Bootcamp Python

Bootcamp Python

Day02 - Basics 3

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

Notions of the day

Decorators, multiprocessing, lambda, build package, ...

General rules

- The version of Python 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 Pep8 standards https://www.python.org/dev/peps/pep-0008/
- 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 dedicated 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 dedicated repository on Github: https://github.com/42-AI/bootcamp_python/issues.

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.

Exercise 01 - args and kwargs ?

Exercise 02 - The logger.

Exercise 03 - Json issues

Exercise 04 - Package ??

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

Implement the higher order functions **`map()`**, **`filter()`** and **`reduce()`**. Take the time to understand the use cases of these three built-in functions.

How they should be prototyped:

```
def ft_map(function_to_apply, list_of_inputs):
    pass

def ft_filter(function_to_apply, list_of_inputs):
    pass

def ft_reduce(function_to_apply, list_of_inputs):
    pass
```

Exercise 01 - args and kwargs ?

Turn-in directory :	ex01
Files to turn in :	main.py
Forbidden functions :	
Remarks :	n/a

Implement the **what_are_the_vars** function that returns an object with the right attributes.

You will have to modify the "instance" ObjectC, NOT THE CLASS.

Have a look to **getattr**, **setattr**.

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

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, var_0="world")
    doom_printer(obj)
```

output

```
>> 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 :	
Remarks :	n/a

You are going to learn some more advanced features of Python.

In this exercice, we want you to 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.

```
import time
from random import randint

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)
```

Terminal

```
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 ]
>
```

Exercise 03 - Json issues

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

It's the context manager that will help you handle this task.

Implement a **`CsvReader`** class that opens, reads, and parses a csv file.
In order to create a context manager the class will need a few built-in methods: **`__init__`**, **`__enter__`** and **`__exit__`**.
It's mandatory to close the file at the end of the procedure.

```
class CsvReader():
    def __init__(self, sep=',', header=False, skip_top=0, skip_bottom=0):
        pass
```

Usually the separator in csv files is the **`,`**, but we want to be able to change this parameter.
You can also tell the class to skip lines at the top and the bottom, and also to keep the first line as a header if **`header`** is **`True`**.

The file shouldn't be corrupted (line with too many elements), if it's corrupted return None.
You have to handle the case **`file not found`**.

You will have to also implement 2 methods: **`getdata()`** and **`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
Forbidden functions :	
Remarks :	n/a

You have to create a package called **`ai42`**.

It will have 2 functionalities:

- the progress bar (day00 ex10), that can be imported via import ai42.progressbar,
- the logger (day02 ex02) import ai42.logging.log,

You may have to rename the functions and change the architecture of the package.

The package will be installed via pip using the following command:

```
bash build.sh && pip install ./dist/ai42-1.0.0.tar.gz
```

The build.sh script has to create the **`ai42-1.0.0.tar.gz`** file.

To ensure the package was properly installed you can run the command **`pip list`**.

You should be able to see if this is the case as **`pip list`** displays the list of installed packages.