EMAT10007 – Introduction to Computer Programming

Exercises – Week 7. Numpy

Exercises this week will explore the use of Numpy, a python module designed for efficient and straightforward manipulation of arrays.
Note: The standard was to import the numpy module is by adding the following line at the start of your code: `import numpy as np`. Any function belonging to the numpy module can then be accessed by writing, for example, `np.array()`.

Part 1. User input

Exercise 1 - Element-wise operations

1. Create a list of integer values from 2 to 9 called `numsList`. Display the list using the `print` function.
2. Now convert that list to a numpy array called `numsArray`. Display the array by using `print` again. Does this differ in any way from the list display?
3. Add 3 to each element in `numsList` using list comprehension. Then add 3 to each element in `numsArray`. Check your outputs are the same using `print`.
4. Subtract 2 from each element in `numsList` and `numsArray`. Check your outputs are the same.
5. Create a new variable called `x` in a single line of code, which is a numpy array containing numbers from 1 to 10.
   Hint: Use `linspace`.
6. Check the type of each element in `x`. Can you think of different ways of creating this numpy array so that each element is of the type: `numpy.int64`?
7. Multiply each element in `x` by 2, then print the first 5 elements of `x`. Your output should look like this:
   ```
   [2.  4.  6.  8.  10.]
   ```
8. Finally, divide each element of `x` by 2, and print out only the even elements. Your output should look the same as for the previous question.

Exercise 2 - Shaping arrays

1. Create a 2d numpy array named `y` that looks like the following :
   ```
   [[2.  2.  2.],
   [2.  2.  2.]]
   ```
   Hint: Use the numpy `ones()` function.
2. Modify `y` so it has the following shape:
   ```
   [[2.  2.],
   [2.  2.],
   [2.  2.]]
   ```
   Hint: Use the numpy `reshape()` function.

3. What happens if you now try `y = y.reshape(4,2)?  Why?`
4. Modify `y` to look like the following:
```
[[1.  2.],
[3.  4.],
[5.  6.]]
```
5. Perform an element-wise multiplication of `y` with itself and save the output as a new array `z`.
6. Use the numpy `dot` function to perform a dot product of `y` with itself. Remember you may need to reshape the array for this to work, refer to the lecture slides if needed.
   Note: There is a specific type of reshape that is called a transpose. Try printing the output of `y.T`

Part 2. Numpy Functionality

Exercise 3 - Lottery game

Here we will code a short program to implement a "Lotto" style game, in which 3 numbers from 1-20 are chosen by the user and then randomly drawn by our program.

1. Use the `input` function to have the user input a list of 3 numbers from 1-20, and convert this to a numpy array named `guessedNumbers`.
2. Implement additional checks in the input phase to ensure the user inputs an array of 3 numbers, with each number being in the range 1-20. If an incorrect input is entered by the user, ask them to re-enter the input until it is in the correct format.
3. Use the numpy `random` function to produce an array of random numbers from 1 to 20, named `lotteryNumbers`.
4. Check whether the arrays are equivalent, to see if the user won the jackpot. If they get all three numbers right, print out the following:
   `Congratulations!  You guessed all 3 numbers and won the jackpot!`
   Hint: You could compare the arrays using the numpy `array_equal` function.
5. Numbers can currently be repeated both in `guessedNumbers` and `lotteryNumbers`. Implement additional checks on the user input to avoid this. Then, ensure the randomly generated numbers are non-repeated.
   Hint: For `lotteryNumbers`, try modifying the following function to fit your needs:
   `np.random.choice(range(20), 10, replace=False)`
6. (*) Your program only checks for the jackpot. Think of ways to extend its functionality so it can tell the user how many numbers they guessed right.