(https://profile.intra.42.fr/searches) migugonz

(https://profile.intra.42.fr)

SCALE FOR PROJECT PYTHON MODULE (/PROJECTS/PYTHON-MODULE-06)

You should evaluate 1 student in this team

★

Git repository

```
git@vogsphere.42malaga.com:vogsphere/intra-uuid-c0495b99-f4c9-4
```

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/197463/es.subject.pdf)

Preliminaries

Basics

- Only grade the work that is in the learner's or group's Git repository.
- Check that only the requested files are available in the git repository.
- Verify the learner has the required file structure:
  - ft_sacred_scroll.py (at root)
  - ft_import_transmutation.py (at root)
  - ft_pathway_debate.py (at root)
  - ft_circular_curse.py (at root)
  - alchemy/ directory with proper package structure
- If any required files are missing, the evaluation stops here.

Yes No

Part I - The Sacred Scroll (<code> __init__.py
mastery)

Sacred Scroll experiment

Test the Sacred Scroll experiment:

- Run: python3 ft_sacred_scroll.py
- Verify the alchemy package has `__init__.py` that controls access
- Check that some elemental spells are exposed, others are hidden
- Confirm package metadata (`__version__` , `__author__`) is present
- Test that direct module import works differently than package import
- Verify the learner understands how `__init__.py` controls package interface

Does the Sacred Scroll properly demonstrate `__init__.py` package control?

Yes No

Code quality

Code Quality Check:

- Is the alchemy/elements.py file properly structured?
- Are the four elemental functions (create_fire, create_water, create_earth, create_air) implem
- Does the `__init__.py` file demonstrate selective exposure of functions?
- Is the code clean and well-organized?

Yes No

Part II - Import Transmutation (from...import m

Import Transmutation experiment

Test the Import Transmutation experiment:

- Run: python3 ft_import_transmutation.py
- Verify different import styles are demonstrated:
  - Full module import (import alchemy.elements)
  - Specific imports (from alchemy.elements import create_fire)
  - Aliased imports (from alchemy.potions import healing_potion as heal)
  - Multiple imports (from module import func1, func2)
- Check that potions.py uses elemental functions properly

- Confirm the learner understands the differences between import methods

Does the transmutation properly demonstrate various import techniques?

✓ Yes ✕ No

Potion Brewing

Potion Brewing Check:

- Are the potion functions (healing_potion, strength_potion, etc.) implemented?
- Do potions properly use elemental functions from elements.py?
- Is the import usage in potions.py appropriate and clean?

✓ Yes ✕ No

Part III - The Great Pathway Debate (absolute v relative imports)

Pathway Debate experiment

Test the Pathway Debate experiment:

- Run: python3 ft_pathway_debate.py
- Verify the transmutation package structure exists
- Check that basic.py uses absolute imports correctly
- Check that advanced.py uses relative imports correctly (.basic, ..potions)
- Confirm both absolute and relative pathways work properly
- Verify the learner understands when to use each approach

Does the pathway debate properly demonstrate absolute vs relative imports?

✓ Yes ✕ No

Transmutation Structure

Transmutation Structure Check:

- Is the alchemy/transmutation/ package properly structured?
- Does basic.py contain lead_to_gold() and stone_to_gem() functions?
- Does advanced.py contain philosophers_stone() and elixir_of_life() functions?
- Are the import statements in each file appropriate for their purpose?

✓ Yes ✕ No

Part IV - Breaking the Circular Curse (circular dependency resolution)

Circular Curse experiment

Test the Circular Curse Breaking experiment:

- Run: python3 ft_circular_curse.py
- Verify the circular dependency problem is explained
- Check that **one** curse-breaking technique is demonstrated:
  - Late imports (importing inside functions)
  - Dependency injection (passing dependencies as parameters)
  - Shared utilities (breaking the dependency chain)
- Confirm the grimoire package structure works without circular imports
- Verify the learner understands why circular dependencies are problematic

Does the curse-breaking properly demonstrate circular dependency resolution?

✓ Yes ✕ No

Grimoire Structure

Grimoire Structure Check:

- Is the alchemy/grimoire/ package properly structured?
- Does spellbook.py contain spell recording functionality?
- Does validator.py contain ingredient validation functionality?

✓ Yes ✗ No

Overall Understanding and Code Quality

Import Mastery Assessment

- Can the learner explain the four sacred mysteries of Python imports?
- Does the learner understand when to use different import styles?
- Can they explain the trade-offs between absolute and relative imports?
- Do they understand how to identify and resolve circular dependencies?
- Is the overall code organization clean and professional?

✓ Yes ✗ No

Alchemical Laboratory Quality

- Is the package structure logical and well-organized?
- Are all `__init__.py` files properly configured?
- Do the demonstration scripts clearly show the concepts being taught?
- Is the code readable and well-documented?
- Does this activity demonstrate mastery of Python import mechanisms?

✓ 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