label.config(text="Invalid input for Salary. Please enter a valid number.", fg="red")
        return

    # Get employee ID as an integer
    try:
        employee_id = int(id_entry.get())
    except ValueError:
        # Display an error message in red
        error_label.config(text="Invalid input for ID. Please enter a valid number.", fg="red")
        return

    # Clear the error label
    error_label.config(text="")

    # Create an Employee object with the provided values
    employee_object = Employee()
    employee_object.set_data(name, position, salary, employee_id)

    # Add the employee object to the list
    employees.append(employee_object)

    # Clear the entry boxes
    name_entry.delete(0, END)
    position_entry.delete(0, END)
    salary_entry.delete(0, END)
    id_entry.delete(0, END)

# Function to display employee information in the console
def display_employees():
    # Display header
    result_text = "Name\tPosition\tSalary\tID\n"

    # Iterate through the list of employees and append their data
    for employee in employees:
        result_text += employee.get_data() + "\n"

    # Display the result in a messagebox popup
    messagebox.showinfo("Employee Information", result_text)

# Creating the main window
root = Tk()

# Setting the title for the window
root.title("Employee Class GUI")

# Setting the window size
root.geometry("400x400")

# List to store employee objects
employees = []

# Labels and Entry Boxes
name_label = Label(root, text="Enter Name:")
name_label.pack(pady=5)
name_entry = Entry(root, font=("Helvetica", 12))
name_entry.pack(pady=5)

position_label = Label(root, text="Enter Position:")
position_label.pack(pady=5)
position_entry = Entry(root, font=("Helvetica", 12))
position_entry.pack(pady=5)

salary_label = Label(root, text="Enter Salary:")
salary_label.pack(pady=5)
salary_entry = Entry(root, font=("Helvetica", 12))
salary_entry.pack(pady=5)

id_label = Label(root, text="Enter ID:")
id_label.pack(pady=5)
id_entry = Entry(root, font=("Helvetica", 12))
id_entry.pack(pady=5)

# Error label for displaying error messages
error_label = Label(root, text="", fg="red")
error_label.pack(pady=5)

# Button to add an employee and display information
add_employee_button = Button(root, text="Add Employee",
            command=add_employee, bg="#4CAF50", fg="white", font=("Roboto", 12, 'bold'))
add_employee_button.pack(pady=10)

# Button to display employee information
display_employees_button = Button(root, text="Display Employees",
            command=display_employees, bg="#4CAF50", fg="white", font=("Roboto", 12, 'bold'))
display_employees_button.pack(pady=10)

# Run the main event loop
root.mainloop()

# End Marker
```

Chapter 5 Ex 3 - Employee Class

Tkinter GUI app for a Student class. Collects user input (name, marks), calculates average grade, and displays info with a messagebox.

Output





Chapter 5 Ex 4 - Shapes

Area Calculator with tkinter, messagebox, and math. Includes 'Shape' base class and 'Circle', 'Rectangle', 'Triangle' subclasses. Users select a shape, input sides, and get area via messagebox.

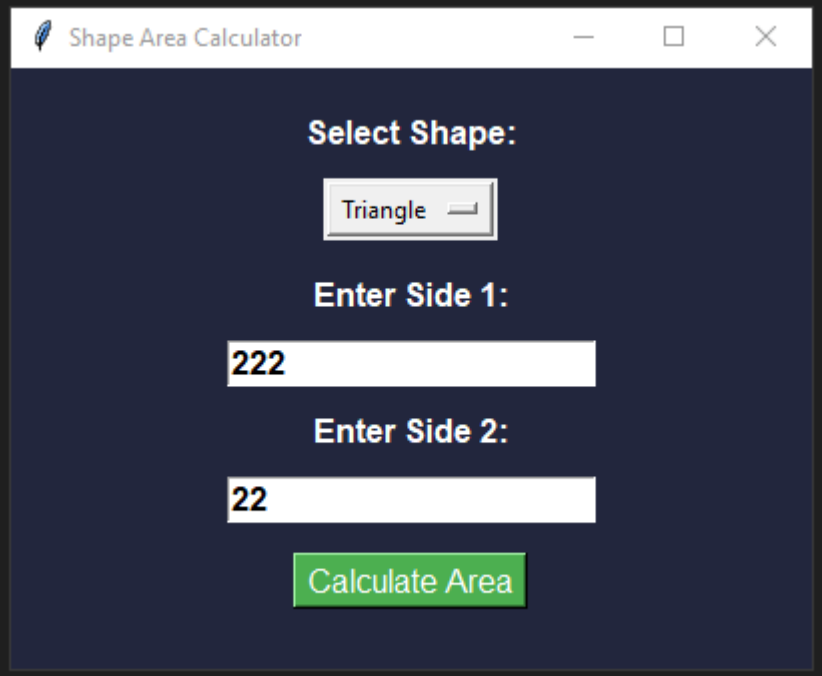
Code

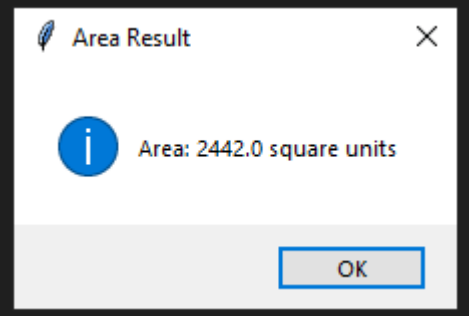
```python
# Chapter 5 Exercise 4 - Shapes

# using the tkinter library,messagebox and math
from tkinter import *
from tkinter import messagebox
import math

# Defining the base class Shape for geometric shapes
class Shape:
    def input_sides():

        #Placeholder method for gathering side inputs.
        # To be overridden by subclasses.
        pass

    def area():

        #Placeholder method for calculating the area.
        # To be overridden by subclasses.
        pass

# Defining the Circle subclass of Shape
class Circle(Shape):
    def input_sides():

        #Get the radius from the user input.
        # Returns float: The radius of the circle.
        return float(side1_entry.get())

    def area():

        #Calculate the area of a circle using the input radius.
        # Returns float: The area of the circle.
        return math.pi * Circle.input_sides() ** 2

# Definung the Rectangle subclass of Shape
class Rectangle(Shape):
    def input_sides():

        #Get the length and width from the user input. Returns tuple:
        # The length and width of the rectangle.
        return float(side1_entry.get()), float(side2_entry.get())

    def area():

        #Calculate the area of a rectangle using the input length
        # and width. Returns float: The area of the rectangle.
        side1, side2 = Rectangle.input_sides()
        return side1 * side2

# Define the Triangle subclass of Shape
class Triangle(Shape):
    def area():
        #Calculate the area of a triangle using the input base and height.
        # Returns  float: The area of the triangle.
        side1, side2 = Triangle.input_sides()
        return 0.5 * side1 * side2

def calculate_area():

    #Calculate and display the area of the selected shape.
    selected_shape = shape_var.get()

    if selected_shape == "Circle":
        area_result = Circle.area()
    elif selected_shape == "Rectangle":
        area_result = Rectangle.area()
    elif selected_shape == "Triangle":
        area_result = Triangle.area()
    else:
        messagebox.showerror("Error", "Please select a shape.")
        return

    messagebox.showinfo("Area Result", f"Area: {area_result:} square units")

# Creating a root window
root = Tk()

# Setting the title for the window
root.title("Shape Area Calculator")

# Setting the dimensions of the window
root.geometry("400x300")

# Creating a Frame for the Shape tab with background color and padding
shape_tab = Frame(root, bg="#E6E6FA", pady=20)

# Label for selecting the shape with specified font and background color
label_shape = Label(shape_tab, text='Select Shape:', font=("Helvetica", 12), bg="#E6E6FA")
label_shape.pack()

# Options for the shapes presented in a drop-down menu
shapes = ["Circle", "Rectangle", "Triangle"]
shape_var = StringVar()
shape_var.set(shapes[0])
shape_option = OptionMenu(shape_tab, shape_var, *shapes)
shape_option.pack(pady=10)

# Labels and Entry widgets for inputting side values
side1_label = Label(shape_tab, text="Enter Side 1:", font=("Helvetica", 12), bg="#E6E6FA")
side1_label.pack(pady=5)
side1_entry = Entry(shape_tab, font=("Helvetica", 12))
side1_entry.pack(pady=5)
side2_label = Label(shape_tab, text="Enter Side 2:", font=("Helvetica", 12), bg="#E6E6FA")
side2_label.pack(pady=5)
side2_entry = Entry(shape_tab, font=("Helvetica", 12))
side2_entry.pack(pady=5)

# Button to trigger the area calculation with specified text, command, font, and color
calculate_button = Button(shape_tab, text='Calculate Area', command=calculate_area,
                          font=("Helvetica", 12), bg="#4CAF50", fg="white")
calculate_button.pack(pady=10)

# Packing the Shape tab
shape_tab.pack(fill=BOTH, expand=YES)

# Run the main event loop
root.mainloop()

# End Marker
```

Chapter 5 Ex 4 - Shapes

Area Calculator with tkinter, messagebox, and math. Includes 'Shape' base class and 'Circle', 'Rectangle', 'Triangle' subclasses. Users select a shape, input sides, and get area via messagebox.

Output





Chapter 5 Ex 5 - Playing around in class

GUI code using tkinter to interact with an Animal class, capturing user input for type, name, colour, age, weight, and noise. Pressing "Create Animal" invokes class functions for greetings, noise, and details.

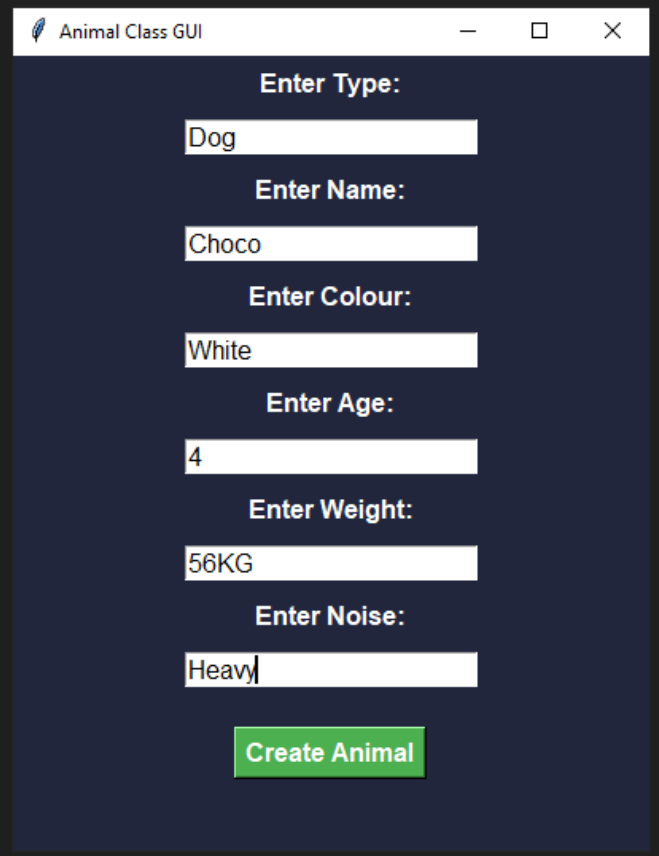
Code

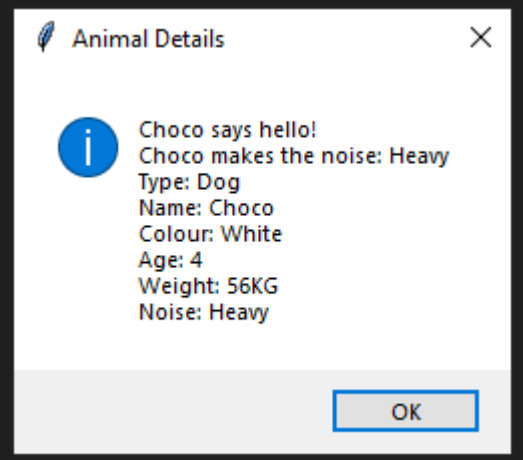
```python
# Chapter 5 Exercise 5 - Playing around in class

# using the tkinter library and messagebox
from tkinter import *
from tkinter import messagebox

# Defiing a class called Animal
class Animal:
    def set_data(animal, animal_type, name, colour, age, weight, noise):
        # Setting values for the members of the Animal class
        animal.type = animal_type
        animal.name = name
        animal.colour = colour
        animal.age = age
        animal.weight = weight
        animal.noise = noise

    def say_hello(animal):
        # to Return a greeting message using the animal's name
        return f"{animal.name} says hello!"

    def make_noise(animal):
        # Return the noise the animal makes along with its name
        return f"{animal.name} makes the noise: {animal.noise}"

    def animal_details(animal):
        # Return all details of the animal
        details = f"Type: {animal.type}\nName: {animal.name}\nColour:
        | {animal.colour}\nAge: {animal.age}\nWeight: {animal.weight}\nNoise: {animal.noise}"
        return details

# making a function to create Animal objects and invoke class functions
def create_animal_objects():
    # Getting values from entry boxes
    animal_type = type_entry.get()
    name = name_entry.get()
    colour = colour_entry.get()
    age = age_entry.get()
    weight = weight_entry.get()
    noise = noise_entry.get()

    # Creating an instance of the Animal class
    animal_object = Animal()

    # Setting data for the Animal instance
    animal_object.set_data(animal_type, name, colour, age, weight, noise)

    # the message to display
    message = f"{animal_object.say_hello()}\n{animal_object.make_noise()}\n
    {animal_object.animal_details()}"

    # the message in a messagebox
    messagebox.showinfo("Animal Details", message)

# Creating the main window
root = Tk()

# Setting the title for the window
root.title("Animal Class GUI")

# Setting the window size
root.geometry("400x500")

# Labels and Entry Boxes for input
type_label = Label(root, text="Enter Type:") # For Type:
type_label.pack(pady=5)
type_entry = Entry(root, font=("Helvetica", 12))
type_entry.pack(pady=5)

name_label = Label(root, text="Enter Name:") # For  Name:
name_label.pack(pady=5)
name_entry = Entry(root, font=("Helvetica", 12))
name_entry.pack(pady=5)

colour_label = Label(root, text="Enter Colour:") # For Colour:
colour_label.pack(pady=5)
colour_entry = Entry(root, font=("Helvetica", 12))
colour_entry.pack(pady=5)

age_label = Label(root, text="Enter Age:") # For Age:
age_label.pack(pady=5)
age_entry = Entry(root, font=("Helvetica", 12))
age_entry.pack(pady=5)

weight_label = Label(root, text="Enter Weight:") # For Weight:
weight_label.pack(pady=5)
weight_entry = Entry(root, font=("Helvetica", 12))
weight_entry.pack(pady=5)
noise_label = Label(root, text="Enter Noise:") # For Noise:
noise_label.pack(pady=5)
noise_entry = Entry(root, font=("Helvetica", 12))
noise_entry.pack(pady=5)

# Button to create Animal objects and call class functions
create_animal_button = Button(root, text="Create Animal",
 command=create_animal_objects, bg="#4CAF50", fg="white", font=("Roboto", 12, 'bold'))
create_animal_button.pack(pady=20)

# Run the main event loop
root.mainloop()

# End Marker
```

Chapter 5 Ex 5 - Playing around in class

GUI code using tkinter to interact with an Animal class, capturing user input for type, name, colour, age, weight, and noise. Pressing "Create Animal" invokes class functions for greetings, noise, and details.

Output





Chapter 5 Ex 6 - Arithmetic Operation

Tkinter-based calculator for arithmetic operations. Allows users to choose an operation, input values, and obtain results with error handling for division by zero or invalid input.

Code

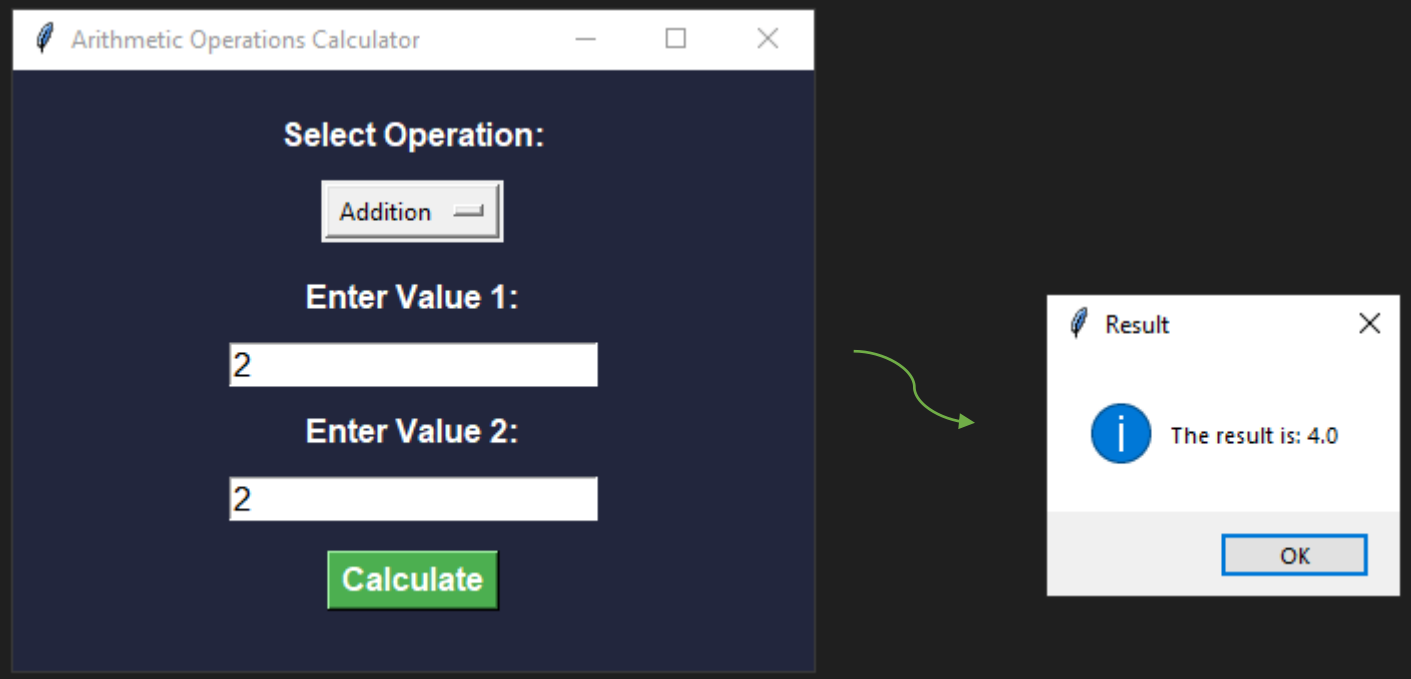
```python
# Chapter 5 Exercise 6 - Arthmetic Operation

# Importing modules from the tkinter library
from tkinter import *
from tkinter import messagebox

# Class for performing arithmetic operations
class ArithmeticOperations:
    def calculate(operation, value1, value2):
        try:
            # To Perform the selected arithmetic operation
            if operation == "Addition":
                return value1 + value2
            elif operation == "Subtraction":
                return value1 - value2
            elif operation == "Multiplication":
                return value1 * value2
            elif operation == "Division":
                # Check for division by zero
                return value1 / value2 if value2 != 0 else messagebox.showerror("Error", "Cannot divide by zero.")
            elif operation == "Exponentiation":
                return value1 ** value2
        except ValueError:
            # Handle invalid input values
            messagebox.showinfo("Error", "Please enter valid numeric values.")
            return

# Function to perform calculation when the "Calculate" button is clicked
def perform_calculation():
    # Getting the selected arithmetic operation
    selected_operation = operation_var.get()

    try:
        # Getting values from entry widgets
        result = ArithmeticOperations.calculate(selected_operation, float(value1_entry.get()),
                                                float(value2_entry.get()))
        if result:
            # Displaying the result in a messagebox pop-up
            messagebox.showinfo("Result", f"The result is: {result}")
    except ValueError:
        # Handle invalid input values
        messagebox.showinfo("Error", "Please enter valid numeric values.")

# Creating the main window
root = Tk()

# Setting the title for the window
root.title("Arithmetic Operations Calculator")

# Setting the window size
root.geometry("400x300")
```

```python
# Widgets for Arithmetic Operations tab
arithmetic_tab = Frame(root, bg="#22263d", pady=20)

# Label for selecting the arithmetic operation
operation_label = Label(arithmetic_tab, text='Select Operation:',
                        font=('Roboto', 12, 'bold'),bg="#22263d",fg='white')
operation_label.pack()

# Options for the arithmetic operations presented in a drop-down menu
operations = ["Addition", "Subtraction", "Multiplication", "Division", "Exponentiation"]
operation_var = StringVar()
operation_var.set(operations[0])
operation_option = OptionMenu(arithmetic_tab, operation_var, *operations)
operation_option.pack(pady=10)

# Labels and Entry widgets for inputting numeric values
value1_label = Label(arithmetic_tab, text="Enter Value 1:",
                     font=('Roboto', 12, 'bold'),bg="#22263d",fg='white')
value1_label.pack(pady=5)
value1_entry = Entry(arithmetic_tab, font=("Helvetica", 12))
value1_entry.pack(pady=5)

value2_label = Label(arithmetic_tab, text="Enter Value 2:",
                     font=('Roboto', 12, 'bold'),bg="#22263d",fg='white')
value2_label.pack(pady=5)
value2_entry = Entry(arithmetic_tab, font=("Helvetica", 12))
value2_entry.pack(pady=5)

# Button to trigger the calculation with specified text, command, font, and color
calculate_button = Button(arithmetic_tab, text='Calculate',
                  command=perform_calculation, font=("Helvetica", 12,'bold'), bg="#4CAF50", fg="white")
calculate_button.pack(pady=10)

# Packing the Arithmetic Operations tab
arithmetic_tab.pack(fill=BOTH, expand=YES)

# Run the main event loop
root.mainloop()

# End Marker
```

Chapter 5 Ex 6 - Arithmetic Operation

Tkinter-based calculator for arithmetic operations. Allows users to choose an operation, input values, and obtain results with error handling for division by zero or invalid input.

Output



Chapter 5 Bonus Ex A - Playing around in class

This Python program defines a 'Animal' class with methods for setting data, saying hello, making noise, and providing details. The 'create_animal_objects' function creates instances of the class using user input and invokes class functions.

Code

```python
# Chapter 5 Bonus Exercise A - Playing around in class

# Defining a class called Animal
class Animal:
    def set_data(animal, animal_type, name, colour, age, weight, noise):
        # Setting values for the members of the Animal class
        animal.type = animal_type
        animal.name = name
        animal.colour = colour
        animal.age = age
        animal.weight = weight
        animal.noise = noise

    def say_hello(animal):
        # to Return a greeting message using the animal's name
        return f"{animal.name} says hello!"

    def make_noise(animal):
        # Return the noise the animal makes along with its name
        return f"{animal.name} makes the noise: {animal.noise}"

    def animal_details(animal):
        # Return all details of the animal
        details = f"Type: {animal.type}\nName: {animal.name}\nColour:
        {animal.colour}\nAge: {animal.age}\nWeight: {animal.weight}\nNoise: {animal.noise}"
        return details
```

```python
# function to create Animal objects and invoke class functions
def create_animal_objects():
    # Getting values from user input in the console
    animal_type = input("Enter Type: ")
    name = input("Enter Name: ")
    colour = input("Enter Colour: ")
    age = input("Enter Age: ")
    weight = input("Enter Weight: ")
    noise = input("Enter Noise: ")

    # Creating an instance of the Animal class
    animal_object = Animal()

    # Setting data for the Animal instance
    animal_object.set_data(animal_type, name, colour, age, weight, noise)

    # Print the results directly to the console
    print(animal_object.say_hello())
    print(animal_object.make_noise())
    print(animal_object.animal_details())

# Calling the function to create Animal objects
create_animal_objects()
```

Output

```
Name: Dog
Colour: white
Age: 2
Weight: 22
Noise: heavy
```

Chapter 5 Bonus Ex A - Playing around in class

Program calculator for advance arithmetic operations. Allows users to choose an operation, input values, and obtain results with error handling for division by zero or invalid input.

Code

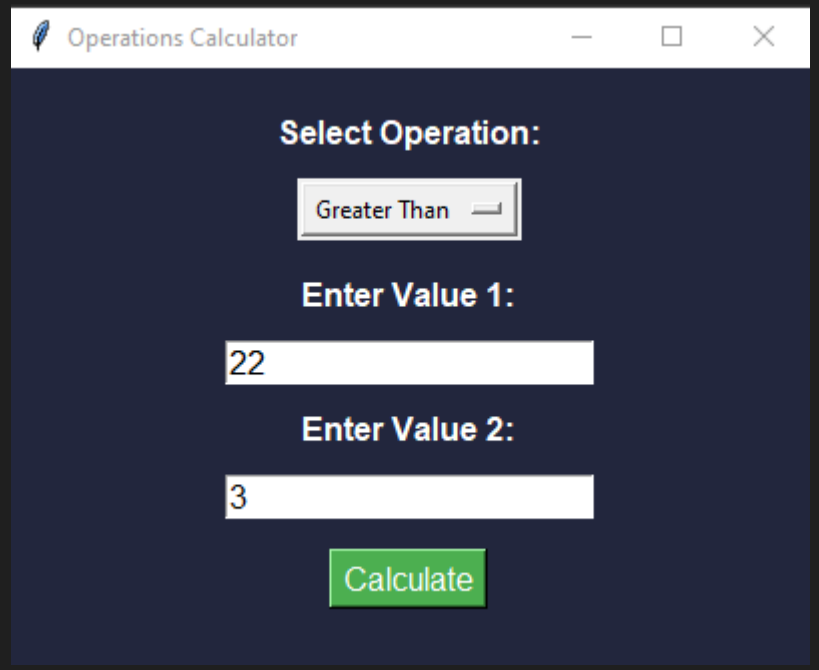
```python
# Chapter 4 Further Exercise i - Arithmetic Operation - Extension

# Importing modules from the tkinter library
from tkinter import *
from tkinter import messagebox

# Class for performing operations
class Operations:
    def calculate_arithmetic(operation, value1, value2):
        try:
            if operation == "Addition":
                return value1 + value2
            elif operation == "Subtraction":
                return value1 - value2
            elif operation == "Multiplication":
                return value1 * value2
            elif operation == "Division":
                return value1 / value2 if value2 != 0 else messagebox.showerror("Error", "Cannot divide by zero.")
            elif operation == "Exponentiation":
                return value1 ** value2
        except ValueError:
            messagebox.showinfo("Error", "Please enter valid numeric values.")
            return None

    def calculate_relational(operation, value1, value2):
        try:
            if operation == "Equal":
                return value1 == value2
            elif operation == "Not Equal":
                return value1 != value2
            elif operation == "Greater Than":
                return value1 > value2
            elif operation == "Less Than":
                return value1 < value2
            elif operation == "Greater Than or Equal To":
                return value1 >= value2
            elif operation == "Less Than or Equal To":
                return value1 <= value2
        except ValueError:
            messagebox.showinfo("Error", "Please enter valid numeric values.")
            return None

# Function to perform calculation when the "Calculate" button is clicked
def perform_calculation():
    selected_operation = operation_var.get()
    value1 = float(value1_entry.get())
    value2 = float(value2_entry.get())

    if selected_operation in arithmetic_operations:
        result = Operations.calculate_arithmetic(selected_operation, value1, value2)
    elif selected_operation in relational_operations:
        result = Operations.calculate_relational(selected_operation, value1, value2)
    else:
        messagebox.showinfo("Error", "Invalid operation selected.")
        return

    if result is not None:
        messagebox.showinfo("Result", f"The result is: {result}")
```

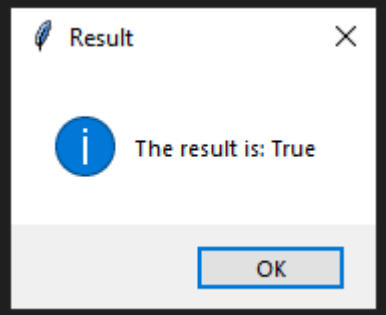
```python
# Creating the main window
root = Tk()

# Setting the title for the window
root.title("Operations Calculator")

# Setting the window size
root.geometry("400x300")

# Widgets for Operations tab# Widgets for Arithmetic Operations tab
operations_tab = Frame(root, bg="#22263d", pady=20)

# Label for selecting the operation
operation_label = Label(operations_tab, text='Select Operation:', font=('Roboto', 12, 'bold'),bg="#22263d",fg='white')
operation_label.pack()

# Options for the operations presented in a drop-down menu
arithmetic_operations = ["Addition", "Subtraction", "Multiplication", "Division", "Exponentiation"]
relational_operations = ["Equal", "Not Equal", "Greater Than", "Less Than", "Greater Than or Equal To",
                         "Less Than or Equal To"]
all_operations = arithmetic_operations + relational_operations

operation_var = StringVar()
operation_var.set(all_operations[0])
operation_option = OptionMenu(operations_tab, operation_var, *all_operations)
operation_option.pack(pady=10)

# Labels and Entry widgets for inputting numeric values
value1_label = Label(operations_tab, text="Enter Value 1:", font=('Roboto', 12, 'bold'),bg="#22263d",fg='white')
value1_label.pack(pady=5)
value1_entry = Entry(operations_tab, font=("Helvetica", 12))
value1_entry.pack(pady=5)

value2_label = Label(operations_tab, text="Enter Value 2:", font=('Roboto', 12, 'bold'),bg="#22263d",fg='white')
value2_label.pack(pady=5)
value2_entry = Entry(operations_tab, font=("Helvetica", 12))
value2_entry.pack(pady=5)

# Button to trigger the calculation with specified text, command, font, and color
calculate_button = Button(operations_tab, text='Calculate', command=perform_calculation,
                          font=("Helvetica", 12), bg="#4CAF50", fg="white")
calculate_button.pack(pady=10)

# Packing the Operations tab
operations_tab.pack(fill=BOTH, expand=YES)

# Run the main event loop
root.mainloop()

# End Marker
```

Chapter 5 Bonus Ex A - Playing around in class

Program calculator for arithmetic operations. Allows users to choose an operation, input values, and obtain results with error handling for division by zero or invalid input.

Output





Chapter 6 - Python Standard Library Modules

Exercises, Bonus And Further

Chapter 6 Ex 1 - Basic Math

Basic Math Operations with the math library. Calculates ceil, floor, factorial, exponentiation, and square root.

Code

```python
# Chapter 5 Exercise 1 - Basic Math


# Using math library to do simple function
import math

# Given values
a1 = 2.3
a2 = 5
a3 = 16

# Calculate ceil and floor
ceil_result = math.ceil(a1)
floor_result = math.floor(a1)

# Calculate factorial
factorial_result = math.factorial(a2)

# Calculate exponentiation
exponentiation_result = 2 ** 3

# Calculate square root
sqrt_result = math.sqrt(a3)

# Print the results
print(f"For a = {a1}, ceil(a) is: {ceil_result}")
print(f"For a = {a1}, floor(a) is: {floor_result}")
print(f"For a = {a2}, the factorial is: {factorial_result}")
print(f"The value of 2^3 is: {exponentiation_result}")
print(f"For a = {a3}, the square root is: {sqrt_result}")

# End marker
```

Output

```
For a = 2.3, ceil(a) is: 3
For a = 2.3, floor(a) is: 2
For a = 5, the factorial is: 120
The value of 2^3 is: 8
For a = 16, the square root is: 4.0
```

Chapter 6 Ex 2 - Numpy Array

This code uses NumPy, a Python numerical computing library, for operations on a list. The program demonstrates array conversion, finding even indices, sorting, and slicing. The comments provide insights into each step.

Code

```python
# Chapter 5 Exercise 2 - NumPy Array

# Import the NumPy library
import numpy as np

# Given list
a = [20, 23, 82, 40, 32, 15, 67, 52]

# Convert the list to a NumPy array
a = np.array(a)

# Find the indices of even numbers
even_indices = np.where(a % 2 == 0)
print("Indices of even numbers:", even_indices[0])

# Sort the list
sorted_list = sorted(a)
print("Sorted list:", sorted_list)

# Slice elements from index 3 to the end of the list, Here a[start:stop], here start = 5, stop = 0
slice1 = a[3:]
print("Slice from index 3 to the end:", slice1)

# Slice elements from index 0 to index 4, Here a[start:stop], here start = 0, stop = 5
slice2 = a[:5]
print("Slice from index 0 to index 4:", slice2)

# Print [32 15 67] using negative slicing, Here a[start:stop], here start = -5, stop = -2
negative_slice = a[-5:-2]
print("Negative slice:", negative_slice)

# End marker
```

Output

```
Indices of even numbers: [0 2 3 4 7]
Sorted list: [15, 20, 23, 32, 40, 52, 67, 82]
Slice from index 3 to the end: [40 32 15 67 52]
Slice from index 0 to index 4: [20 23 82 40 32]
Negative slice: [40 32 15]
```

Chapter 6 Ex 3 - Calculator

Simple calculator program allowing users to perform basic arithmetic operations with a menu, user interaction loop, and error handling for invalid input.

Code

```python
# Chapter 5 Exercise 3 - Calculator

# Importing the math module to use arithmetic operations
import math

# Function to create a simple calculator
def calculator():
    # Displaying the calculator menu
    print("Calculator Menu")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")
    print("5. Modulus")
    print("6. Check greater number")
    print("Enter Q to quit")

    # Main loop for the calculator
    while True:
        try:
            # Getting user input for the operation choice
            choice = input("Enter your choice (1-6): ")

            # Checking if the user wants to quit
            if choice == "Q" or choice == "q":
                print("Closing program")
                break

            # Converting the choice to an integer
            choice = int(choice)

            # Validating the user's choice
            if 1 <= choice <= 6:
                # Getting user input for the numbers
                num1 = float(input("Enter first number: "))
                num2 = float(input("Enter second number: "))

                # Performing the selected operation based on the user's choice
                if choice == 1:
                    result = num1 + num2
                elif choice == 2:
                    result = num1 - num2
                elif choice == 3:
                    result = num1 * num2
                elif choice == 4:
                    result = num1 / num2
                elif choice == 5:
                    result = num1 % num2
                elif choice == 6:
                    result = max(num1, num2)

                # Displaying the result of the calculation with quit option
                print("Result:", result)
                print("Enter Q to quit")
            else:
                print("Invalid choice. Please enter a number between 1 and 6.")
        except ValueError:
            # Handling invalid input errors
            print("Invalid input. Please enter a valid number.")

# Calling the calculator function to run the program
calculator()

# End marker
```

Output

```
Calculator Menu
1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulus
6. Check greater number
Enter Q to quit
Enter your choice (1-6): 2
Enter first number: 33
Enter second number: 3
Result: 30.0
Enter Q to quit
Enter your choice (1-6): q
Closing program
```

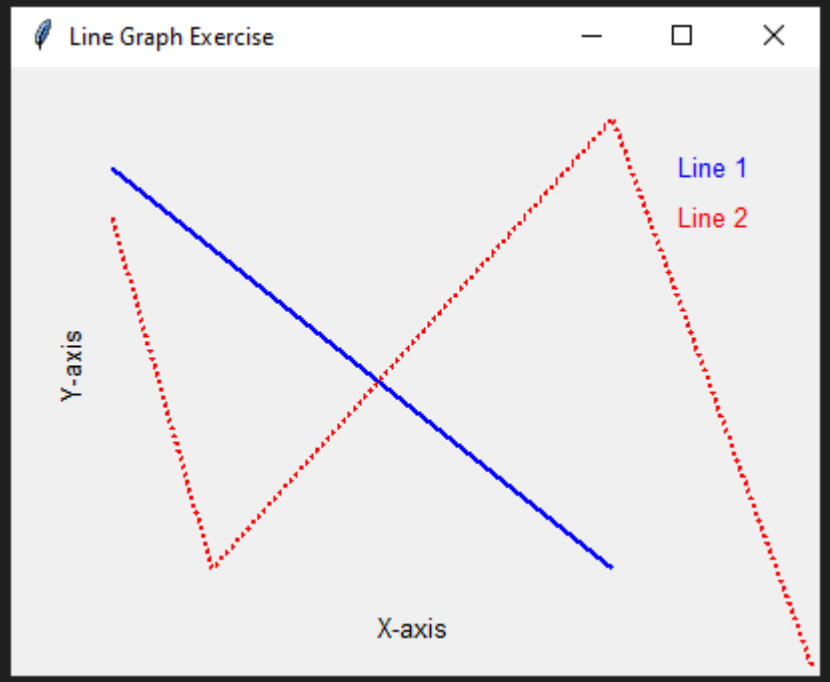
Chapter 6 Ex 4 - Line graph

This Python code uses the tkinter library to create a simple line graph. It generates a Tkinter window with a canvas widget, drawing two lines, adding axis labels, and including a legend. The code, its output, and explanatory comments are provided below.

Code

```python
# Chapter 5 Exercise 4 - Line Graph

# using the tkinter library
from tkinter import *

# Creating a Tkinter window
root = Tk()
root.title('Line Graph Exercise')

# Creating a Canvas widget
canvas = Canvas(root, width=400, height=300)
canvas.pack()

# Drawing Line 1: From (50, 50) to (300, 250)
canvas.create_line(50, 50, 300, 250, fill='blue', width=2, tags='line1')

# Drawing Line 2: Dotted line from (50, 75) to (100, 250) to (300, 25) to (400, 300)
canvas.create_line(50, 75, 100, 250, 300, 25, 400, 300, fill='red', width=2, dash=(4, 4), tags='line2')

# Adding labels
canvas.create_text(200, 280, text='X-axis', font=('Helvetica', 10))
canvas.create_text(30, 150, text='Y-axis', font=('Helvetica', 10), angle=90)

# Adding legend
canvas.create_text(350, 50, text='Line 1', font=('Helvetica', 10), fill='blue')
canvas.create_text(350, 75, text='Line 2', font=('Helvetica', 10), fill='red')

# Run the main event loop
root.mainloop()

# End Marker
```

Output



Chapter 6 Ex 5 - Working with JSON File

This code captures user input for student details, writes it to a JSON file, reads the file, and displays individual values. Comments explain each step.

Code

```python
# Chapter 5 Exercise 5 - Working with JSON File


# Import the json module to work with JSON data
import json

# Specify the file path
file_path = 'StudentJson.json'

# Step 1: Ask the user to enter student details and write to JSON file

# Create a dictionary to store the user's input for student details
student_details = {
    "Name": input("Enter student name: "),
    "ID": int(input("Enter student ID: ")),
    "course": input("Enter student course: ")
}

try:
    # Opening the JSON file in read and append mode
    with open(file_path, 'w+') as json_file:
        # Move the file pointer to the beginning before reading
        json_file.seek(0)

        # Load the existing JSON data from the file into a Python dictionary
        try:
            existing_data = json.load(json_file)
        except json.JSONDecodeError:
            existing_data = {}

        # Update existing_data with new student details
        existing_data["StudentDetails"] = student_details

        # Move the file pointer to the beginning before writing
        json_file.seek(0)

        # Write the updated data to the file
        json.dump(existing_data, json_file, indent=4)
except FileNotFoundError:
    # Handle the case when the file doesn't exist
    with open(file_path, 'w') as json_file:
        json.dump({"StudentDetails": student_details}, json_file, indent=4)

# Step 2: Read and display the individual values

# Opening the JSON file in read mode
with open(file_path, 'r') as json_file:
    # Load the JSON data from the file into a Python dictionary
    data = json.load(json_file)

    # Print the details of the student
    print("\nDetails of the Student are")
    for key, value in data["StudentDetails"].items():
        print(f"\t{key}: {value}")

#End marker
```

Output

```json
{
    "StudentDetails": {
        "Name": "ASIF",
        "ID": 2022469,
        "course": "CC L5"
    }
}
```

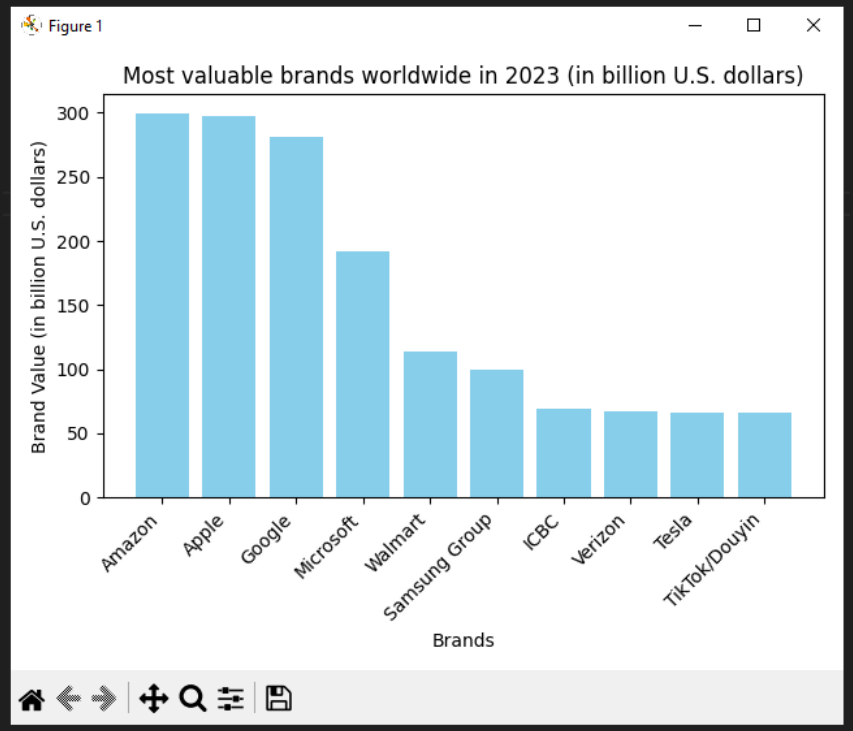
Chapter 6 Bonus Ex A – Bar graph

This code uses matplotlib to create a bar graph showing the 2023 brand values. It illustrates the most valuable brands worldwide in billion U.S. dollars. Python code, the resulting graph, and explanatory comments are provided below.

Code

```python
# Chapter 6 Bonus Exercise A - Bar Grapgh

# Import the pyplot module from the matplotlib library for graph
import matplotlib.pyplot as plt

# Information for the bar graph
title = "Most valuable brands worldwide in 2023 (in billion U.S. dollars)"
brands = ["Amazon", "Apple", "Google", "Microsoft", "Walmart", "Samsung Group", "ICBC", "Verizon", "Tesla", "TikTok/Douyin"]
values = [299.28, 297.51, 281.38, 191.57, 113.78, 99.66, 69.55, 67.44, 66.21, 65.67] # Given Data

# Plotting the bar graph
plt.bar(brands, values, color='skyblue')  # Create a bar graph with brands on the x-axis and values on the y-axis
plt.xlabel('Brands')  # Label for the x-axis
plt.ylabel('Brand Value (in billion U.S. dollars)')  # Label for the y-axis
plt.title(title)  # Title of the plot
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent clipping of labels

# Display the bar graph
plt.show()

# End marker
```

Output



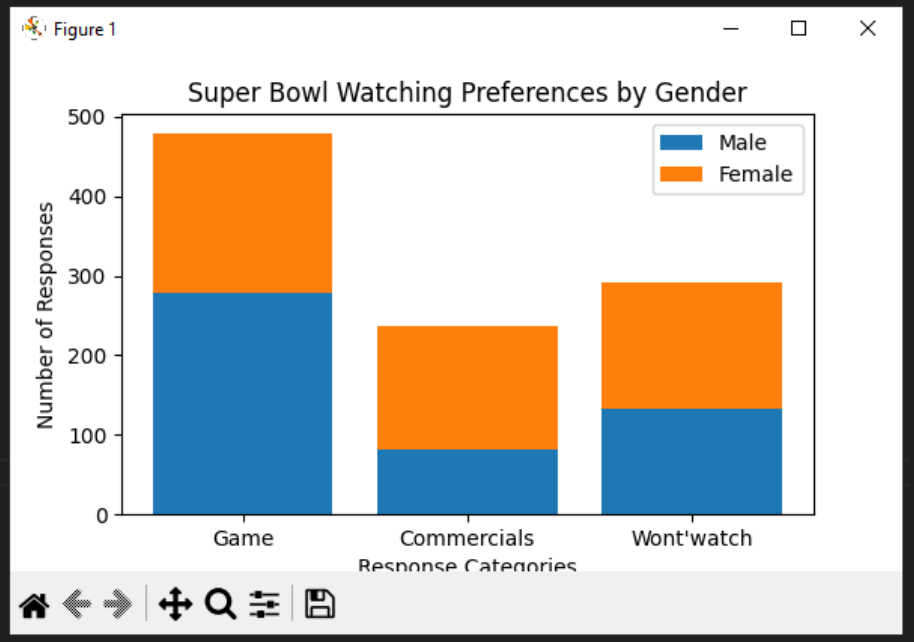
Chapter 6 Further Ex I - Stack bar graph

This Python code uses matplotlib to create a stacked bar graph showing Super Bowl watching preferences by gender. It captures responses in categories like 'Game,' 'Commercials,' and 'Won't watch.' The code, output (a stacked bar graph), and comments are provided below for clarity.

Code

```python
# Chapter 5 Further Exercise I - Stack Bar Graph

# Import the pyplot module from the matplotlib library for graph
import matplotlib.pyplot as plt

# Define data from the table
categories = ['Game', 'Commercials', "Wont'watch"]
male_responses = [279, 81, 132]
female_responses = [200, 156, 160]

# Plotting the bar chart
plt.bar(categories, male_responses, label='Male')  # Plotting bars for male responses

# Plotting bars for female responses on top of male responses
plt.bar(categories, female_responses, bottom=male_responses, label='Female')

# Adding labels, title, and legend
plt.xlabel('Response Categories')  # Label for the x-axis
plt.ylabel('Number of Responses')  # Label for the y-axis
plt.title('Super Bowl Watching Preferences by Gender')  # Title of the plot
plt.legend()  # Displaying legend to differentiate between male and female responses

# Display the bar chart
plt.show()

# End marker
```

Output



Chapter 6 Further Ex II - Stack bar graph

This code captures user input for student details, writes it to a JSON file, reads the file, and displays individual values. Comments explain each step.

Code

```python
# Chapter 5 Further Exercise II - Working with JSON file (Extension)

# Import the json module to work with JSON data
import json

# the file path
file_path = 'StudentJson.json'

# Function to get student details
def get_student_details():
    return {
        "Name": input("Enter student name: "),
        "ID": int(input("Enter student ID: ")),
        "course": input("Enter student course: ")
    }

# Step 1: Asking the user to enter details for the first student and write to JSON file
first_student_details = get_student_details()

# Adding CourseDetails for the first student
first_student_details["CourseDetails"] = {
    "Group": "A",
    "Year": 2
}

# Opening the JSON file in write mode and save the details of the first student
with open(file_path, 'w') as json_file:
    json.dump({"StudentDetails": first_student_details}, json_file, indent=4)

# Step 2: Reading and display the details of the first student
with open(file_path, 'r') as json_file:
    data = json.load(json_file)

    # Printing the details of the first student
    print("\nDetails of the First Student are")
    for key, value in data["StudentDetails"].items():
        print(f"\t{key}: {value}")

# Step 3: Appending and update the JSON file with details for a second student
second_student_details = get_student_details()

# Adding CourseDetails for the second student
second_student_details["CourseDetails"] = {
    "Group": "A",
    "Year": 2
}

# Opening the JSON file in read mode and load existing data
with open(file_path, 'r') as json_file:
    existing_data = json.load(json_file)

# Updating the existing data with details for the second student
existing_data["SecondStudentDetails"] = second_student_details


# Reopen the file in write mode to save the updated data
with open(file_path, 'w') as json_file:
    json.dump(existing_data, json_file, indent=4)

# Step 4: Reading and print the updated values for all students
with open(file_path, 'r') as json_file:
    updated_data = json.load(json_file)
    print("\nDetails of the Students are")

    # Printing the details of all students, including the updated course details for both students
    for student, details in updated_data.items():
        print(f"\tStudent: {student}")
        for key, value in details.items():
            print(f"\t\t{key}: {value}")

# End marker
```

Output

```jsonc
{
    "StudentDetails": {
        "Name": "Asif]",
        "ID": 2022469,
        "course": "CC L5",
        "CourseDetails": {
            "Group": "A",
            "Year": 2
        }
    },
    "SecondStudentDetails":
        "Name": "Ahmed",
        "ID": 22323,
        "course": "cc l5",
        "CourseDetails": {
            "Group": "A",
            "Year": 2
        }
    }
}
```