**System Requirements Specification Index**

**For**

**Plant Care Advisory System Console Application**

**Version 1.0**

**IIHT Pvt. Ltd.**

**fullstack@iiht.com**

**Table of Contents**

1 Project Abstract

2 Business Requirements

3 Constraints

4 Template Code Structure

5 Execution Steps to Follow

**Pizza Shop Calculator Console**

**System Requirements Specification**

**1** **Project Abstract**

BloomWise Solutions, a growing agritech startup based in Pune, requires a smart gardening system with a plant care advisory program. This Python console application helps users determine watering schedules, sunlight requirements, and care instructions based on plant type, environmental conditions, and season. The system uses conditional statements (if, if-else, if-elif-else) to provide specific care recommendations. As urban gardening gains popularity across Indian metropolitan areas, BloomWise Solutions recognized the need for accessible plant care guidance for novice gardeners living in apartments with limited space and varying light conditions. Their customer research revealed that many first-time plant owners struggle with establishing proper care routines, leading to plant health issues and customer frustration. This application aims to bridge the knowledge gap by offering customized care recommendations that adapt to seasonal changes and specific plant varieties common in Indian homes.

**2** **Business Requirements:**

| Screen Name | Console input screen |
| --- | --- |
| Problem Statement | 1. Application must determine watering frequency based on plant type using if-elif-else  2. System should adjust care based on season using if-else  3. Program should calculate sunlight requirements using if statements  4. Console should recommend plant care instructions based on conditions  5. Program should provide care warnings based on temperature and humidity |

**3** **Constraints**

**3.1** **Input Requirements**

1. Plant Type:

o Must be stored as integer in variable **plant\_type**

o 1: Succulent

o 2: Tropical

o 3: Flowering

o 4: Herb

o Example: 2

2. Current Season:

o Must be stored as integer in variable season

o 1: Spring

o 2: Summer

o 3: Fall

o 4: Winter

o Example: 1

3. Temperature:

* + Must be stored as float in variable temperature
  + Must be between -10.0 and 50.0 Celsius
  + Example: 25.5

4. Humidity:

* + Must be stored as integer in variable humidity
  + Must be between 0 and 100 percent
  + Example: 60

**3.2** **Calculation Constraints**

**1.** Watering Schedule (if-elif-else):

○ Succulent (plant\_type 1): Return exactly 14 days

○ Tropical (plant\_type 2): Return exactly 3 days

○ Flowering (plant\_type 3): Return exactly 2 days

○ Herb (plant\_type 4): Return exactly 1 day

**2.** Seasonal Adjustment (if-else):

○ Summer (season 2): Decrease watering days by 1 (minimum 1 day)

○ Winter (season 4): Increase watering days by 1

○ Spring/Fall (season 1/3): No change to watering days

**3.** Temperature Warning (if-elif-else):

○ Too Hot: When temperature > 30.0, return exactly: "Temperature too high Risk of heat stress"

○ Too Cold: When temperature < 10.0, return exactly: "Temperature too low Risk of cold damage"

○ Optimal: Otherwise return exactly: "Temperature optimal for plant growth"

**4.** Humidity Requirements (if-elif-else):

○ Low: When humidity < 30, return tuple: ("Low", "Increase humidity with misting")

○ Medium: When 30 <= humidity <= 60, return tuple: ("Medium", "Humidity is optimal")

○ High: When humidity > 60, return tuple: ("High", "Monitor for fungal growth")

**5.** Sunlight Requirements (if-elif-else):

○ Succulent (plant\_type 1): Return exactly "Full sun to partial shade"

○ Tropical (plant\_type 2): Return exactly "Bright indirect light"

○ Flowering (plant\_type 3): Return exactly "Full sun"

○ Herb (plant\_type 4): Return exactly "At least 6 hours of direct sunlight"

**3.3** **Output Constraints**

1. Display Format:

o Show "Watering Schedule: Every {X} days"

o Show "Sunlight Requirement: {requirement}"

o Show "Temperature Status: {status}"

o Show "Humidity Level: {level}"

o Show "Special Care Instructions:" followed by plant-specific instructions

**2.** Plant-Specific Care Instructions:

o Succulent (plant\_type 1): Include "Avoid overwatering"

o Tropical (plant\_type 2): Include "Maintain high humidity"

o Flowering (plant\_type 3): Include "Remove dead flowers regularly"

o Herb (plant\_type 4): Include "Harvest regularly to promote growth"

**3.** Seasonal Care Tips:

o Summer (season 2): Include "Increase watering frequency"

o Winter (season 4): Include "Reduce watering frequency"

**4.** Environmental Advice:

o High Temperature (> 30.0): Include "Provide shade and increase watering"

o Low Temperature (< 10.0): Include "Protect from cold and reduce watering"

o Include the humidity advice from determine\_humidity\_needs()

**4. Template code Structure:**

**1.** Conditional Functions:

o calculate\_watering\_schedule() [if-elif-else]

o adjust\_for\_season() [if-else]

o check\_temperature() [if-elif-else]

o determine\_humidity\_needs() [if-elif-else]

o generate\_care\_instructions() [nested if]

o get\_sunlight\_requirement() [if-elif-else]

**2.** Input Section:

o Get plant type (int)

o Get current season (int)

o Get temperature (float)

o Get humidity (int)

**3.** Processing Section:

o Calculate base watering schedule

o Apply seasonal adjustments

o Check environmental conditions

o Generate care recommendations

**4.** Output Section:

o Display watering schedule

o Show environmental status

o List care instructions

o Display warnings

# **5. DETAILED FUNCTION IMPLEMENTATION GUIDE**

## **5.1 Plant Type Classification Functions**

**1. Write a Python function to calculate watering frequency based on plant type.** Define: calculate\_watering\_schedule(plant\_type)

The function should:

* Accept one integer parameter representing the plant type (1-4)
* Validate that input is an integer using isinstance(plant\_type, int)
* Check for boolean input first since bool is a subclass of int: isinstance(plant\_type, bool)
* Raise ValueError with message "Plant type must be an integer." for non-integer inputs
* Validate that plant\_type is between 1 and 4 (inclusive)
* Raise ValueError with message "Invalid plant type. Must be between 1 and 4." for out-of-range values
* Use if-elif-else structure to determine watering frequency:
  + if plant\_type == 1: return 14 (Succulent - water every 14 days)
  + elif plant\_type == 2: return 3 (Tropical - water every 3 days)
  + elif plant\_type == 3: return 2 (Flowering - water every 2 days)
  + elif plant\_type == 4: return 1 (Herb - water every 1 day)
* Return the watering frequency as an integer
* Example: calculate\_watering\_schedule(1) should return 14

**2. Write a Python function to determine sunlight requirements for each plant type.** Define: get\_sunlight\_requirement(plant\_type)

The function should:

* Accept one integer parameter representing the plant type (1-4)
* Validate input using same validation as calculate\_watering\_schedule
* Check for boolean input first: isinstance(plant\_type, bool)
* Raise appropriate ValueError messages for invalid inputs
* Use if-elif-else structure to determine sunlight needs:
  + if plant\_type == 1: return "Full sun to partial shade" (Succulent)
  + elif plant\_type == 2: return "Bright indirect light" (Tropical)
  + elif plant\_type == 3: return "Full sun" (Flowering)
  + elif plant\_type == 4: return "At least 6 hours of direct sunlight" (Herb)
* Return the exact sunlight requirement string as specified
* Example: get\_sunlight\_requirement(2) should return "Bright indirect light"

## **5.2 Seasonal Adjustment Functions**

**3. Write a Python function to adjust watering schedule based on current season.** Define: adjust\_for\_season(days, season)

The function should:

* Accept two integer parameters: base watering days and season (1-4)
* Validate that both inputs are integers, checking for booleans first
* Raise ValueError with message "Days must be an integer." for invalid days input
* Raise ValueError with message "Season must be an integer." for invalid season input
* Validate that days is non-negative: if days < 0:
* Raise ValueError with message "Base schedule cannot be negative." for negative days
* Validate that season is between 1 and 4 (inclusive)
* Raise ValueError with message "Invalid season. Must be between 1 and 4." for invalid season
* Use if-elif-else structure to adjust watering frequency:
  + if season == 2: return max(1, days - 1) (Summer - decrease by 1, minimum 1 day)
  + elif season == 4: return days + 1 (Winter - increase by 1 day)
  + else: return days (Spring/Fall - no change)
* Return the adjusted watering days as an integer
* Example: adjust\_for\_season(3, 2) should return 2 (summer adjustment)

## **5.3 Environmental Assessment Functions**

**4. Write a Python function to check if temperature is within optimal range.** Define: check\_temperature(temperature)

The function should:

* Accept one float parameter representing temperature in Celsius
* Validate that input is a number using isinstance(temperature, (int, float))
* Exclude boolean inputs: or isinstance(temperature, bool)
* Raise ValueError with message "Temperature must be a number." for invalid inputs
* Validate temperature range between -10.0 and 50.0 Celsius
* Raise ValueError with message "Temperature must be between -10.0 and 50.0 Celsius." for out-of-range values
* Use if-elif-else structure to categorize temperature:
  + if temperature > 30.0: return "Temperature too high - Risk of heat stress"
  + elif temperature < 10.0: return "Temperature too low - Risk of cold damage"
  + else: return "Temperature optimal for plant growth"
* Return the exact temperature status string as specified
* Example: check\_temperature(35.0) should return "Temperature too high - Risk of heat stress"

**5. Write a Python function to determine humidity level and provide care advice.** Define: determine\_humidity\_needs(humidity)

The function should:

* Accept one integer parameter representing humidity percentage (0-100)
* Validate that input is an integer, checking for booleans first
* Raise ValueError with message "Humidity must be an integer." for invalid inputs
* Validate humidity range between 0 and 100 (inclusive)
* Raise ValueError with message "Humidity must be between 0 and 100 percent." for out-of-range values
* Use if-elif-else structure to categorize humidity and provide advice:
  + if humidity < 30: return ("Low", "Increase humidity with misting")
  + elif humidity > 60: return ("High", "Monitor for fungal growth")
  + else: return ("Medium", "Humidity is optimal")
* Return a tuple containing exactly 2 string elements: (level, advice)
* Example: determine\_humidity\_needs(20) should return ("Low", "Increase humidity with misting")

## **5.4 Care Instructions Generation Function**

**6. Write a Python function to generate comprehensive care instructions.** Define: generate\_care\_instructions(plant\_type, season, temperature, humidity)

The function should:

* Accept four parameters: plant\_type (int), season (int), temperature (float), humidity (int)
* Validate all inputs using the same validation logic as individual functions
* Check for boolean inputs first for integer parameters
* Raise appropriate ValueError messages for each invalid input type or range
* Calculate base watering days using calculate\_watering\_schedule(plant\_type)
* Adjust watering for season using adjust\_for\_season(base\_days, season)
* Get environmental status using check\_temperature(temperature) and determine\_humidity\_needs(humidity)
* Get sunlight requirements using get\_sunlight\_requirement(plant\_type)
* Build instructions list starting with basic information:
  + "Watering Schedule: Every {adjusted\_days} days"
  + "Sunlight Requirement: {sunlight\_req}"
  + "Temperature Status: {temp\_status}"
  + "Humidity Level: {humidity\_level}"
  + "Special Care Instructions:"
* Add plant-specific care using if-elif-else structure:
  + if plant\_type == 1: append "Avoid overwatering" (Succulent)
  + elif plant\_type == 2: append "Maintain high humidity" (Tropical)
  + elif plant\_type == 3: append "Remove dead flowers regularly" (Flowering)
  + elif plant\_type == 4: append "Harvest regularly to promote growth" (Herb)
* Add seasonal advice using if-elif structure:
  + if season == 2: append "Increase watering frequency" (Summer)
  + elif season == 4: append "Reduce watering frequency" (Winter)
* Add environmental advice using if-elif structure:
  + if temperature > 30.0: append "Provide shade and increase watering"
  + elif temperature < 10.0: append "Protect from cold and reduce watering"
* Append humidity advice from determine\_humidity\_needs() function
* Join all instructions with newlines: return "\n".join(instructions)
* Return complete care instructions as a formatted string
* Example: Function should return comprehensive multi-line care guide

## **5.5 Main Program Function**

**7. Write a Python main program to demonstrate the plant care advisory system.** Define: main() function or use if \_\_name\_\_ == "\_\_main\_\_": block

The function should:

* Use try-except block to handle potential ValueError exceptions
* Print system header: "=== Plant Care Advisory System ==="
* Print welcome message: "Welcome to BloomWise Solutions Plant Care Advisor!"
* Display plant type options: "Plant Types: 1=Succulent, 2=Tropical, 3=Flowering, 4=Herb"
* Get plant type input: plant\_type = int(input("Enter plant type (1-4): "))
* Display season options: "Seasons: 1=Spring, 2=Summer, 3=Fall, 4=Winter"
* Get season input: season = int(input("Enter current season (1-4): "))
* Get temperature input: temperature = float(input("Enter temperature in Celsius (-10.0 to 50.0): "))
* Get humidity input: humidity = int(input("Enter humidity percentage (0-100): "))
* Generate care instructions using generate\_care\_instructions(plant\_type, season, temperature, humidity)
* Display results with header: "=== Your Plant Care Instructions ==="
* Print the complete care instructions
* Handle ValueError exceptions with message: print(f"Input Error: {e}")
* Handle general exceptions with message: print(f"An error occurred: {e}")
* Example execution should provide complete interactive plant care consultation

## **5.6 Implementation Notes**

**Key Requirements:**

* All conditional functions must use proper if-elif-else structures as specified
* Input validation must check for boolean types before integer types due to inheritance
* Error messages must match exactly as specified in the constraints
* Return values must be exact strings/numbers as defined in the SRS
* Environmental and seasonal advice must be integrated into final care instructions
* The system should handle edge cases like minimum watering days (1 day minimum for summer)
* All functions should work together seamlessly in the main program flow

**6. Execution Steps to follow:**

1. Run the program

2. Enter plant type

3. Enter current season

4. Enter temperature

5. Enter humidity level

6. View complete plant care advisory report

Execution Steps to Follow:

● All actions like build, compile, running application, running test cases will be through Command Terminal.

● To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal

● This editor Auto Saves the code

● If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B -command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.

● These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.

● To launch application: python3 filename.py

● To run Test cases: python3 -m unittest

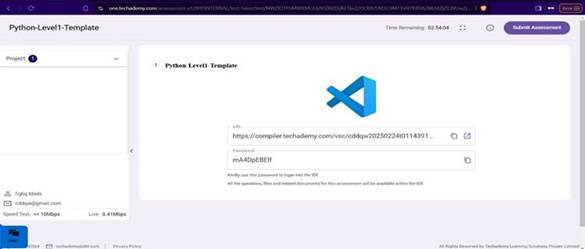
● Before Final Submission also, you need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

Screen shot to run the program

To run the application

Python3 filename.py

To run the testcase python3 -m unittest



● Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.