Exercises for Week 2

It is recommended to solve these exercises on the HPC platform, so you gain some experience. Remember to activate the course conda conda environment first (see guide Initializing the 02613 Conda Environment on Learn).

1. Basic Python

1. **Autolab** Write a Python function called `listsum` that sums numbers in a list.
   Input: A Python list of numbers.
   Output: A single number which is the sum of the numbers in the input list.
   Example: The input [1, 2, 3, 4] should return 10.
2. **Autolab** Write a Python function called `deduplicate` that removes duplicates from a list. Hint: you can use the Python builtin `set`.
   Input: A Python list which may contain duplicate elements.
   Output: A Python list containing the unique elements of the input list.
   Example: The input [1, 2, 3, 3, 2, 2, 4] should return [1, 2, 3, 4].
3. **Autolab** Write a Python function called `sorttuples` that sorts a list of tuples according to their last element. Hint: you may use the builtin function `sorted`. Look at the `key` parameter.
   Input: A Python list of tuples.
   Output: The input list with elements sorted according to the last element in each tuple.
   Example: The input [(2, 5), (1, 2), (4, 4), (2, 3), (2, 1)] should return [(2, 1), (1, 2), (2, 3), (4, 4), (2, 5)].
4. **Autolab** Write a Python function `squarecubes` which recieves a Python list of numbers and returns two lists containing, respectively, the squares and cubes of each number.
   Input: A Python list of numbers.
   Output: A tuple of two Python lists. The first list contains the squares of the numbers in the input list. The second list contains the cubes.
   Example: The input [1, 2, 3, 4] should return ([1, 4, 9, 16], [1, 8, 27, 64]).
5. **Autolab** Write a Python *program* that will receive any number of numerical grades as command line arguments. It must then compute the mean and print it back to the user followed by "Pass" if the mean is at least 5 and "Fail" otherwise. Hint: you can use `sys.argv` to access the commandline arguments.
   Input: Any number grades given as command line arguments. Each grade will be a number.

Output: The mean grade, followed by a space, followed by `Pass", if the mean is at least 5, otherwiseFail"`.
Example: For the input 0, 2, 4 the program should print the string `4.0 Fail". For the input [4, 7, 10, 12] the program should print the string`8.25 Pass".

6. **Autolab** Write a Python *program* that will receive any amount of numbers as command line arguments. It must then remove all odd numbers and print out the list of even numbers. Hint: you can use the builtin function `filter` or a list comprehension (see example in link).
   Input: Any amount of numbers as command line arguments.
   Output: All even numbers in the input.
   Example: For the input 0, 1, 4, 2, 3, -2 the program should print the string “[0, 4, 2, -2]".

7. **Autolab** Write a Python class `Student`, that has the student name and a list of courses the student is attending as attributes. These should given as argument to the class constructor. The class should also have a method `attends` that receives a course name and returns `True` if the student is attending that course and `False` if not.
   Constructor input: The first argument is a string with the student's name. The second is a list of course names, where each course name is a string.
   `attends` input: A string with a course name.
   `attends` output: `True` if the student attends that course, i.e., it is in the list of courses the student attends. `False` if not.
   Example: If we create a student as
   `s = Student('X', ['01005', '02613'])`, we expect that
   `s.attends('01005')` and `s.attends('02613')` returns `True` and
   `s.attends('02510')` returns `False`.

8. **Autolab** Write a Python function `coursestudents` that a list of `Student` classes as well as a course name. It must return a new list containing the names of the students that attends that course.
   Input: The first argument is a Python list of `Students`. The second is a string specifying a course name.
   Output: A list containing the names of the students that attends the given course.
   Example: Given the list of `Students`:
   `students = [Student('A', ['01005']), Student('B', ['02613']), <br >Student('C', ['01005', '02613'])]`, we expect that
   `coursestudents(students, '02613')` returns `['B', 'C']`.

2. Basic Numpy

1. **Autolab** Write a Python function `magnitude`, which takes an n-dimensional vector stored as a

NumPy array as input and return its magnitude, i.e., norm.
Input: An n-dimensional vector stored as a NumPy array.
Output: A number giving the magnitude of the input vector.
Example: For the input [1, 1, 3, 3, 4] we expect the output 6.

2. **Autolab** Write a Python *program* that will receive any amount of numbers as command line arguments. These numbers are the components of an n-dimensional vector. The program must then print the magnitude of that vector.
   Input: An n-dimensional vector where each component is given as a command line argument.
   Output: A number giving the magnitude of the input vector.
   Example: For the input 1, 1, 3, 3, 4 we expect the output 6.

3. **Autolab** Write a Python *program* that will receive any amount of numbers as command line arguments. These numbers are the diagonal of an $n \times n$ matrix. The program must then save this matrix as a .npy file. Hint: use the numpy.save function.
   Input: The diagonal of an $n \times n$ matrix where each component is given as a command line argument.
   Output: A new .npy file containing a matrix with the input numbers as the diagonal.
   Example: For the input 1 8 4 5, we expect the program to create a .npy file with the matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}.$$

4. **Autolab** Write a Python *program* that receives the path to a .npy file as a command line argument. This file will contain an $n \times m$ matrix. The program must then save two new files:

   a) 'cols.npy' containing an m-dimensional vector with the means of each column
   b) 'rows.npy' containing an n-dimensional vector with the means of each row.

   Hint: use the numpy.load function.
   Input: The path to an .npy file containing an $n \times m$ matrix.
   Output: Two new files 'cols.npy' and 'rows.npy' containing, respectively, the column and row means.
   Example: For a file containing the matrix

$$\begin{bmatrix} 1 & 3 & 2 & 0 \\ 5 & 7 & 3 & 9 \\ 9 & 2 & 4 & 6 \end{bmatrix}.$$

we expect 'cols.npy' to contain the vector [5, 4, 3, 5] and 'rows.npy' to contain the vector [1.5, 6, 5.25].

5. **Autolab** Write a Python program that receives the path to a .npy file as well as a strictly positive integer p as a command line argument. The .npy files will contain a matrix A, and the program must create a new .npy file containing the results of multiplying A with itself p times, i.e., A^{p+1}. The program must *also* print the time (in seconds) that it took to perform these multiplications. Hint: you can use `perf_counter` from the `time` module.
Input: The path to an .npy file containing a matrix A and a strictly positive integer p given as command line arguments.
Output: A .npy file containing A^{p+1}. Also, the time (in seconds) it took to compute A^{p+1} must be printed.
Example: For a file containing the matrix
$$\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}.$$
and for $p = 2$, we expect the output to contain the matrix
$$\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}^{2+1} = \begin{bmatrix} 136 & 216 \\ 360 & 568 \end{bmatrix}.$$

6. **Autolab** Run the above program as a batch job. Assume the input file always has the path ./input.npy and p is always 10. Submit the job to the hpc' queue and request 1 core. Remember to show good etiquette and also specify a job name, expected run time, memory usage and files for stdout and stderr. Hint: remember to initialize the course conda environment in the job script (see Initializing the 02613 Conda Environment).