Bootcamp Python

Module02 Basics 3

Module02 - Basics 3

Let's continue practicing with more advanced Python programming exercises.

Notions of the module

Decorators, lambda, context manager and build package.

General rules

- 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 bootcamp you will follow the PEP 8 standards. 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.
- You can also ask questions in the `#bootcamps` channel in the 42 AI Slack: 42-ai.slack.com.
- If you find any issue or mistakes in the subject please create an issue on our bootcamp python repository on Github.

Helper

Ensure that you have the right Python version.

```
> which python
/goinfre/miniconda/bin/python
> python -V
Python 3.7.*
> which pip
/goinfre/miniconda/bin/pip
```


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
Remarks:	n/a

Objective:

THe goal of the 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 functools module. You are not expected to code specific classes to create `ft_map`, `ft_filter` or `ft_reduce` objects, take a look to the examples section to know what to do.

Here the signatures of the methods:

```
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.
    """

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.
    """

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.
    """
```

Examples:

```
# 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).

Exercise 01 - args and kwargs?

Turn-in directory:	ex01/
Files to turn in:	main.py
Forbidden functions:	None
Remarks:	n/a

Objective:

The goal of the 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 to `getattr`, `setattr` built-in functions.

```
def what_are_the_vars(...):
    """
    ...
    """
    ... Your code ...

class ObjectC(object):
    def __init__(self):
        pass

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("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, hello="world")
    doom_printer(obj)
```

Examples:

```
$> python main.py
var_0: 7
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
```

Exercise 02 - The logger

Turn-in directory:	ex02/
Files to turn in:	logger.py
Forbidden functions:	None
Remarks:	n/a

Objective:

In this exercise, you will discover the notion of 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 methods. You may notice the username from environment variable is written to the log file.

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

```
... 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 "[". Draw the corresponding conclusions on this part of a log entry.

Exercise 03 - Json issues

Turn-in directory:	ex03/
Files to turn in:	csvreader.py
Forbidden functions:	None
Remarks:	Context Manager

Objective:

The goal of this exercise is to implement a context manager as a class. Thus you are strongly encouraged to do some 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 class. In order to create it, your class requires few built-in methods:

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

It is mandatory to close the file once the process has 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 length.

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

    def __enter__(...):
        ... Your code ...
```

```
    def __exit__(...):
        ... Your code ...

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

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

CSV (for Comma-Separated Values) file is a delimited text file which uses a comma to separate values. Therefore, the field separator (or delimiter) is usually a comma (`,`) but with your context manager you have to offer the possibility to change this parameter.

You can make the class skip lines at the top and the bottom of the file, and also 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 also have to handle the case `file not found`. You have 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")
```

Exercise 04 - MiniPack

Turn-in directory:	ex04/
Files to turn in:	build.sh, *.py, *.md, *.cfg, *.txt
Forbidden functions:	None
Remarks:	n/a

Objective:

The goal of the exercise is to learn how to build a package and understang the magnificience of PyPi.

Instruction:

You have to create a package called `my_minipack`.

Hint: RTFM

It will have 2 modules:

- the progress bar (module00 ex10), that can be imported via `import my_minipack.progressbar`,
- the logger (module02 ex02) `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 env 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 but make it simple ...#
(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: How-to 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 :: How-To
Topic :: Package
License :: OSI Approved :: GNU General Public License v3 (GPLv3)
Programming Language :: Python :: 3
Programming Language :: Python :: 3 :: Only
(tmp_env) $>
```

The `build.sh` script upgrades `pip`, `wheel` and `setuptools` packages and creates the `my_miniapck-1.0.0.tar.gz` and the `my_miniapck-1.0.0-py3-none-any.whl` files in the `dist/` repository.

Info:

You can ensure 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 wish to.

Exercise 05 - TinyStatistician

Turn-in directory:	ex05/
Files to turn in:	TinyStatistician.py
Forbidden functions:	Any function that calculates mean, median, quartiles, variance or standard deviation for you
Forbidden libraries:	Numpy
Remarks:	n/a

Objective:

Initiation to very basic statistic notions.

Instructions:

Create a class named `TinyStatistician` that implements the following methods.
All methods take a `list` or a numpy.`ndarray` as parameter. We are assuming that all inputs are correct, i.e. you don't have to protect your functions against input errors.

- `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 array.

Given a vector x of dimension m * 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 array.
- `quartiles(x)` : computes the 1st and 3rd quartiles of a given non-empty array `x`. The method returns the quartile as a float, otherwise `None` if `x` is an empty 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 array.

Given a vector x of dimension m * 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 array.

Given a vector x of dimension m * 1, the mathematical formula of its standard deviation is:

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

Examples

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

>>> tstat.mean(a)
82,4

>>> tstat.median(a)
42.0

>>> tstat.quartile(a)
[10.0, 59.0]

>>> tstat.var(a)
12279.439999999999

>>> tstat.std(a)
110.81263465868862
```