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

You should evaluate 1 student in this team

★

Git repository

`sphere-v2.42.fr:vogsphere/intra-uuid-510c4673-ef5a-42e8-b5a`

Comments

This second module is the evolution of two original projects created by the
Paris-based student association 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 opposition/refusal to this idea)

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 objects, classes, inheritance, built-in functions, magic methods, generator ...

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/34188/en.subject.pdf)

banking_test2.py (/uploads/document/document/5823/banking_test2.py)

banking_test1.py (/uploads/document/document/5824/banking_test1.py)

Exercise 00 - The Book

The goal of the exercise is to get you familiar with the notions of classes and the manipulation of the objects related to those classes.

Error management and basic tests

There are 2 classes, Book in book.py and Recipe in recipe.py.
A Book object stores and manages Recipe.
With the help of the test.py file, carry out at least the following tests
to try to stress the error management:
- Recipe:
Initialize a Recipe object with different combinaison of parameters

to check the implementation, a combinaison different that those possible in
the subject, should, as specified, print a detailed error message and exit
properly.
- Book:
- Verify also for Book instance the validity of the initialization with
valid and incorrect parameters
- After the initialization of a correct Book object, verify:
- creation_date and last_update have the same value
- calling add_recipe() try the following bad arguments.
Each of them must declare an error and exit properly.
- wrong type (like an int) for the 'name'
- wrong type (like a string) for 'cooking_lvl'
- invalid value like 0 for 'cooking_lvl'
- wrong type (like a string) for 'cooking_time'
- invalid value like -10 for 'cooking_time'
- Add a new correct recipe and verify that 'last_update' has changed
- get_recipe_by_name()
- Give an existing recipe as parameter, the function must printed it and
returned an instance of it
- Give a non existing recipe as parameter, the function must declare
an error and handle it correctly. Verify also that 'last_update'
did not change
- get_recipe_by_type()
- With an correct type (with at least a recipe), the function must
print all the recipes which match the type.
- With a correct type which exists but being empty. This case must be handled properly
- Verify also 'last_update' did not change
- With a non existing recipe type, the function must declare an
error and handle it correctly.

✓ Yes ✕ No

Exercise 01 - Family Tree

The goal of the exercise is to tackle the notion inheritance of class.

notion of class

The exercise shows an example of inheritance of a class from a parent class.

- Verify the implementation of the differents classes (GotCharacter and
the children), parent should contains the attributes 'first_name' and
'is_alive' while the children classes must have 'family_name',
'house_words' and the 2 methods 'print_house_words' and 'die'

- Verify when creating an instance of the GotCharacter class without any
value that you get an object with the attributes 'first_name' sets to None

and 'is_alive' sets to True.
- Perform the same test and specify a value for the first_name and and
is_alive, for example:
`luigi = GotCharacter("Name", True)`
Verify that the GotCharacter instance is alive and it's name.
- Check the presence of the docstring
`print(luigi.__doc__)`
- Verify that the GotCharacter instance has no attributes 'house_words'
or 'family_name' (of course you should not be able to call print_house_words()
or the die() neither)

- children class
- Creates an instance of the children class and checks:
`arya = Stark("Arya")`
- Check the presence of the docstring
`print(arya.__doc__)`
- Verify all the attributes of the character
- Verify that the methods associated to the character object
- Use the method die() and verify the value of is_alive
- Create a new instance with a different name, and verify it is alive
and it name

✓ Yes ✗ No

Exercise 02 - The Vector

The goal of the exercise is to get you used with built-in methods, more particularly with those allowing to perform operations. Student is expected to code built-in methods for vector-vector and vector-scalar operations as rigorously as possible. Concerning the __str__ and __repr__ you have the opportunity to observe their behaviors through the tests.

Implementation of built-in methods

First, verifiy all the expected built-in methods are implemented, and
hen printing a vector you observe an output similar to:
'(Vector [n1, ..., n2])'

You can note that clever implementation may used __add__ method within
radd, __sub__ and __rsub__. Nethertheless it is perfectly fine if it is
not the case.

✓ Yes ✗ No

built-in method __init__

Verify the correct implementation of the __init__ method:
Create multiple instances of Vector class according to the different
possibilities and verify if the values and shape are correct:
- a list of floats : [1. , 2e-3, 3.14, 5.]
- a list of floats with a bad type within : [1. , 2e-3, None, 5.]
This is an error and should be properly handled by displaying an
error message.
- only a positive non zero integer N.
Vector made of N elements must be created with default values starting from 0.0: [[0.0], [1.0], ...]
- an integer of value zero, and a negative integer
This is an error and should be properly handled by displaying an
error message.
- a positive float:
This is an error and should be properly handled by displaying an
error message.
- a tuple of two integers, (a, b), with a < b:
vector of (b - a) elements should be created with values
laying between a and b included.
- a tuple of two integers (a, b), with a > b:
This is an error and should be properly handled by displaying an
error message.
- a tuple of two integers (a, a):
Should return None.
- a tuple of one integer and one float (4, 2.) and (4, 2.1):
This is an error and should be properly handled by displaying an
error message.

✓ Yes ✗ No

built-in method __mul__ and __rmul__

Verify the correct implementation of the built-in method __mul__ and
__rmul__ by performing several tests related to '*' operator:
For each test verify that the result of the operation is a vector object or a scalar,
and if it's value is correct.
- 'Vector(m) * Vector(m)'
The result correspond to the dot product of the 2 vectors.
- 'Vector(m) * Vector(n)', with m != n
An error should be displayed, and properly handled.
- 'Vector(m) * 2'
- '2 * Vector(m)'
- 'Vector(m) * None' and 'Vector(m) * "hello"'
An error should be displayed, and properly handled.
- 'None * Vector(n)' and '"world" * Vector(m)'
An error should be displayed, and properly handled.
- 'Vector(1) * Vector(1)'
An error should be displayed, and properly handled (not defined for vector - vector)

✓ Yes ✗ No

built-in method __add__, __radd__, __sub__ and __rsub__

Verify the correct implementation of the built-in method __add__,
__radd__, __sub__ and __rsub__ by performing several tests related
to '+' and '-' operators:
For each test verify that the result of the operation is a vector object or a scalar,
and if it's value is correct.
- 'Vector(m) + Vector(m)'
- 'Vector(1) + Vector(1)'
- 'Vector(m) + Vector(n)', with m != n
An error should be displayed, and properly handled.
- 'Vector(m) + 2' and '2 + Vector(m)'
An error should be displayed, and properly handled
- 'Vector(m) + None' and 'Vector(m) + "hello"'
An error should be displayed, and properly handled
- 'None + Vector(n)' and '"world" + Vector(m)'
An error should be displayed, and properly handled

Perform the same kind of tests (at least the same tests) for __sub__
and __rsub__.

✓ Yes ✗ No

built-in method __div__ and __rdiv__

- For division perform the following tests:
- 'Vector(m) / 2'
- 'Vector(m) / 3.14'
- 'Vector(m) / 0'
An error should be displayed, and properly handled.
- 'Vector(m) / Vector(m)' and 'Vector(m) / Vector(n)', n and m != 0 and n != m
An error should be displayed, and properly handled.
- 'Vector(m) / None' and 'None / Vector(m)'
An error should be displayed, and properly handled
- '2 / Vector((n, m))' and '2 / Vector((0, n))', n and m != 0 and n != m
An error should be displayed, and properly handled.

✓ Yes ✗ No

Exercise 03 - Generator!

The goal of the exercise is to discover the concept of generator object in Python, throught the implementation of a function 'generator' which performs a splitting operation and an ordering class operation.

Error managament and basic tests

- First, check the implementation of the function (all behaviors are implemented: 'shuffle', 'ordered' and 'unique').

- Verify the following cases are correctly managed and the message "ERROR" is printed:
- Parameter 'text' is None
- Parameter 'text' has a type different than 'str'
- Parameter 'option' has a type different than 'str'
- Parameter 'option' is specified and is a string different than 'unique', 'ordered' and 'shuffle'

- Secondly, with 'text' parameter being a string with multiple words and punctuation (for example text="This is a simple string for a basic test. Very simple."), runs the tests:
```
for elem in my_generator:
print(elem)
```
With my_generator being:
- `generator(txt, sep=' ')`
- `generator(txt, sep='.')`
- `generator(txt, sep='i')`
- `generator(txt, sep='si')`

For the given example the tests give (with line break between each strings):
- First test: 'This' 'is' 'a' 'simple' 'string' 'for' 'a' 'basic' 'test.' 'Very' 'simple.'
- Second test: 'This is a simple string for a basic test' ' Very simple' ''
- Third test: 'Th' 's ' 's a s' 'mple str' 'ng for a bas' 'c test. Very s' 'mple.'
- Fourth test: 'This is a ' 'mple strung for a ba' 'c test. Very ' 'mple.'

- Finally, with 'text' parameter being a non empty string you must test 'option' parameter. Check the value 'ordered' gives a alphanumerical ordering, 'unique' gives a set of the words (no identical strings) and 'shuffle' randomly arrange the words. Spend a particular attention on the last value, 2 runs with 'shuffle' should not give the same arrangement (th opposite is highly unlikely). Verify also there is no missing words.

Yes No

Exercise 04 - Working with lists

The goal of the exercise is to discover 2 useful methods for lists, tuples, dictionnaries (iterable class objects more generally) named zip and enumerate.

Error managament and basic tests

- First, check that the implementation of the class and methods.
You must find 'zip' in the method 'zip_evaluate' and 'enumerate'
in 'enumerate_evaluate' and no while loop. It is a normal implementation
to find 'enumerate(zip(coefs, words))' so do not be surpised.
- Verify the following cases are correctly managed and -1 is returned:
- Both 'words' and 'coefs' lists are None
- Both 'words' and 'coefs' lists are empty
- Either 'words' or 'coefs' is None/empty list and the other is a well
formatted list (list of strings for 'words', integers or floats for
'coefs')
- 'words' or 'coefs' does not correspond to the rigth format, for
example 'words' contains an integer as one of its elements

- Then, test several cases with lists of variable length to check the
validity of the implementation and check no term of lists are missing
in the sum of products (especially the last one).

✓ Yes ✕ No

Exercise 05 - Bank account

The goals of this exercise is to teach you new built-in functions and get a deeper understanding about class manipulation and to be aware of possibility to modify instanced objects. In this exercise you learn how to modify or add attributes to an object.

Error managament and basic tests

- First, you have to check the classes implementation, meaning that you
have to check the presence of:
- the class Account and Bank in the file bank.py
- the methods' __init__', 'transfer' and the attribute 'ID_COUNT' in
the class Account.
- the methods '__init__', 'add', 'transfer' and 'fix_account' in the
class Bank.
Student may have implement extra methods to manage the security
aspect, thus it is okay to have more than the methods mentionned.

- Second, you have to perform the following test to verify the security
management:
- `bank1.add(account1)`
where 'bank1' is an instance of Bank and:
- 'account1' is not an instance of the class Account().
- 'account1' is a instance of Account() but there is already an account
with the same name attribute in 'bank1'.
- `bank1.transfer(arg1, arg2, arg3)`
with a several combinaison of wrong parameters:
- arg1 and/or arg2 are not strings.

- amount is not an int or a float.
- `bank1.transfer(acc1.name, acc2.name, amount)`
where 'bank1' is an instance of Bank, 'acc1' and 'acc2' 2 instances
of Account. Try the 2 cases:
- 'acc1' and 'acc2' are valid and there is enough money on 'acc1'
to perform the operation (success case).
- 'acc1' and 'acc2' are valid but there is not enough money on
'acc1' to perform the operation (failed case).
- One of the accounts 'acc1' and 'acc2' is not valid (failed case).
- 'acc1' and 'acc2' are valid and there is enough money on 'acc1'
to perform the operation but 'acc2' is not in accounts attribute of `bank1` (failed case).
- `bank1.fix_account(name)`:
- `name` is not the attribute of an account in `bank1.accounts` (return False).
- `name` is not a string (return False).
- `name` is the attribute of an account in `bank1.accounts` (return True).

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

Privacy policy (https://signin.intra.42.fr/legal/terms/5)
Terms of use for video surveillance (https://signin.intra.42.fr/legal/terms/1)
Declaration on the use of cookies (https://signin.intra.42.fr/legal/terms/2)
General term of use of the site (https://signin.intra.42.fr/legal/terms/6)
Legal notices (https://signin.intra.42.fr/legal/terms/3)
Règlement Intérieur (https://signin.intra.42.fr/legal/terms/7)