SCALE FOR PROJECT PYTHON MODULE 02 (/PROJECTS/PYTHON-MODULE-02)

You should evaluate 1 student in this team

★

Git repository

```
git@vogsphere-v2.1337.ma:vogsphere/intra-uuid-e699e4da-ef82
```

Comments

This third module is the evolution of two original projects created by the
Paris-based student organization 42 AI.
They are named Bootcamp Python and Bootcamp Machine Learning.
active members of 42 AI re-designed both of them for the school curriculum.

Bootcamps has been developped between August 2019 and March/April 2020.
Active 42AI members organized severals sessions of 2 weeks to 42Paris students
to offer them the possibility to get famliar with Python and basics concepts
of machine learning.

The success of those sections brings the pedagogy to accept the idea to integrate
the 2 bootcamps to the curriculum (initial discussion (01-05/2019) with 42 Paris
pedago team highlighted a categorical opidea)

The transcription had been realized over the direction of Matthieu David
several 42AI members contributed to the redaction on the correction scales.
For futur corrections on the scale, please contact the 42AI association via
contact@42ai.fr or the current 42AI pedagogical supervisor.

Introduction

The Bootcamp Python and Bootcamp Machine Learning were originally created by
[42AI](https://github.com/42-AI) active members and were adapted to 'piscine'
format for the school 42 curriculum.
For any issue or suggestion: [42Paris](https://github.com/42-AI/bootcamp_python/issues) and
[42AI](https://github.com/42-AI/bootcamp_machine-learning/issues).

As usual, you have to observe the following courtesy rules:

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

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

- You must consider that there might be some differences in the understanding of and approach to project instructions, and the scope of its functionalities, between you and your peers. Always remain open-minded and grade them as fairly as possible. The pedagogy is valid only and only if peer-evaluation is conducted seriously.

The goal of this module is to get started with the Python language.
You will study map, reduce, filter, args and kwargs ...

Disclaimer

The serie of modules started to be produce at the time of the release of Python 3.7. Students are free to use later version of Python as long as they verified the producted code complies with all the aspects precised in the subjects.
As a consequence we recommend to students to perform the modules with the the Python version 3.7 (but this is just an advice).
Version can be checked with the command ```python -V```.

Guidelines

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

- Double-check that the GiT repository does belong to the student. Ensure that the work is the one expected for the corrected exercise and don't forget to verify that the command "git clone" is run in an empty folder.

- Check carefully that no malicious aliases were used to make you evaluate files that are not from 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 them to read the entire subject prior to starting the defense.

- Use the flags available on this scale to signal an empty repository,

non-functioning program, a Norm error (specified next in general rules),
cheating, and so forth.
In these cases, the grading is over and the final grade is 0,
or -42 in case of cheating. However, except the exception of cheating, you
are encouraged to continue to discuss your work even if the later is in
progress in order to identify any issues that may have caused the project
failure and avoid repeating the same mistake in the future.

- Use the appropriate flag.

- Remember that for the duration of the defense, no other unexpected,
premature, or uncontrolled termination of the program, else the final
grade is 0.

- You should never have to edit any file except the configuration file if
the latter exists. If you want to edit a file, take the time to explain why
with the evaluated student and make sure both of you agree on this.

- The Norm: You will follow the PEP 8 standards.

- The function eval is never allowed.

- Your exercises are going to be evaluated by other students,
make sure that your variable names and function names are appropriate and civil.

Attachments

subject.pdf (https://cdn.intra.42.fr/pdf/pdf/42654/en.subject.pdf)

good.csv (/uploads/document/document/7260/good.csv)

bad.csv (/uploads/document/document/7261/bad.csv)

Exercise 00: Map, filter, reduce

In the error management question, function refers to an object function (a lambda function as instance) and iter refers to an iterable object. As mentioned in the subject, we do not expect the identical exception messages to be exactly the same than those of the functions `map`, `filter` and `reduce`. The goal of the exercise is to work on the built-in functions `map`, `filter` and `reduce`.

Errors management

Perform the following tests, you should observed a raised error for each case:
- ft_map function:
- Pass None as function parameter:
- `ft_map(None, iterable)`: you should get the representation generator object id ("< generator object ft_map at hexa_adress>")

- `list(ft_map(None, iterable))`: You should get a TypeError Exception.
- Pass None for as iterable parameter:
- `ft_map(function, None)`: you should get a TypeError Exception.
- Pass a non iterable object (an integer for example):
- `ft_map(function, 2)`: you should get a TypeError Exception.
- Pass a non function object as function parameter:
- `ft_map("Bob", iterable)`: you should get a TypeError Exception.

- ft_filter function:
- Pass None as function parameter:
- `ft_filter(None, iterable)`: you should get the representation generator object id ("< something at hexa_adress>")
- `list(ft_filter(None, iterable))`: You should get an identic copy of the iterable.
- Pass None for as iterable parameter:
- `ft_filter(function, None)`: you should get a TypeError Exception.
- Pass a non iterable object (an integer for example):
- `ft_filter(function, 2)`: you should get a TypeError Exception.
- Pass a non function object as function parameter:
- `ft_filter("Alice", iterable)`: you should get the representation generator object id ("< something at hexa_adress>")
- `list(ft_filter("Alice", iterable))`: you should get a TypeError Exception.

- ft_reduce function:
- Pass None as function parameter:
- `reduce(None, iterable)`: you should get a TypeError Exception.
- Pass None as iterable parameter:
- `reduce(function, None)`: you should get a TypeError Exception.
- Pass a non function object as function parameter:
- `reduce(None, iterable)`: you should get a TypeError Exception.
- Pass an empty list as iterable parameter:
- reduce(function, []): you should get a TypeError Exception

✓ Yes ✗ No

Basic tests

Perform some basic test such as the following:
- `list(ft_map(lambda x: x + 2, []))`:
you should get `[]`.
- `list(ft_map(lambda x: x + 2, [1]))`:
you should get `[3]`.
- `list(ft_map(lambda x: x ** 2, [1, 2, 3, 4, 5]))`:
you should get `[1, 4, 9, 16, 25]`.
- `list(ft_filter(lambda x: x <= 1, []))`:
you should get `[]`.
- ft_filter(lambda x: x <= 0, [1]):
you should get `[]`.
- ft_filter(lambda x: x <= 2, [1, 2, 3, 4, 5]):
you should get `[1, 2]`.
- ft_reduce((lambda x, y: x + y), [1]):

you should get `[1]`.
- ft_reduce((lambda x, y: x + y), [1, 2, 3, 4]):
you should get `10`.
- ft_reduce((lambda x, y: x * y), [1, 2, 3, 4]):
you should get `24`.
Feel free to realize few more tests.

✓ Yes ✕ No

Exercise 01: args and kwargs

The goal of the exercise is to discover and manipulate `*args` and `**kwargs`.

Basic tests

- With None as unique parameter:
```
obj = what_are_the_vars(None)
doomprinter(obj)
```
You should get as output:
```
var_0: None
end
```
- With 2 None as parameters:
```
obj = what_are_the_vars(None, None)
doomprinter(obj)
```
You should get as output:
```
var_0: None
var_1: None
end
```
- With a function as argument and a function as keyword argument:
```
obj = what_are_the_vars(lambda x: x, function=what_are_the_vars)
doomprinter(obj)
```
You should get as output:
```
function:
var_0: at 0x...>
end
```

✓ Yes ✕ No

Exercise 02: the logger

The goal of the exercise is to discover the decorator in Python and work with a very current one: "@log".

Basic tests

Verify log decorator is correctly implemented:
- It must be define in the same file.
- The definition of log take only one parameter which is supposed to be a
function.
- name is obtained via the function received as parameter (function.__name__).
- username is obtained via the environment variable (os.environ("USER")).

Verify the logger.py have the same output as the one expected (see the subject):
()Running: [exec-time =]

Feel free to perform additional tests (by adding or removing call to
methods of CoffeMachine instance int the main part of the script).

✓ Yes ✕ No

Exercise 03: Json issues

Implementation

Verify the different methods are implemented:
- __init__(self, filename=None, sep=',', header=False, skip_top=0, skip_bottom=0)
- __enter__(self)
- __exit__(self, type, value, traceback)
- getdata(self)
- getheader(self)

✓ Yes ✕ No

Errors management

Perform the following tests. They should be handled properly:
- the file given does not exists: should return None.
- the file has inconsistent number of fields
(number of columns in header != number of columns in data).
- All lines do not have the same number of columns.

✓ Yes ✗ No

Basic tests

Perform the following tests. They should work and give correct results:
- CSV files with and without header (test the header parameter).
- CSV files with different separator (sep = ',' ';' ':' ...).
- when the skips are set, the getdata collects the corresponding lines.
- getdata allows us to retrieve the csv content.
- getheader allows us to retrieve the csv header
(being a list of str or None depending of the boolean value when instanced).

✓ Yes ✗ No

Exercise 04 - MiniPack

The goal of the exercise is to learn how to build a package and a distribution in Python.

turn-in files

A python package has to be constituted of the following files according to
the file structure:

```
ex02/
├── LICENSE.txt (mandatory for the exercise, recommended otherwise)
├── pyproject.toml (recommended)
├── README.md (mandatory for the exercise, recommended otherwise)
├── setup.cfg (mandatory)
├── setup.py (mandatory)
├── my_minipack/
│   ├──── __init__.py
│   ├──── logger.py
│   └──── progressbar.py
└──── tests/ (not asked here, recommended otherwise)
```

First, check the different files in the directory. License, README,
setup.py and setup.cfg files must be present and a directory containing the sources.
In the sources directory you should have 3 python files:
- __init__.py
- logger.py
- progressbar.py

✓ Yes ✗ No

build script

Run the command 'bash build.sh' .
- Does the packages wheel, pip and setuptools are upgraded (`pip list`)?
- Is the package my_minipack is installed ? You should observed the command line
`python -m setup.py sdist bdist_wheel` in the build.sh script.

Check the metadata with `pip show -v my_minipack`. You should observed:
- name and version of the package
- author information (name and email)
- License and summary
- Intended audience and Topic items in the classifiers

✓ Yes ✗ No

Ratings

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

✔ Ok

Empty work Norme Cheat Crash Incomplete group Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation