Mandatory Assignment 3

Basic Python Programming (15 points + 2 BP)

University of Oslo - IN3110/IN4110

Fall 2020

Your solutions to this mandatory assignment should be placed in the directory `assignment3` in your Github repository.

Let's get used to good practice from the beginning. Your full assignment is expected to contain

- a `README.md` containing information on how to run your scripts. It is especially important that you document how to run your tests.[1]
- Good documentation! All functions should have docstrings explaining what the function does and how. We also expect an explanation of the parameters and return value (including types). We **highly** recommend you to use a well-established docstring style such as the Google style docstrings[2]. However, you are free to choose your own docstring style - as long as the documentation is comprehensive.

3.1 `wc` (3 points)

Make a Python implementation of the standard utility `wc` which counts the words of a file. When called with a file name as command line argument, print the single line `a b c fn` where `a` is the number of lines in the file, `b` the number of words, `c` the number of characters, and `fn` the filename.

Further, extend your script so that it can be called as `wc *` to print a nice list of word counts for all files in the current directory, or `wc *.py` to print a nice list of word counts for all python scripts in the current directory.

Exactly what constitues a word is not very important. A simple approach where words are separated by space is enough.

3.2 Unit Tests for Arrays (3 points)

Let's dive into the "real world" of development. In this part you are going to work with *test driven development*. Before writing code, you write tests that

[1] Hint: Checkout the lecture.

[2] `https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html`

can confirm that your code works as expected. Might sound weird, but you gain actually a lot. Why?

1. We are forced to think about what our code actually should do, before we start coding. Planning ahead is generally a good idea[3], and test driven development forces us to do exactly that.

2. It is easier to check our code and check that changes we made by "improving it" actually doesn't break it. With tests we can confirm that the code works just as it did before (but maybe faster).

3. If you later want to publish a module or package, you are quite likely required to verify that your code actually does what you say it does.

For instance, if you want to write an addition function `plus(a, b)`, you would expect that 2 and 2 becomes 4. To check that the function actually does that, you can formalize it as a unit test:

```
def test_two_plus_two():
    assert plus(2, 2) == 4
```

A test should by convention have a name starting with `test_`, and raise an `AssertionError` if the test fails (this is what the `assert` statement does). The test should always test the specific task your function or class is performing. Furthermore, the test should always test the same task/functionality, i.e. generating something random in a test is usually a bad idea since you might end up with tests that *sometimes* pass, which makes debugging difficult.

In this part of the assignment you are going to implement a class `Arrays` in python. Yeah!

For our implementation, the goal is to define a class `Array` that can be used as follows:

```
shape = (4,)
# define my_array
my_array = Array(shape, 2, 3, 1, 0)
# __getitem__ should be implemented correctly.
assert my_array[2] == 1
# Printing the array print the array values nicely.
# For example: [2, 3, 1, 0]
print(my_array)
```

Arrays are pretty cool data structures, which represent a grid of values. These structures allow for storing a single data type - which makes them homogeneous. They are indexed by a tuple of integers. They are not only pretty neat but also the most frequently used data structure in data science. So it is actually worth it, spending some (more) time on them. Even though, you get

[3]That could be another good life-advice

to enjoy implementing the `Array` yourself, which will help you understand important concepts of how arrays function (Yeah!), we highly recommend looking into the fantastic implementation of arrays in `NumPy - numpy.array` [4].

Take a look at the outline of the `Array`-Class in `array.py` to see which methods we are going to implement. Before you start filling in the blanks (which is the next fun exercise), write some unit tests. Implement the following tests:

- Check that your print function actually returns the nice string
- Two or more tests verifying that adding to a 1d-array element-wise returns what it's supposed to
- Two or more tests verifying that substracting from a 1d-array element-wise returns what it's supposed to
- Two or more tests verifying that multiplying a 1d-array element-wise by a factor or other 1-d array returns what it's supposed to
- Two or more tests verifying that comparing two arrays (by `==`) returns what it is supposed to - which should be a boolean.
- Two or more tests verifying that comparing a 1d-array element-wise to another array through `is_equal` returns what it's supposed to - which should be a boolean array.
- Two or more tests verifying that the mean of the array is returned correctly
- Two or more tests verifying that the variance of the array is returned correctly[5]
- Two or more tests verifying that the the element returned by `min_element` is the "smallest" one in the array

It is of course close to impossible to catch everything that might go wrong with your code. However, this does not mean that you can go for the easiest tests just so you have a test. Try to cover different scenarios for the two (or more) tests of each functionality (test with different data types for example).

We recommend implementing your tests with pytest[6]. The tests should live in a separate file named `test_Array.py`. You will need to import the `Array`-Class properly in order to run your tests [7].

[4] We will cover NumPy in the lecture, but if you want to get your feet wet already you can start reading here: https://numpy.org/doc/stable/user/quickstart.html

[5] Hint: Check out the description of this function in `array.py`

[6] `http://doc.pytest.org/en/latest/getting-started.html`

[7] If you've forgotten how to - checkout the lecture slides.

3.3 Implement the Array Class (5 points)

Implement the Array class so that an Array can be initiated by `Array(shape, 1, 2, ..., n)`. Furthermore, implement all the methods that you have developed tests for in the previous task, including `__radd__`, `__rsub__` and `__rmul__`. In summary, implement the methods in the `array.py` template. **Make sure that you check that your array actually is an array, which means it is homogeneous (all elements have the same datatype).**

Note: You should not use `numpy.array` or `numpy.reshape` or Pythons own `array` class for your own class implementation.

3.4 Additional tests for 2D Arrays (1 point)

Add tests for 2-dimensional arrays. That is, arrays with shape (n, m) where n and m are integers. Make sure that you have a test for **at least** each of these methods:

- `__add__`
- `__sub__`
- `__eq__`
- `is_equal`
- `mean`
- `min_element`

3.5 Adapt your implementation to work with 2D Arrays (3 points)

Modify your Arrays implementation such that both the previous 1D tests and the new 2D tests pass. The following code should be a valid way of defining a 2D array with shape $(3, 2)$.

```
# define my_array
my_array = Array((3, 2), 8, 3, 4, 1, 6, 1)
# accessing values should work as follows
assert my_array[1][0] == 4
```

Start with modifying your class constructor `__init__` to handle both 1D and 2D.

Hint: It can be a good idea to flatten the 2D array when performing element-wise operations. Then you do not need to handle the 1D and 2D case differently everywhere.

Note: You should not use `numpy.array` or `numpy.reshape` or Pythons own `array` class for your own class implementation.

3.6 Adapt your Implementation to Work with n-dimensional Arrays - 2 Bonus Points for IN3110 & IN4110

Make all the methods implemented in `3.3 - Implement the Array Class for 1D Arrays` work with n-dimensional arrays. Make sure not to break your previous tests.

Good luck!