Python & ML - Module 02

Basics 3

Summary: Let's continue practicing with more advanced Python programming exercises. Destination: Decorators, lambda, context manager and packaging.

Chapter I

Common Instructions

- The version of Python recommended to use is 3.7, you can check the version of Python with the following command: `python -V`
- The norm: during this piscine, it is recommended to follow the PEP 8 standards, though it is not mandatory. You can install pycodestyle which is a tool to check your Python code.
- The function `eval` is never allowed.
- The exercises are ordered from the easiest to the hardest.
- Your exercises are going to be evaluated by someone else, so make sure that your variable names and function names are appropriate and civil.
- Your manual is the internet.
- If you are a student from 42, you can access our Discord server on 42 student's associations portal and ask your questions to your peers in the dedicated Bootcamp channel.
- You can also ask questions in the `#bootcamps` channel on Slack at 42AI or 42born2code.
- If you find any issue or mistakes in the subject please create an issue on 42AI repository on Github.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be run after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

Contents

Chapter II

Exercise 00

	Exercise : 00
Map, filter, reduce	
Turn-in directory : $ex00/$	
Files to turn in : `ft_map.py, ft_filter.py, ft_reduce.py`	
Forbidden functions : `map, filter, reduce`	

Objective

The goal of this exercise is to work on the built-in functions `map`, `filter` and `reduce`.

Instructions

Implement the functions `ft_map`, `ft_filter` and `ft_reduce`.

Take the time to understand the use cases of these two built-in functions (`map` and `filter`) and the function `reduce` in the functools module.

You are not expected to code specific classes to create `ft_map`, `ft_filter` or `ft_reduce` objects, take a closer look at the examples to know what to do.

Here are the signatures of the functions:

```python
def ft_map(function_to_apply, iterable):
        """Map the function to all elements of the iterable.
        Args:
          function_to_apply: a function taking an iterable.
          iterable: an iterable object (list, tuple, iterator).
        Return:
          An iterable.
          None if the iterable can not be used by the function.
        """
  # ... Your code here ...

def ft_filter(function_to_apply, iterable):
        """Filter the result of function apply to all elements of the iterable.
        Args:
          function_to_apply: a function taking an iterable.
          iterable: an iterable object (list, tuple, iterator).
        Return:
          An iterable.
          None if the iterable can not be used by the function.
        """
  # ... Your code here ...

def ft_reduce(function_to_apply, iterable):
        """Apply function of two arguments cumulatively.
        Args:
          function_to_apply: a function taking an iterable.
          iterable: an iterable object (list, tuple, iterator).
        Return:
          A value, of same type of elements in the iterable parameter.
          None if the iterable can not be used by the function.
        """
  # ... Your code here ...
```

Examples

```python
# Example 1:
x = [1, 2, 3, 4, 5]
ft_map(lambda dum: dum + 1, x)
# Output:
<generator object ft_map at 0x7f708faab7b0> # The adress will be different

list(ft_map(lambda t: t + 1, x))
# Output:
[2, 3, 4, 5, 6]

# Example 2:
ft_filter(lambda dum: not (dum % 2), x)
# Output:
<generator object ft_filter at 0x7f709c777d00> # The adress will be different

list(ft_filter(lambda dum: not (dum % 2), x))
# Output:
[2, 4]

# Example 3:
lst = ['H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
ft_reduce(lambda u, v: u + v, lst)
# Output:
"Hello world"
```

You are expected to produce the raise of exception for the functions similar to exceptions of `map`, `filter` and `reduce` when wrong parameters are given (but no need to reproduce the exact same exception messages).

Chapter III

Exercise 01

	Exercise : 01
	args and kwargs?
Turn-in directory : $ex01/$	
Files to turn in : `main.py`	
Forbidden functions : `None`	

Objective

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

Instructions

In this exercise you have to implement a function named `what_are_the_vars` which returns an instance of class ObjectC.

ObjectC attributes are set via the parameters received during the instanciation. You will have to modify the 'instance' `ObjectC`, **NOT** the class.

You should take a look at `getattr`, `setattr` built-in functions.

```python
def what_are_the_vars(...):
        """
        ...
        """
        # ... Your code here ...

class ObjectC(object):
        def __init__(self):
                # ... Your code here ...

def doom_printer(obj):
        if obj is None:
                print("ERROR")
                print("end")
                return
        for attr in dir(obj):
                if attr[0] != '_':
                        value = getattr(obj, attr)
                        print("{}: {}".format(attr, value))
        print("end")

if __name__ == "__main__":
        obj = what_are_the_vars(7)
        doom_printer(obj)
  obj = what_are_the_vars(None, [])
        doom_printer(obj)
        obj = what_are_the_vars("ft_lol", "Hi")
        doom_printer(obj)
        obj = what_are_the_vars()
        doom_printer(obj)
        obj = what_are_the_vars(12, "Yes", [0, 0, 0], a=10, hello="world")
        doom_printer(obj)
        obj = what_are_the_vars(42, a=10, var_0="world")
        doom_printer(obj)
  obj = what_are_the_vars(42, "Yes", a=10, var_2="world")
        doom_printer(obj)
```

Examples

```
$> python main.py
var_0: 7
end
var_0: None
var_1: []
end
var_0: ft_lol
var_1: Hi
end
end
a: 10
hello: world
var_0: 12
var_1: Yes
var_2: [0, 0, 0]
end
ERROR
end
a: 10
var_0: 42
var_1: Yes
var_2: world
end
```

Chapter IV

Exercise 02

	Exercise : 02
The logger	
Turn-in directory : $ex02/$	
Files to turn in : `logger.py`	
Forbidden functions : `None`	

Objective

In this exercise, you will learn about decorators and we are not talking about the decoration of your room.

The `@log` will write info about the decorated function in a `machine.log` file.

Instructions

You have to create the log decorator in the same file.

Pay attention to all the different actions logged at the call of each method. You may notice the username from environment variables is written to the log file.

```python
import time
from random import randint
import os

#... your definition of log decorator...

class CoffeeMachine():

        water_level = 100

        @log
        def start_machine(self):
          if self.water_level > 20:
                  return True
          else:
                  print("Please add water!")
                  return False

        @log
        def boil_water(self):
                return "boiling..."

        @log
        def make_coffee(self):
                if self.start_machine():
                        for _ in range(20):
                                time.sleep(0.1)
                                self.water_level -= 1
                        print(self.boil_water())
                        print("Coffee is ready!")

        @log
        def add_water(self, water_level):
                time.sleep(randint(1, 5))
                self.water_level += water_level
                print("Blub blub blub...")


if __name__ == "__main__":

        machine = CoffeeMachine()
        for i in range(0, 5):
                machine.make_coffee()

        machine.make_coffee()
        machine.add_water(70)
```

Examples

```
$> python logger.py
boiling...
Coffee is ready!
boiling...
Coffee is ready!
boiling...
Coffee is ready!
boiling...
Coffee is ready!
Please add water!
Please add water!
Blub blub blub...
$>
```

```
$> cat machine.log
(cmaxime)Running: Start Machine   [ exec-time = 0.001 ms ]
(cmaxime)Running: Boil Water      [ exec-time = 0.005 ms ]
(cmaxime)Running: Make Coffee     [ exec-time = 2.499 s ]
(cmaxime)Running: Start Machine   [ exec-time = 0.002 ms ]
```

```
(cmaxime)Running: Boil Water     [ exec-time = 0.005 ms ]
(cmaxime)Running: Make Coffee    [ exec-time = 2.618 s ]
(cmaxime)Running: Start Machine  [ exec-time = 0.003 ms ]
(cmaxime)Running: Boil Water     [ exec-time = 0.004 ms ]
(cmaxime)Running: Make Coffee    [ exec-time = 2.676 s ]
(cmaxime)Running: Start Machine  [ exec-time = 0.003 ms ]
(cmaxime)Running: Boil Water     [ exec-time = 0.004 ms ]
(cmaxime)Running: Make Coffee    [ exec-time = 2.648 s ]
(cmaxime)Running: Start Machine  [ exec-time = 0.011 ms ]
(cmaxime)Running: Make Coffee    [ exec-time = 0.029 ms ]
(cmaxime)Running: Start Machine  [ exec-time = 0.009 ms ]
(cmaxime)Running: Make Coffee    [ exec-time = 0.024 ms ]
(cmaxime)Running: Add Water      [ exec-time = 5.026 s ]
$>
```

Pay attention, the length between ":" and "[" is 20]. Draw the corresponding conclusions on this part of a log entry.

Chapter V

Exercise 03

	Exercise : 03
Json issues	
Turn-in directory : $ex03/$	
Files to turn in : `csvreader.py`	
Forbidden functions : `None`	

Objective

The goal of this exercise is to implement a context manager as a class.

Thus you are strongly encouraged to do some preliminary research about context manager.

Instructions

Implement a `CsvReader` class that opens, reads, and parses a CSV file.

This class is then a context manager as a class.

In order to create it, your class requires a few built-in methods:

- `__init__`,
- `__enter__`,
- `__exit__`.

It is mandatory to close the file once the process is completed. You are expected to handle properly badly formatted CSV file (i.e. handle the exception):

- mistmatch between number of fields and number of records,
- records with different lengths.

```
class CsvReader():
	def __init__(self, filename=None, sep=',', header=False, skip_top=0, skip_bottom=0):
		# ... Your code here ...

	def __enter__(...):
		# ... Your code here ...

	def __exit__(...):
		# ... Your code here ...

	def getdata(self):
	""" Retrieves the data/records from skip_top to skip bottom.
	  Returns:
		nested list (list(list, list, ...)) representing the data.
	"""
		# ... Your code here ...

	def getheader(self):
	""" Retrieves the header from the csv file.
	Returns:
		list: representing the data (when self.header is True).
	  None: (when self.header is False).
	"""
		# ... Your code here ...
```

`CSV` (for Comma-Separated Values) files are delimited text files which use a given character to separate values.

The separator (or delimiter) is usually a comma (`,`) or an hyphen comma (`;`), but with your context manager you have to offer the possibility to change this parameter.

One can decide if the class instance skips lines at the top and the bottom of the file via the parameters `skip_top` and `skip_bottom`.

One should also be able to keep the first line as a header if `header` is `True`.

The file should not be corrupted (either a line with too many values or a line with too few values), otherwise return `None`.

You have to handle the case `file not found`.

You are expected to implement two methods:

- `getdata()`,
- `getheader()`.

```
from csvreader import CsvReader

if __name__ == "__main__":
	with CsvReader('good.csv') as file:
		data = file.getdata()
		header = file.getheader()
```

```
from csvreader import CsvReader

if __name__ == "__main__":
	with CsvReader('bad.csv') as file:
		if file == None:
			print("File is corrupted")
```

Chapter VI

Exercise 04

	Exercise : 04
MiniPack	
Turn-in directory : *ex*04/	
Files to turn in : `build.sh, *.py, *.md, *.cfg, *.txt`	
Forbidden functions : `None`	

Objective

The goal of this exercise is to learn how to build a package and understand the magnificence of PyPi.

Instructions

You have to create a package called `my_minipack`.

RTFM

It will have 2 **modules**:

- the progress bar (module00 ex10) which should be imported it via `import my_minipack.progress`
- the logger (module02 ex02), which should be imported via `import my_minipack.logger`.

The package will be installed via pip using one of the following commands (both should work):

```
$> pip install ./dist/my_minipack-1.0.0.tar.gz
$> pip install ./dist/my_minipack-1.0.0-py3-none-any.whl
```

Based on the following terminal commands and corresponding outputs, draw the necessary conclusion.

```
$> python -m venv tmp_env && source tmp_env/bin/activate
(tmp_env) > pip list
# Ouput
Package   Version
---------- -------
pip       19.0.3
setuptools 40.8.0

(tmp_env) $> cd ex04/ && bash build.sh
# Output ... No specific verbose expected, do as you wish ...
...
(tmp_env) $> ls dist
# Output
my_minipack-1.0.0-py3-none-any.whl my_minipack-1.0.0.tar.gz

(tmp_env) $> pip list
# Output
Package    Version
---------- -------
my-minipack 1.0.0
pip        21.0.1 # the last version at the time
setuptools 54.2.0 # the last version at the time
wheel      0.36.2 # the last version at the time

(tmp_env) $> pip show -v my_minipack
# Ouput (minimum metadata asked)
Name: my-minipack
Version: 1.0.0
Summary: Howto create a package in python.
Home-page: None
Author: mdavid
Author-email: mdavid@student.42.fr
License: GPLv3
Location: [PATH TO BOOTCAMP PYTHON]/module02/tmp_env/lib/python3.7/site-packages
Requires:
Required-by:
Metadata-Version: 2.1
Installer: pip
Classifiers:
Development Status :: 3 - Alpha
Intended Audience :: Developers
Intended Audience :: Students
Topic :: Education
Topic :: HowTo
Topic :: Package
License :: OSI Approved :: GNU General Public License v3 (GPLv3)
Programming Language :: Python :: 3
Programming Language :: Python :: 3 :: Only
(tmp_env) $>
```

Also add a LICENSE.md (you can choose a real license or a fake one it does not matter) and a README file where you will write a small documentation about your packaged library.

The ‘build.sh‘ script upgrades ‘pip‘, and **builds** the distribution packages in ‘wheel‘ and ‘egg‘ formats.

You can check whether the package was properly installed by running the command pip list that displays the list of installed packages and check the metadata of the package with pip show -v my_minipack. Of course do not reproduce the exact same metadata, change the author information, modify the summary Topic and Audience items if you want to.

Chapter VII

Exercise 05

	Exercise : 05
TinyStatistician	
Turn-in directory : $ex05/$	
Files to turn in : `TinyStatistician.py`	
Forbidden functions : `Any function that calculates mean, median, quartiles, variance or standar deviation for you.`	

Objective

Initiation to very basic statistic notions.

Instructions

Create a class named `TinyStatistician` that implements the following methods:

- `mean(x)`: computes the mean of a given non-empty list or array `x`, using a for-loop. The method returns the mean as a float, otherwise `None` if `x` is an empty list or array. Given a vector `x` of dimension $m \times 1$, the mathematical formula of its mean is:
$$\mu = \frac{\sum_{i=1}^{m} x_i}{m}$$
- `median(x)`: computes the median of a given non-empty list or array `x`. The method returns the median as a float, otherwise `None` if `x` is an empty list or array.
- `quartiles(x)`: computes the 1^{st} and 3^{rd} quartiles of a given non-empty array `x`. The method returns the quartile as a float, otherwise `None` if `x` is an empty list or array.
- `var(x)`: computes the variance of a given non-empty list or array `x`, using a for-loop. The method returns the variance as a float, otherwise `None` if `x` is an empty

list or array. Given a vector `x` of dimension $m \times 1$, the mathematical formula of its variance is:

$$\sigma^2 = \frac{\sum_{i=1}^{m}(x_i - \mu)^2}{m} = \frac{\sum_{i=1}^{m}[x_i - (\frac{1}{m}\sum_{j=1}^{m} x_j)]^2}{m}$$

- `std(x)` : computes the standard deviation of a given non-empty list or array `x`, using a for-loop. The method returns the standard deviation as a float, otherwise `None` if `x` is an empty list or array. Given a vector `x` of dimension $m \times 1$, the mathematical formula of its standard deviation is:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{m}(x_i - \mu)^2}{m}} = \sqrt{\frac{\sum_{i=1}^{m}[x_i - (\frac{1}{m}\sum_{j=1}^{m} x_j)]^2}{m}}$$

All methods take a `list` or a `numpy.ndarray` as parameter.

We are assuming that all inputs have a correct format, i.e. a list or array of numeric type or empty list or array.

You don't have to protect your functions against input errors.

Examples

```
from TinyStatistician import TinyStatistician
tstat = TinyStatistician()
a = [1, 42, 300, 10, 59]

tstat.mean(a)
# Expected result: 82.4

tstat.median(a)
# Expected result: 42.0

tstat.quartile(a)
# Expected result: [10.0, 59.0]

tstat.var(a)
# Expected result: 12279.439999999999

tstat.std(a)
# Expected result: 110.81263465868862
```

Contact

You can contact 42AI association by email: contact@42ai.fr

If you are a student from 42, you can access our Discord server on 42 student's associations portal and ask your questions to your peers in the dedicated Bootcamp channel.

Find all the relevant and up-to-date information about 42AI on our Website !

Thank you for attending this Python Bootcamp module02 !

Acknowledgements

The modules Python & ML is the result of a collective work, we would like to thank:

- Maxime Choulika (cmaxime),
- Pierre Peigné (ppeigne, pierre@42ai.fr),
- Matthieu David (mdavid, matthieu@42ai.fr),
- Quentin Feuillade–Montixi (qfeuilla, quentin@42ai.fr)
- Mathieu Perez (maperez, mathieu.perez@42ai.fr)

who supervised the creation, the enhancement and this present transcription.

- Louis Develle (ldevelle, louis@42ai.fr)
- Augustin Lopez (aulopez)
- Luc Lenotre (llenotre)
- Owen Roberts (oroberts)
- Thomas Flahault (thflahau)
- Amric Trudel (amric@42ai.fr)
- Baptiste Lefeuvre (blefeuvr@student.42.fr)
- Mathilde Boivin (mboivin@student.42.fr)
- Tristan Duquesne (tduquesn@student.42.fr)

for your investment for the creation and development of these modules.

- Barthélémy Leveque (bleveque@student.42.fr)
- Remy Oster (roster@student.42.fr)
- Quentin Bragard (qbragard@student.42.fr)
- Marie Dufourq (madufour@student.42.fr)

- Adrien Vardon (advardon@student.42.fr)

who betatest the first version of the modules of Machine Learning.