SCALE FOR PROJECT PYTHON MODULE (/PROJECTS/PYTHON-MODULE-03)

You should evaluate 1 student in this team

★

Git repository

```
git@vogsphere.42malaga.com:vogsphere/intra-uuid-d1988ea1-c8b8-4
```

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 (https://cdn.intra.42.fr/pdf/pdf/198393/en.subject.pdf)

data_quest_tools.tar.gz (https://cdn.intra.42.fr/document/document/46053/data_quest_tools.tar

Preliminaries

Basics

- Only grade the work that is in the learner's GiT repository.
- Check that the Git repository belongs to the learner, and that the work is the learner's.
- Check that only the requested files are present at the root of the repository.
- Each exercise must be graded in the order of the subject.

Yes No

Exercises

Exercise 0 - Command quest

Check that the file ft_command_quest.py exists and demonstrates command-line argument proces

- The program correctly accesses sys.argv
- It displays the program name (sys.argv[0])
- It counts and displays the total number of arguments
- It handles cases with no arguments provided
- It displays each argument individually when provided
- Test with: python3 ft_command_quest.py
- Test with: python3 ft_command_quest.py hello world 42
- Test with: python3 ft_command_quest.py "Data Quest"

Ask the learner to explain what sys.argv contains and the difference between program name and a
Does the program correctly demonstrate command-line argument processing?

Yes No

Exercise 1 - Score Cruncher

Check that the file ft_score_analytics.py exists and contains the required functionality:

- The program accepts command line arguments as player scores
- It processes lists of scores correctly
- It handles invalid input gracefully (non-numeric arguments)
- It displays analytics including total, average, high/low scores
- Test with: python3 ft_score_analytics.py 100 250 180 90 300
- Test with invalid input: python3 ft_score_analytics.py 100 abc 200
- Test with no arguments: python3 ft_score_analytics.py

Ask the learner to explain how sys.argv works and how they process the list of scores.
Does the program work correctly in all test cases?

Yes No

Exercise 2 - Position Tracker

Check that the file ft_coordinate_system.py exists and demonstrates tuple operations:

- The program creates and manipulates 3D coordinate tuples (x, y, z)
- It demonstrates tuple unpacking (x, y, z = coordinate)
- It calculates 3D Euclidean distances correctly
- It includes custom exception handling with tuple unpacking of exception arguments
- Test coordinate creation and unpacking
- Test distance calculations between 3D coordinates
- Test error handling with invalid coordinates

Ask the learner to explain tuple unpacking and how exception arguments are unpacked.
Does the program correctly demonstrate tuple concepts for 3D coordinates?

✓ Yes ✗ No

Exercise 3 - Achievement Hunter

Check that the file ft_achievement_tracker.py exists and uses sets effectively:

- The program removes duplicate achievements using sets
- It finds common achievements between players (intersection)
- It identifies unique achievements per player (difference)
- It calculates rare achievements (owned by few players)
- It analyzes player communities based on shared achievements
- Test with sample player data containing duplicate achievements

Ask the learner to explain set operations (union, intersection, difference) and their use cases.
Does the program demonstrate effective use of sets for data deduplication?

✓ Yes ✗ No

Exercise 4 - Inventory Master

Check that the file ft_inventory_system.py exists and uses dictionaries comprehensively:

- The program accepts command-line arguments in format item:quantity
- It stores inventory data in a dictionary structure (item names as keys, quantities as values)
- It demonstrates nested dictionaries for item categorization (abundant, moderate, scarce)
- It calculates inventory statistics (total items, percentages, most/least abundant)
- It uses dictionary methods: keys(), values(), items(), get(), update()
- Test with: python3 ft_inventory_system.py sword:1 potion:5 shield:2 armor:3
- Test with no arguments to verify usage message is displayed
- Test with invalid format to verify error handling

Ask the learner to explain dictionary operations and how nested dictionaries organize data by categ
Does the program effectively use dictionaries for inventory management?

✓ Yes ✗ No

Exercise 5 - Stream Wizard

Check that the file ft_data_stream.py exists and demonstrates generators and for-in loops:

- The program creates generators that yield data on-demand
- It processes data streams using for-in loops without loading everything into memory
- It includes mathematical generators (Fibonacci, primes)
- It demonstrates memory efficiency compared to lists
- It processes game events in batches using generators
- Test the generators and verify they don't store all data in memory

Ask the learner to explain how generators work and why they're memory efficient.
Does the program correctly demonstrate generator concepts and for-in loop usage?

✓ Yes ✗ No

Exercise 6 - Data Alchemist

Check that the file ft_analytics_dashboard.py exists and uses all types of comprehensions:

- The program uses list comprehensions for data filtering and transformation
- It uses dictionary comprehensions for creating mappings and analytics
- It uses set comprehensions for unique data analysis
- It combines comprehensions with complex data processing
- It generates comprehensive analytics from gaming data
- Test the dashboard with sample data and verify all comprehension types work

Ask the learner to explain the differences between list, dict, and set comprehensions.
Does the program demonstrate mastery of comprehensions for data processing?

✓ Yes ✗ No

General Review

Generale Understanding

Evaluate the learner's overall understanding:

- Can they explain the differences between lists, tuples, sets, and dictionaries?
- Do they understand when to use each data structure?
- Can they explain how generators save memory compared to lists?
- Do they understand tuple unpacking and its applications?
- Can they explain comprehensions and their benefits?
- Do they understand command line argument processing?

The learner should demonstrate solid understanding of Python's core data structures and their appropriate use cases in data engineering scenarios.

Yes No

Code Quality

Assess the overall code quality:

- Is the code written in Python 3.10 or higher?
- Does the code adhere to the flake8 linter standards (no errors)?
- Are type hints REQUIRED for all functions and methods (parameters and return values)?
- Docstrings are NOT required for this module.
- Is the code well-structured and readable?
- Are variable names meaningful and descriptive?
- Is error handling implemented appropriately?
- Are the solutions efficient for their intended purpose?
- Does the code follow Python best practices?

The code should demonstrate good programming practices and clear understanding of the concepts being taught.

Yes No

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

API General Terms of Use (https://profile.intra.42.fr/legal/terms/33) Declaration on the use of cookies (https://profile.intra.42.fr/legal/terms/2) Privacy policy (https://profile.intra.42.fr/legal/terms/5) General term of use of the site (https://profile.intra.42.fr/legal/terms/6) Rules of pro (https://profile.intra.42