SCALE FOR PROJECT PYTHON MODULE 02

You should evaluate 1 student in this team

Git repository

`git@vogsphere.1337.ma`

Introduction

- Remain polite, courteous, respectful and constructive
throughout the evaluation process. The well-being of the community
depends on it.

- Identify with the person (or the group) evaluated the eventual
dysfunctions of the work. Take the time to discuss
and debate the problems you have identified.

- You must consider that there might be some difference in how your
peers might have understood the project's instructions and the
scope of its functionalities. Always keep an open mind and grade
him/her as honestly as possible. The pedagogy is valid only and
only if peer-evaluation is conducted seriously.

Guidelines

- Only grade the work that is in the student or group's
GiT repository.

- Double-check that the GiT repository belongs to the student
or the group. Ensure that the work is for the relevant project
and also check that "git clone" is used in an empty folder.

- Check carefully that no malicious aliases was used to fool you
and make you evaluate something other than the content of the
official repository.

- To avoid any surprises, carefully check that both the evaluating
and the evaluated students have reviewed the possible scripts used
to facilitate the grading.

- If the evaluating student has not completed that particular
project yet, it is mandatory for this student to read the
entire subject prior to starting the defence.

- Use the flags available on this scale to signal an empty repository,
non-functioning program, a norm error, cheating etc. In these cases,
the grading is over and the final grade is 0 (or -42 in case of
cheating). However, with the exception of cheating, you are
encouraged to continue to discuss your work (even if you have not
finished it) in order to identify any issues that may have caused
this failure and avoid repeating the same mistake in the future.

- Remember that for the duration of the defence, no segfault,
no other unexpected, premature, uncontrolled or unexpected
termination of the program, else the final grade is 0. Use the
appropriate flag.
You should never have to edit any file except the configuration file if it exists.
If you want to edit a file, take the time to explicit the reasons with the
evaluated student and make sure both of you are okay with this.

- You must also verify the absence of memory leaks. Any memory allocated on the heap must
be properly freed before the end of execution.
You are allowed to use any of the different tools available on the computer, such as
leaks, valgrind, or e_fence. In case of memory leaks, tick the appropriate flag.

Attachments

subject.pdf

Preliminaries

Basics

- Only grade the work that is in the learner's Git repository
- Ensure the Git repository belongs to the learner
- Check that no malicious aliases are used
- Verify the project structure matches the subject requirements

Are all preliminary checks satisfied?

Yes No

General Instructions

During the evaluation of each exercises, ensure the following
general instructions are met:
- Programs must be written in Python 3.10 or higher
- The code must respect the flake8 linter standards (with no warnings or errors)
- Each exercise should be in its own file as specified
- Code must include simple docstrings for functions and classes
- Programs must never crash unexpectedly (no uncaught exceptions)

Yes No

Exercises

Exercise 0 - Agricultural Data Validation Pipeline

Check ft_first_exception.py:

Basic error handling:
- Function check_temperature(temp_str) exists and works correctly
- Handles ValueError when input is not a number (e.g., "abc")
- Validates temperature range (0 to 40 degrees for plants)
- Raises ValueError with descriptive messages for invalid ranges
- Returns valid temperature when input is correct

Test function:
- test_temperature_input() function exists
- Tests good input ("25")
- Tests bad input ("abc")
- Tests extreme values ("100", "-50")
- Program continues running after errors
- Output matches expected format: "=== Garden Temperature Checker ==="
- Ensure the test_temperature_input() function demonstrates expected

outputs and error handling as described in the subject.

Does the program demonstrate agricultural data validation with basic try/except correctly?

Yes No

Exercise 0 - Understanding check

Understanding check:
- Can the learner explain what happens when you try to convert "abc" to a number?
- Do they understand the difference between raising and catching exceptions?
- Can they explain why the program doesn't crash when errors occur?

Does the learner understand basic exception handling concepts?

Yes No

Exercise 1 - Different Types of Problems

Check ft_different_errors.py:

Error type demonstration:
- garden_operations() function creates different error scenarios
- Demonstrates ValueError (bad data conversion)
- Demonstrates ZeroDivisionError (division by zero)
- Demonstrates FileNotFoundError (missing file)
- Demonstrates KeyError (missing dictionary key)

Error handling:
- test_error_types() function exists
- Catches each error type individually with specific except blocks
- Shows how to catch multiple error types together
- Program continues after each error
- Output matches expected format: "=== Garden Error Types Demo ==="

Does the program demonstrate different built-in exception types correctly?

Yes No

Exercise 1 - Understanding check

Understanding check:
- Can the learner explain why Python has different error types?
- Do they understand how to catch multiple error types with one except block?
- Can they give examples of when each error type might occur?

Does the learner understand Python's built-in exception types?

Yes No

Exercise 2 - Making Your Own Error Types

Check ft_custom_errors.py:

Custom exception classes:
- GardenError class exists and inherits from Exception
- PlantError class exists and inherits from GardenError
- WaterError class exists and inherits from GardenError
- Classes are simple and well-structured

Usage demonstration:
- Functions that raise custom errors in different situations
- test_custom_errors() function exists (or similar test function)
- Shows catching specific custom error types (PlantError, WaterError)
- Demonstrates inheritance (catching GardenError catches all garden errors)
- Output matches expected format: "=== Custom Garden Errors Demo ==="

Does the program demonstrate custom exception creation and inheritance correctly?

Yes No

Exercise 2 - Understanding check

Understanding check:
- Can the learner explain when to create custom exceptions vs using built-in ones?
- Do they understand how inheritance works with exception classes?
- Can they explain the benefit of having a base exception class?

Does the learner understand custom exception design principles?

Yes No

Exercise 3 - Finally Block - Always Clean Up

Check ft_finally_block.py:

Finally block usage:
- water_plants(plant_list) function exists
- Uses try/except/finally structure correctly
- Finally block always executes (prints cleanup message)
- Handles errors appropriately in try block
- Demonstrates cleanup happens even when errors occur

Test demonstration:
- test_watering_system() function exists
- Tests normal operation (no errors)
- Tests error scenario (None in plant list)
- Shows cleanup happens in both cases

- Output matches expected format: "=== Garden Watering System ==="

Does the program demonstrate finally block usage correctly?

Yes No

Exercise 3 - Understanding check

Understanding check:
- Can the learner explain why cleanup is important even when errors happen?
- Do they understand when the finally block executes?
- Can they give examples of resources that need cleanup?

Does the learner understand the purpose and behavior of finally blocks?

Yes No

Exercise 4 - Raising Your Own Errors

Check ft_raise_errors.py:

Error raising:
- check_plant_health(plant_name, water_level, sunlight_hours) function exists
- Validates plant name (not empty)
- Validates water level (1-10 range)
- Validates sunlight hours (2-12 range)
- Raises ValueError with descriptive messages for invalid inputs
- Returns success message for valid inputs

Test demonstration:
- test_plant_checks() function exists
- Tests good values (should succeed)
- Tests bad plant name, water level, and sunlight hours
- Catches and handles each error appropriately
- Output matches expected format: "=== Garden Plant Health Checker ==="

Does the program demonstrate raising custom errors correctly?

Yes No

Exercise 4 - Understanding check

Understanding check:
- Can the learner explain when their program should raise errors?
- Do they understand how to create helpful error messages?
- Can they explain the difference between handling and raising errors?

Does the learner understand when and how to raise errors?

Yes No

Exercise 5 - Garden Management System

Check ft_garden_management.py:

Integration of concepts:
- A GardenManager class exists with appropriate methods
- Uses custom exception classes (GardenError, PlantError, WaterError)
- add_plant() method validates input and raises errors appropriately
- water_plants() method uses try/finally for cleanup
- check_plant_health() method raises appropriate errors
- Demonstrates error recovery and graceful handling

System functionality:
- test_garden_management() function exists
- Tests adding plants (good and bad inputs)
- Tests watering system with cleanup
- Tests plant health checking
- Shows error recovery mechanisms
- Output matches expected format: "=== Garden Management System ==="

Does the program integrate all error handling concepts effectively?

Yes No

Exercise 5 - Understanding check

Understanding check:
- Can the learner explain how all error handling techniques work together?
- Do they understand what makes a program robust and reliable?
- Can they discuss the benefits of defensive programming?

Does the learner understand integrated error handling design?

Yes No

Generale Code Quality and Understanding

Code Quality and Understanding

Evaluate overall understanding and code quality:

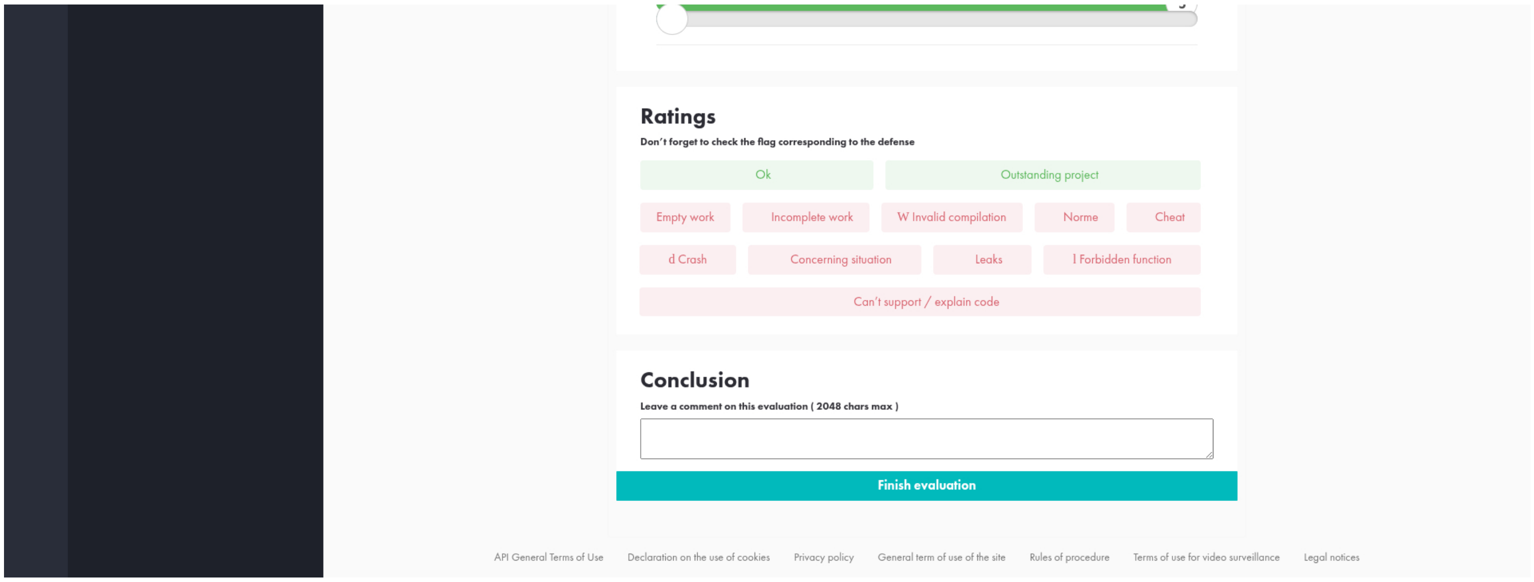
Error Handling Mastery:
- Can the learner explain the basic try/except/finally structure?
- Do they understand when to use built-in vs custom exceptions?
- Can they explain the benefits of proper error handling?
- Do they understand exception inheritance and how to use it?

Code Quality:
- Is the code well-structured and readable?
- Are error messages helpful and descriptive?
- Does the code follow good Python practices?
- Are the solutions appropriate for a Module 02 level?

Practical Application:
- Can the learner apply these concepts to other projects?
- Do they understand the progression from basic to integrated error handling?
- Can they explain why error handling is important in real programs?

Note: This is for feedback and doesn't fail the evaluation.
Focus on understanding and learning progress.

Rate it from 0 (failed) through 5 (excellent)



Ratings

Don't forget to check the flag corresponding to the defense

Conclusion

Leave a comment on this evaluation (2048 chars max)