Assignment 2A

November, 2023

Exercises to practice

Use the exercises below to practice before going to problems 2 and 3. You don't need to submit them (you only need to submit the problems in this assignment), but they will help you to break problem X into smaller problems.

Exercise 1

The code below creates an array *x* with values from 0 to 30 and steps of 0.1. Suppose you want to evaluate the mathematical function $f(x) = x^2 + 2 * x + 3$ for every value of the vector *x*. Write a function that receives *x* and returns a vector with *f(x)*.

```python
x = np.arange(0,30,0.1)
```

Use pyplot to plot f(x) and x.

Exercise 2

Implement a function swap_idx that receives an array x, and two integers i and j as parameters. Your function should swap the values of x at the indices i and j, if this operation is possible. Check if those are valid indices for x before doing the swap.

Exercise 3

Implement a function sum_array that receives an array of integers x, and returns the sum of all the even elements in x.

You can test if your function works with the following array:

```python
x = np.arange(0,100,3) # similar to range(), but creates an array
```

Exercise 4

Write a function that receives a 1D-array x and create a new array y, swap the first and second half of x. Ex.: the array [0, 1, 2, 3, 4, 5] becomes [3, 4, 5, 0, 1, 2]. If the number of elements is odd, the middle element is repeated.

Problem 3 - Think-Pair-Share

In the first part of this assignment you will create and manipulate 1-dimensional numpy arrays. An array has 1 dimension if their shape has only one element. That is, *len(x.shape)* is 1, for the array x. Notice you can have a 2d array with only 1 row/column that looks like 1d. In that sense, *numpy.zeros((10,1))* is a 2d array, while *numpy.zeros((10,))* is a 1d array.

Implement the following functions to operate with 1D arrays:

- **random_array(a: int, b: int, n: int)**: That uses the function *randint* from module random to create an 1d-array of integers of size *n*. The values of the array should be randomly sampled between a and b (inclunding both a and b).
  - Your function should deal with the case in which a and b are swapped, correcting so that a is the lower value.
  - The function should return the created array, and the type of the array should be int32.
- **element_mult(x: numpy.ndarray, y: numpy.ndarray)**: implement a function called that receive two 1-dimensional arrays *x* and *y* and returns a new array z with the element-wise multiplication of x and y (that is, the ith element of z is the multiplication of the ith elements of x and y). Compute each value of z individually (do not use x*y).
  - If x and y have different lengths, your function should return None.
  - The array z should inherit the type of either x and/or y (ex.: If x and y are int, z is also int. If x is float and y is int, z is float).
- **find_max(x: numpy.ndarray)**: implement a function that receives an array x (of positive numbers) and returns an integer with the index of the maximum value in the array. (Do not use any numpy or built-in **max** functions, you should use loops and conditions to find the maximum value).
  - In case of a tie, your function should return the index of the last occurring maximum value.
  - If there are negative values in the array the function should return None.

The *main()* function of the script creates an array and plot it as a time-series, highlighting the maximum value. You can use that code to test your functions *random_array()* and *find_max()* once they are ready.

Problem 4 - Think-Pair-Share

Arrays can also have more than 1 dimension. A 2D array, often called a matrix, can be used, for example,to represent a system of equations and help us find it's solution. Also, some matrices have interesting properties. For example, square matrices (same number of rows and columns) always have an inverse matrix. Another example is when a 2D is a 'magic square'. That is, when the sum of every column, row and diagonal is a constant number.

Write the following functions that create and manipulate 2D matrices (and sometimes N-dimensional matrices). Some of those functions have a very simple solution if you use numpy (np)/built-in functions, which is not the goal of the exercise. I specify in each question things that you should not use in your solutions. You can always use: np.zeros, np.ones, np.where, slicing and an array shape.

- **transpose(x: numpy.ndarray)**: write a function that returns a new array y that is the transpose of the 2D array x. In other words, the value of x[i,j] should be equal to y[j,i] for every possible i and j. If the shape of x is (10,2), the shape of y should be (2,10). (You should implement your own function, do not use x.T or numpy.transpose()).
    - The type of elements in y should be the same of elements in x.
- **is_square(x: numpy.ndarray)**: receives an array x and checks if all the dimensions of this array have the same length. The function returns True if so, and False otherwise.
    - Your function should generalize for more than 2 dimensions. If all the dimensions of a N-dimensional array have the same size, the function should return True.
    - If the function receives a 1D array with only one element, it should also return True, and False if the 1D array have more than 1 element.
- **is_magic(x: numpy.ndarray)**: receives an array x and checks if this is a magic square. In a magic square, the sum of every column, row and the 2 diagonals are equal. If that is the case, the function should return True. It returns False otherwise. (compute the sums and the diagonal manually, without using numpy or built-in functions). An example of a magic matrix is also given in *main()*.
    - If the array has more or less than 2 dimensions the function should return None.
    - If the array is not a square matrix the function should return None.