Software Development Life Cycle

ISE 1000 project

Lachlan Dauth | 20937625 | Tuesday 4:00 pm Practical

Introduction

This report documents the implementation and evaluation of two scenarios developed as part of the ISE 1000 project. The project aims to showcase the application of programming concepts and principles in solving real-world problems.

The two scenarios addressed in this report are Scenario A and Scenario B. Scenario A focuses on determining the season of the year based on the given month and country. Scenario B involves calculating and displaying the temperature difference between a given temperature and the mean temperature of a specific city and time period.

The report provides an overview of each scenario, describing the functionality of the implemented modules and their relationships. It discusses the modularity of the code, highlighting the individual functions and their purposes. Furthermore, the report presents the results of black box testing conducted on both scenarios, examining the test cases and identifying any issues or improvements.

Lastly, the report explores the ethical considerations and professionalism demonstrated in the development of the scenarios. It emphasizes the adherence to ethical principles, such as accuracy, transparency, user privacy, and cultural sensitivity. The professionalism of the code is evaluated based on factors like user experience, error handling, and aesthetic presentation.

Module Descriptions

Scenario A

Task:

Finding the season of the year when a country name and the month of the year is given. Different countries around the world have different seasons and there can be variations within a given country also. For simplicity, season variations applicable to few countries shown in Figure 1 (available in the Assignment page of Blackboard) are selected for the sample system.

For this task I assumed that the input will be given from the console and can be given in either the format "January" or "jan" without being case sensitive and the country will be given with correct spelling. I also have used matplotlib to display the image and the console to display the output I have chosen using the console to display the output as the formatted string is more useful to the user than any program that receives it.

month_to_code(month_str)

This function converts a month string to its corresponding numerical code. It can be easily improved to include many more moth formats as it utilises a dictionary.

Parameters:

- `month_str` (str): A string representing a month, e.g., "January", "feb".

Returns:

- `int`: The numerical code for the month, e.g., 1 for "January", 2 for "February", etc.

Raises:

- `ValueError`: If `month_str` is not a valid month format.

display_img(image_path)

This function displays an image using Matplotlib. I have separated this function out as has a single well defined use and could be used again if the module was expanded.

Parameters:

- `image_path` (str): Path to the image file.

Returns:

- `None`

month_to_season(month_str, country_str)

This function determines the season for a given month in a specific country and displays an image representing that season. It is easy to implement more countries as the countries are stored in a dictionary. I also separated out the image path file for reducing repetitive code and making it easier if the image files were to move.

Args:

- `month_str` (str): A string representing a month.
- `country_str` (str): A string representing a country.

Returns:

- `None`

Raises:

- `ValueError`: If the provided country is not found.

input_month_to_season()

This function gets user input for the month and country and calls the `month_to_season` function with the provided values. This function was implemented as it makes it easier to use the function above in only one line

main()

The main function that calls the `input_month_to_season` function.

Scenario B

Task:

Finding whether a given temperate reading is above or below the average temperature of a city (Morning or Evening) will be provided by the system. If the difference is more than 5ºC, additional message also will be provided.

For this task I assumed that the input will be given from the console and can be given in either the format "10" or "10.1" and the city will be given with correct spelling. I have chosen using the console to display the output as the formatted string is more useful to the user than any program that receives it.

temperature_difference(temp, mean_temp)

This function calculates the temperature difference between a given temperature and the mean temperature of a specific city and time period and display relevant information. I separated this function out from the temperature_finder function as it has a single defined use

Args:

- `temp` (float): The given temperature.
- `mean_temp` (float): The mean temperature.

Returns:

- `str`: A formatted string describing the temperature difference.

temperature_finder(temperature_str, city_str, time_str)

This function finds the temperature difference between a given temperature and the mean temperature of a specific city and time period. It is easy to add more cities as I use a dictionary to store the data.

Args:

- `temperature_str` (str): The given temperature.
- `city_str` (str): The name of the city.
- `time_str` (str): The time period (e.g., "morning" or "afternoon").

input_temperature()

This function gets user input for the temperature, city, and time and calls the `temperature_finder` function with the provided values.

main()

The main function that calls the `input_temperature` function.

Modularity

How to run the programs

1. `month_to_season(month_str, country_str)`:
    - This function determines the season for a given month in a specific country and displays an image representing that season.
    - Pass the month and country as arguments to the function, e.g., `month_to_season("January", "Australia")`.
    - The function will determine the corresponding season for the provided month and country.
    - It will print the month, country, and the corresponding season.
    - Additionally, it will display an image representing the season using the `display_img` function.
    - If the provided country is not found in the database, a `ValueError` will be raised.
2. `input_month_to_season()`:
    - This function prompts the user to enter a month and a country.
    - To use this function, simply call it without any arguments, e.g., `input_month_to_season()`.
    - The function will ask the user to enter a month and a country.
    - Once the user provides the input, it will call the `month_to_season` function with the user-provided values.
3. `temperature_finder(temperature_str, city_str, time_str)`:
    - This function finds the temperature difference between a given temperature and the mean temperature of a specific city and time period.
    - Pass three arguments to the function: `temperature_str` (a string representing the given temperature), `city_str` (a string representing the city), and `time_str` (a string representing the time period, e.g., "morning" or "afternoon").
    - The function uses a dictionary `temperature_dict` to store mean temperatures for different cities and time periods.
    - It retrieves the mean temperature based on the provided city and time period, and then calls the `temperature_difference` function to calculate and print the temperature difference.
4. `input_temperature()`:

    - This function prompts the user to enter the temperature, city, and time period.
    - To use this function, simply call it without any arguments, e.g., `input_temperature()`.
    - The function will ask the user to enter the temperature, city, and time.
    - Once the user provides the input, it will call the `temperature_finder` function with the user-provided values.

Modularity Checklist

Certainly! Here's a report on the cohesion, coupling, and code redundancy in both scenarios:

Scenario A:

Cohesion:

- The `temperature_difference(temp, mean_temp)` function has high cohesion as it performs a single task of calculating the temperature difference and returning a formatted string.
- The `temperature_finder(temperature_str, city_str, time_str)` function also exhibits high cohesion as it is responsible for finding the temperature difference based on the provided inputs and printing the result.
- The `input_temperature()` function is cohesive as it handles user input and calls the necessary functions to find and display the temperature difference.

Coupling:

- There is low coupling between the functions in Scenario A. Each function operates independently and takes explicit input parameters, which reduces the coupling between them. They communicate through the parameters and return values.

Code Redundancy:

- Scenario A does not exhibit code redundancy as each function serves a specific purpose and there are no instances of duplicate code. The functions are well-encapsulated and avoid repetition.

Scenario B:

Cohesion:

- The `month_to_code(month_str)` function has high cohesion as it focuses on a single task of converting a month string to its corresponding numerical code.
- The `display_img(image_path)` function also exhibits high cohesion as it is responsible for displaying an image using Matplotlib.
- The `month_to_season(month_str, country_str)` function has high cohesion as it determines the season for a given month in a specific country and displays an image representing that season.

Coupling:

- There is low coupling between the functions in Scenario B. Each function operates independently and takes explicit input parameters, reducing the coupling between them. They communicate through the parameters and return values.

Code Redundancy:

- Scenario B does not show significant code redundancy. However, there is some redundancy in the dictionary `country_season`, which includes the same month-to-season mappings for different countries. This redundancy could be reduced by having a separate dictionary for the common month-to-season mappings and referencing it in the country-specific dictionaries.

Overall, both scenarios demonstrate good cohesion, low coupling, and minimal code redundancy. The functions are focused on their respective tasks and can be used independently without excessive interdependence or duplicated code.

Black Box Testing

For black box testing I will test the temperature_finder and month_to_season functions. For scenario B I also did boundary analysis on the values around a difference of 5.

Scenario A

Test Case	Month	Country	Expected Output
Case 1	January	Australia	When it is January in Australia, it is the Summer season.
Case 2	March	Noongar	When it is March in Noongar, it is the Bunuru season.
Case 3	July	Spain	When it is July in Spain, it is the Summer season.
Case 4	December	Japan	When it is December in Japan, it is the Winter season.
Case 5	May	Mauritius	When it is May in Mauritius, it is the Autumn season.
Case 6	September	Sri Lanka	When it is September in Sri Lanka, it is the Southeast Monsoon season.
Case 7	October	Malaysia	When it is October in Malaysia, it is the Inter-Monsoon season.

Scenario B

Temperature	City	Time	Expected Output
18.0	perth	morning	"It is colder than the mean temperature."
24.0	perth	afternoon	"It is hotter than the mean temperature."
16	adelaide	morning	"It is colder than the mean temperature."
23	adelaide	afternoon	"It is hotter than the mean temperature."
18.2	perth	morning	"It is the same as the mean temperature."
18	perth	afternoon	"It is colder than the mean temperature."

Temperature	City	Time	Expected Output
28	perth	afternoon	"It is hotter than the mean temperature."
17.9	perth	afternoon	"It is 17.9 degrees outside, 5.1 degrees colder than the mean temperature, 23."
28.1	perth	afternoon	"It is 28.1 degrees outside, 5.1 degrees hotter than the mean temperature, 23."

Result of black box testing

As a result of black box testing I found out the scenario B code had a problem when the temperature was the same as the mean temperature it would report as colder and due to a bug in python subtraction I was getting 5.1000001 instead of 5.1 which needed the function round() to fix. And because of the success of black box testing and the limited number of if statements or loops I choose to skip white box testing.

Ethics and professionalism

Scenario A: Scenario A revolves around temperature calculations and finding temperature differences between a given temperature and the mean temperature. It also involves user input for temperature, city, and time.

1. Ethical Considerations:

   - Accuracy and Transparency: The scenario promotes ethical values by striving for accuracy and transparency in temperature calculations. Users receive reliable information about temperature differences, enabling them to make informed decisions.
   - User Privacy: The scenario respects user privacy by collecting only necessary information (temperature, city, time) without storing or transmitting any personal data. It adheres to ethical principles of data minimization and confidentiality.

2. Professionalism:

   - User Experience: The scenario emphasizes professionalism by providing clear prompts for user input and gracefully handling potential errors or invalid inputs. It ensures a smooth user experience and maintains the integrity of the program.

Scenario B: Scenario B involves converting month strings to numerical codes, determining seasons for specific months and countries, and displaying corresponding images.

1. Ethical Considerations:

   - Cultural Sensitivity: The scenario exhibits ethical awareness by associating seasons with specific countries, respecting diverse cultural and climatic patterns worldwide. It avoids making assumptions based solely on a single region's climate, fostering inclusivity and cultural understanding.
   - User Experience and Accessibility: The scenario promotes ethical values by considering user experience and accessibility. The display of images using Matplotlib ensures that

visually impaired users are not excluded, providing alternative ways to consume information.

2. Professionalism:

    - Robustness and Error Handling: The functions in Scenario B showcase professionalism by handling potential errors or invalid inputs. Proper error handling and exception raising enhance code robustness and stability.
    - Aesthetic Presentation: The scenario demonstrates professionalism by presenting images in a clean and aesthetically pleasing manner. Hiding unnecessary elements and following standard practices for image display enhance the overall user experience.