Mandatory Assignment 3

Basic Python Programming (15 points + 2 BP)

University of Oslo - IN3110/IN4110

Fall 2021

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. We recommend to check out the live lecture `https://bit.ly/3yZqTbO` and the lecture notes `https://bit.ly/3jUy68N`.

Furthermore, we expect 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[1]. However, you are free to choose your own docstring style - as long as the documentation is comprehensive. In this assignment, most of the documentation has already been added for you, so you just need to add, what is missing.

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

In this part of the assignment you are going to implement a class `Arrays` in python. Yeah! 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 worth spending some (more) time on them. Even though you get 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`[2].

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

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

[2] 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

```python
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 prints the array values nicely.
# For example: [2, 3, 1, 0]
print(my_array)
```

Listing 1: Usage of Array Class.

Take a look at the outline of the `Array`-Class in `array.py` to see which methods we are going to implement.

3.1 Implement the Array Class (6 points)

Implement the Array class so that an Array can be instantiated by `Array(shape, 1, 2, ..., n)`. *shape* is a tuple of integers and refers to the dimensionality of the array. Therefore, *shape* refers to the ”rows” and ”columns” of the array, e.g. *shape(rows, columns)*.

A 1D array can be defined with shape (4,), that is, 4 elements in the first (and only) dimension.

```python
a = Array((4,), 1, 2, 3, 4)
```

Implement the methods outlined in the `array.py` template.

You will need to implement the `__getitem__()` method, to index the array. An example is provided below.

```python
def __getitem__(self, item):
  """Returns value of item in array.
      Args:
        item (int): Index of value to return.
      Returns:
        value: Value of the given item.
  """
  return self._array[item]
```

Listing 3.1.1: Example getitem.

The arrays you create should be homogeneous, meaning all elements have the same datatype. You only need to consider numeric types. There are three distinct numeric types: integers, floating point numbers, and complex numbers. In addition, booleans are a subtype of integers. **For this assignment, we will only consider the types integers, floats and booleans.**

It is okay to implement the class in such a way that if a combination of integers, floats and booleans are given, all values are converted to floats, mimicking the default behavior of numpy.

For the mathematical methods, `__add__`, `__sub__`,`__mul__`, `__radd__`, `__rsub__` and `__rmul__`, you want to check if the argument is a scalar or an array with the same shape. If it is something else you can return `NotImplemented`.

_The r methods The methods `__radd__`, `__rsub__` and `__rmul__` are called to implement the arithmetic operations `__add__`, `__sub__`,`__mul__` with swapped operands. The `r methods` are only called, if the left (first) operand does not support the operation provided and the operands are of different type. For example, imagine we have an array

```python
array1 = Array((6,), 1, 2, 3, 4, 5, 6)
```

and want to evaluate `10 + array1`, where array1 is an instance of our `Array` class, which has a `__radd__()` method. Python first calls `10___add__(array1)`. Since 10 (int or float) does not support the instance of the `Array` class, `10___add__(array1)` it returns `NotImplemented`. The r-method `array1___radd__(10)` is then called.

```python
array1 = Array((6,), 1, 2, 3, 4, 5, 6)
i = 10
i.__add__(array1) # Returns NotImplemented

array1.__radd__(i) # Returns Array((6,), 11, 12, 13, 14, 15, 16)
```

Listing 3.1.2: radd example.

Another explanation can be found here[3].

Do not use `NotImplemented` for other normal methods (such as `min_element`). For `min_element`, we don't require error handling, but you can raise a `ValueError` or `TypeError`.

Read the outline we provided of the array class carefully, as we provided additional information on the different methods to be implemented.

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

3.2 Unit Tests for Arrays (4 points)

Let's dive into the "real world" of development. If you later want to publish a module or package, you are 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:

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

Listing 3.2.1: Test Example.

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.

Implement the following tests:

[3] https://stackoverflow.com/questions/9126766/addition-between-classes-using-radd-method/38196153

- Check that your print function returns the nice string
- One or more tests verifying that adding to a 1d-array element-wise returns what it's supposed to
- One or more tests verifying that substracting from a 1d-array element-wise returns what it's supposed to
- One 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
- One or more tests verifying that comparing arrays (by `==`) returns what it is supposed to - which should be a boolean.
- One 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.
- One 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 one (or more) tests of each functionality (test with different data types for example).

We recommend implementing your tests with pytest[4]. 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[5].

Docstrings are not needed for tests, but you should have a comment that describes what the method does in `test_Array.py` Good examples can be found here [6].

3.3 Additional tests for 2D Arrays (2 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`

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

[5] If you've forgotten how to - checkout the lecture slides or live lecture recording.

[6] `https://docs.pytest.org/en/6.2.x/getting-started.html#create-your-first-test`

Docstrings are not needed for tests, but you should have a comment that describes what the method does in `test_Array.py` Good examples can be found here[7].

3.4 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 $(3, 2)$.

```python
# 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
```

Listing 3.4.1: Defining a 2D- Array.

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.

An inspiration for a function to do so, is given below:

```python
def flat_array(self):
  """Flattens the N-dimensional array of values into a 1-dimensional array.
  Returns:
    list: flat list of array values.
  """
  flat_array = self._array
  for _ in range(len(self.shape[1:])):
    flat_array = list(chain(*flat_array))
  return flat_array
```

Listing 3.4.2: Example on how to flatten an array.

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

Make all the methods implemented in `3.1 - Implement the Array Class for 1D Arrays` work with n-dimensional arrays. Make sure not to break your previous tests.

[7] `https://docs.pytest.org/en/6.2.x/getting-started.html#create-your-first-test`

Good luck!