SCALE FOR PROJECT PYTHON MODULE (/PROJECTS/PYTHON-MODULE-01)

You should evaluate 1 student in this team

★

Git repository

```
git@vogsphere.1337.ma:vogsphere/intra-uuid-b01f05a5-5543-4861-9
```

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 learner or group's GiT repository.

- Double-check that the GiT repository belongs to the learner 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 learners have reviewed the possible scripts used to facilitate the grading.

- If the evaluating learner has not completed that particular project yet, it is mandatory for this learner 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 learner 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 (https://cdn.intra.42.fr/pdf/pdf/191337/en.subject.pdf)

Preliminaries

Basics

- Only grade the work that is in the learner's or group's Git repository.
- Check that the following files exist and are properly named: • ex0/ft_garden_intro.py • ex1/ft_ • ex2/ft_plant_growth.py • ex3/ft_plant_factory.py • ex4/ft_garden_security.py • ex5/ft_plant_t ex6/ft_garden_analytics.py
    - Verify no additional unauthorized files are present. If any file is missing or incorrectly n evaluation stops here.

Yes No

General Instructions

During the evaluation of each exercises, ensure the following general instructions are met:

- The code must be written in Python 3.10+ or higher.
- The code must resspect the flake8 linter standards with no errors.
- A proper naming convention for functions, classes and methods is followed.
- A proper docstring is included in the code.
- Type hints are encouraged for functions and methode but not mandatory.
- the learner don't need to handle input validation unless explicitly mentioned
- All programs must run without errors or crashes.

If any of these general instructions are not met, the evaluation must be stop.

Yes No

Exercises

Exercise 0 - Planting Your First Seed (Program Structure)

Test the ft_garden_intro.py program:

Basic functionality:
• Run the program: python ex0/ft_garden_intro.py
• Verify the program uses the if **name** == "**main**": pattern
• Check that plant information is stored in simple variables
• Verify the program displays information using print()
• Ask the learner to explain what if **name** == "**main**": means
• Ask the learner why this pattern is important in Python

Key concepts to verify:
• Correct use of if **name** == "**main**":
• Variables to store data (name, height, age)
• Print statements to display information

Understanding check:
• Ask: "What happens if you remove the if **name** == '**main**': line?"
• Ask: "When would this code NOT run?"
• Verify they understand this is the program's entry point

Edge cases:
• Output is readable and well-formatted
• Code structure is clean and simple

Does the program demonstrate understanding of basic Python program structure?

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

5

Exercise 1 - Garden Data Organizer (Classes)

Test the ft_garden_data.py program:

Basic functionality:
• Run the program: python ex1/ft_garden_data.py
• Verify the program defines and uses a CLASS structure
• Check that plant data is organized using class instances
• Verify the program displays plant information correctly
• Ask the learner to explain what a class is and why they used it
• Ask the learner to show where they defined the class and created instances

Key concepts to verify:
• Class definition with proper syntax
• Object instantiation (creating instances)
• Basic attributes (plant properties)
• Simple display of plant data

Edge cases:
• Output is readable and well-formatted
• Class structure is logical and appropriate

Does the program demonstrate understanding of classes as data organization tools?

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

5

Exercise 2 - Plant Growth Simulator (Methods)

Test the ft_plant_growth.py program:

Basic functionality:
• Run the program: python ex2/ft_plant_growth.py
• Verify the program uses METHODS to perform actions on plants
• Check that plants can grow and age over time
• Verify the simulation shows changes over multiple days
• Ask the learner to explain what methods are and why they used them
• Ask the learner to demonstrate calling methods on plant objects

Key concepts to verify:
• Method definition within classes
• Method calls on object instances
• State changes through method execution
• Simulation logic using methods

Test scenarios:
• Plants grow when methods are called
• Plant state changes are persistent
• Multiple method calls work correctly
• Output shows progression over time

Does the program demonstrate understanding of methods as actions objects can perform?

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

5

Exercise 3 - Plant Factory (Initialization)

Test the ft_plant_factory.py program:

Basic functionality:
• Run the program: python ex3/ft_plant_factory.py
• Verify the program uses **init** method for object initialization
• Check that plants are created with initial values
• Verify multiple plants can be created with different properties
• Ask the learner to explain what **init** does and why it's needed
• Ask the learner to show how they set initial values for plant attributes

Key concepts to verify:
• **init** method definition and usage
• Parameter passing during object creation
• Initial state setting for new objects
• Multiple object creation with different initial values

Test scenarios:
• Plants are created with specified initial properties
• Different plants have different initial values
• Factory pattern creates multiple plants efficiently
• All created plants function correctly

Does the program demonstrate understanding of object initialization and construction?

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

5

Exercise 4 - Garden Security System (Encapsulation)

Test the ft_garden_security.py program:

Basic functionality:
• Run the program: python ex4/ft_garden_security.py
• Verify the program uses ENCAPSULATION to protect data
• Check that invalid data is rejected (negative values, etc.)
• Verify access to plant data is controlled through methods
• Ask the learner to explain what encapsulation is and why it's important
• Ask the learner to show how they protected plant data from invalid modifications

Key concepts to verify:
• Protected attributes (using underscore convention)
• Getter and setter methods for controlled access
• Data validation before allowing changes
• Error handling for invalid operations

Test scenarios:
• Valid data updates work correctly
• Invalid data is rejected with appropriate messages
• Direct attribute access is discouraged/prevented
• Plant data integrity is maintained

Does the program demonstrate understanding of data protection and controlled access?

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

5

Exercise 5 - Specialized Plant Types (Inheritance)

Test the ft_plant_types.py program:

Basic functionality:
• Run the program: python ex5/ft_plant_types.py
• Verify the program uses INHERITANCE to create specialized plant types
• Check that different plant types (Flower, Tree, Vegetable) exist
• Verify each type has both common and specialized features
• Ask the learner to explain inheritance and why they used it
• Ask the learner to show the parent-child relationships between classes

Key concepts to verify:
• Base class (Plant) with common features
• Derived classes (Flower, Tree, Vegetable) with specialized features
• super() calls to parent class methods
• Method overriding for specialized behavior

Test scenarios:
• All plant types share common functionality
• Each type has unique features (blooming, shade, nutrition)
• Inheritance chain works correctly
• Specialized methods function as expected

Does the program demonstrate understanding of inheritance and code reuse?

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

0

Exercise 6 - Garden Analytics Platform (Advanced OOP)

Test the ft_garden_analytics.py program:

Basic functionality:
• Run the program: python ex6/ft_garden_analytics.py
• Verify the program demonstrates ADVANCED OOP concepts
• Check for nested classes, inheritance chains, class methods
• Verify distinction between member and non-member functions
• Ask the learner to explain the advanced concepts they implemented

Key concepts to verify:
• Nested classes (class within a class)
• Inheritance chains (A -> B -> C)
• Class methods (using classmethod()) vs instance methods
• Static methods (using staticmethod()) vs regular functions
• Non-member functions (outside any class)

Test scenarios:
• Nested class functionality works correctly
• Inheritance chain demonstrates proper super() usage
• Class methods can create instances without decorators
• Static methods work without instance using traditional syntax
• Non-member functions operate on class instances

Advanced concepts discussion:
• Ask about when to use each type of method
• Discuss the benefits of nested classes
• Explain the inheritance chain design choices

Does the program demonstrate mastery of advanced OOP concepts?

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

General Code Quality and Programming Understanding

Overall Understanding and Code Quality

Evaluate the overall understanding and code quality:

Programming Concept Mastery:
• Can the learner explain how Python programs are structured?
• Do they understand the difference between classes and objects?
• Do they understand when and why to use inheritance?
• Can they explain encapsulation benefits?
• Do they understand method types (instance, class, static)?

Code Quality:
• Is the code well-structured and readable?
• Are programming principles applied appropriately?
• Are the garden/plant metaphors used consistently?
• Does the progression from simple to complex make sense?

Discovery-Based Learning:
• Ask the learner how they discovered each concept
• Did they understand why each concept was needed?
• Can they apply these concepts to other domains?

Note: This is for feedback purposes and doesn't fail the evaluation.
Focus on understanding rather than perfect implementation.

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

Ratings

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

✔ Ok ★ Outstanding project

Empty work Incomplete work Invalid compilation Norme Cheat

Concerning situation Leaks Forbidden function Can’t support /

Conclusion

Leave a comment on this evaluation (2048 chars max)

Finish evaluation