# Koç University
# COMP 125: Programming with Python
# Homework #4

**Deadline: December 27, 2020 at 23:59**
**Submission through: Blackboard**
Make sure you read and understand every part of this document

This homework assignment contains 3 programming questions. Each question may contain multiple parts.

Download Hw4.zip from Blackboard and unzip the contents to a convenient location on your computer. The Python files (.py extension) contain starter codes for the programming questions. Remaining files are sample text/data files for the File I/O questions.

Solve each question in its own Python file. **Do not change the names of the files. Do not change the headers of the given functions (function names, function parameters).** When you are finished, see the end of this document for submission instructions.

- Your submitted code should run as is. It should not yield syntax errors.
- We will not edit or comment/uncomment parts of your code in order to fix syntax errors.

## Q1: Sparse Matrix Operations - 30 pts

A sparse matrix is a matrix where most of the elements are zero (i.e. it has very few non-zero elements). They occur frequently in scientific computation and engineering applications. A 5x5 example is given below:

$$\begin{bmatrix} 0 & 1 & 5 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 9 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 2 & 0 & 0 & 8 \end{bmatrix}$$

These matrices can get quite large. To save on space, only their non-zero elements are stored. There are multiple ways of storing these matrices. In this homework, you are going to use dictionaries to store these matrices. Let $A_{NxM}$ be a sparse matrix with $N$ rows and $M$ columns. Let $A_{ij}$ be the element of $A$ in the $i^{th}$ row and the $j^{th}$ column ($i=0...N-1$, $j=0...M-1$). then this sparse matrix should be represented as follows:

- Let `sp_A` be a python dictionary that will store $A$
- The keys are tuples of the row ($i$) and column ($j$) indices of non-zero elements. (`sp_A[(i,j)]`=$A_{ij}$, $A_{ij} \neq 0$)
- A special key (-1) stores the matrix dimensions (`sp_A[-1]`=`[N,M]`)

The starter code for this question is given to you in *sparse_matrices.py*. This question has three parts:

Part 1: You will be given the matrices in a list of lists format. You need to convert them to the aforementioned sparse representation. Fill in the `dense_to_sparse` function to complete this part.

$$A = \begin{bmatrix} 0 & 3 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$\Rightarrow sp\_A[(0,1)] = 3, sp\_A[(1,0)] = 1, sp\_A[-1] = [2,3]$$

Part 2: You are going to implement the matrix transpose operation (switching the row and column indices of non-zero elements, $A_{ij} \leftarrow A_{ji}$) for the dictionary based sparse matrix representation described in this question. Fill in the `sparse_transpose` function to complete this part.

$$A = \begin{bmatrix} 0 & 3 & 0 \\ 1 & 0 & 0 \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} 0 & 1 \\ 3 & 0 \\ 0 & 0 \end{bmatrix}$$

Part 3: The code for matrix multiplication for list of lists implementation is given to you. You are going to implement matrix multiplication for the dictionary based sparse matrix representation described in this question. Fill in the `sparse_mat_mult` function to complete this part.

Let $A_{NxM}, B_{MxK}$, be two matrices and $C_{NxK} = AB$ their multiplication. The elements of $C$ are calculated as:

$$\Rightarrow C_{ij} = \Sigma_{k=0}^{M} A_{ik} B_{kj}$$

For all of the parts, follow the comments. **Do not iterate over the entire rows and columns for any of the parts**! This will be inefficient and you will lose points. There is an example function, `sparse_mat_add`, as an example for you.

There are also other functions that may help you debug or get inspirations from. Make sure to go over the *sparse_matrices.py* file, both comments and code, for your own benefit.

# Q2: Protein Center of Mass Calculation - 35 pts

The Protein Data Bank archive serves as a single repository of information about the 3D structures of proteins, nucleic acids and complex assemblies. Each PDB file contains various kinds of information. The type of information is indicated in the first six characters of each line, such as HEADER, SOURCE, COMPND, AUTHOR, REMARKS, etc. For this homework, you just need to concentrate on lines beginning with the word “ATOM”. An example is given below, where the columns represent the atom record, atom number, atom identifier, amino acid type, chain identifier, residue sequence number, x-coordinate (in Å), y-coordinate (in Å), z coordinate (in Å), occupancy, β-factor and element symbol respectively. The symbol Å denotes Angstrom (1Å= $10^{-10}$ m).

```
HEADER    CHROMOSOMAL PROTEIN                      02-JAN-87   1UBQ
TITLE     STRUCTURE OF UBIQUITIN REFINED AT 1.8 ANGSTROMS RESOLUTION
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: UBIQUITIN;
COMPND   3 CHAIN: A;
COMPND   4 ENGINEERED: YES
SOURCE    MOL_ID: 1;
SOURCE   2 ORGANISM_SCIENTIFIC: HOMO SAPIENS;
SOURCE   3 ORGANISM_COMMON: HUMAN;
SOURCE   4 ORGANISM_TAXID: 9606
KEYWDS    CHROMOSOMAL PROTEIN
EXPDTA    X-RAY DIFFRACTION
AUTHOR    S.VIJAY-KUMAR,C.E.BUGG,W.J.COOK
REVDAT   5   09-MAR-11 1UBQ    1       REMARK
REVDAT   4   24-FEB-09 1UBQ    1       VERSN
REVDAT   3   01-APR-03 1UBQ    1       JRNL
```

…

```
ORIGX3      0.000000  0.000000  1.000000        0.00000
SCALE1      0.019670  0.000000  0.000000        0.00000
SCALE2      0.000000  0.023381  0.000000        0.00000
SCALE3      0.000000  0.000000  0.034542        0.00000
ATOM      1  N   MET A   1      27.340  24.430   2.614  1.00  9.67           N
ATOM      2  CA  MET A   1      26.266  25.413   2.842  1.00 10.38           C
ATOM      3  C   MET A   1      26.913  26.639   3.531  1.00  9.62           C
ATOM      4  O   MET A   1      27.886  26.463   4.263  1.00  9.62           O
ATOM      5  CB  MET A   1      25.112  24.880   3.649  1.00 13.77           C
ATOM      6  CG  MET A   1      25.353  24.860   5.134  1.00 16.29           C
ATOM      7  SD  MET A   1      23.930  23.959   5.904  1.00 17.17           S
ATOM      8  CE  MET A   1      24.447  23.984   7.620  1.00 16.11           C
ATOM      9  N   GLN A   2      26.335  27.770   3.258  1.00  9.27           N
ATOM     10  CA  GLN A   2      26.850  29.021   3.898  1.00  9.07           C
ATOM     11  C   GLN A   2      26.100  29.253   5.202  1.00  8.72           C
ATOM     12  O   GLN A   2      24.865  29.024   5.330  1.00  8.22           O
ATOM     13  CB  GLN A   2      26.733  30.148   2.905  1.00 14.46           C
ATOM     14  CG  GLN A   2      26.882  31.546   3.409  1.00 17.01           C
ATOM     15  CD  GLN A   2      26.786  32.562   2.270  1.00 20.10           C
ATOM     16  OE1 GLN A   2      27.783  33.160   1.870  1.00 21.89           O
ATOM     17  NE2 GLN A   2      25.562  32.733   1.806  1.00 19.49           N
ATOM     18  N   ILE A   3      26.849  29.656   6.217  1.00  5.87           N
ATOM     19  CA  ILE A   3      26.235  30.058   7.497  1.00  5.07           C
                                  X       Y       Z                    ELEMENT
```

Implement the following functions:

**A. pdb_parser(pdb_filename):**

Receives one argument, **pdb_filename**. Opens the PDB file, reads all lines starting with ATOM and stores the **X**, **Y**, **Z** coordinates and the **ELEMENT** type of the atom in a list, **atoms**.

e.g.

**atoms= [[27.340, 24.430, 2.614,'N'],[26.266, 25.413, 2.842,'C'],...]**

**atoms** list is of length **N**, number of atoms that make up the protein in the PDB file. The function should return **atoms**.

**B. center_of_mass(atoms):**

This function calculates the center of mass of the protein, as follows.

$$\mathbf{r}_{\mathrm{CM}} = \frac{\sum_{i=1}^{N} m_i \mathbf{r}_i}{\sum_{i=1}^{N} m_i}$$

$\mathbf{r}_{cm}$ is coordinates of the center of mass of the protein, $\mathbf{r}_i$ is a list containing the [x, y, z] coordinates of the $\mathbf{i}_{th}$ atom), $\mathbf{m}_i$ is the mass of the $\mathbf{i}_{th}$ atom and **N** is the total number of atoms. $\mathbf{m}_i$ should be obtained from the dictionary **mass={'C':12.01, 'O':16.00, 'H':1.008, ...}** by using the **element** type of the $\mathbf{i}_{th}$ atom as key. **mass** dictionary is already provided in the pdb.py template.

**C. shift(atoms, vec):**

This function translates the protein by **vec** and returns the updated coordinates of the protein, **atomsnew**. **vec** is a list of size 3.

E.g. vec=[a, b, c], atoms=[[x, y, z, 'C']] -> atomsnew=[[x+a, y+b, z+c, 'C']]

After implementing these functions, demonstrate them as follows:

i) Read the 1ubq.pdb file and parse the information.

ii) Calculate the center of mass.

iii) Shift the coordinates of the protein such that the molecule's center of mass is at the origin.

The sample output from the program should be:

```
Center of mass of the protein is 30.317, 28.775, 15.353
New center of mass of the protein is  0.000,  0.000,  0.000
```

# Q3: Course Scheduling - 35 pts

In this question, you will practice File I/O with multiple input files. In particular, you are given three files related to the scheduling of classes at Koç University in Spring 2020:

- instructors.txt: contains the names, IDs, and instructors of courses
- locations.txt: contains the physical location (room) of each course
- times.txt: contains the timeslot (day, start time, end time) of each course

The starter code for this question is given in **CourseScheduling.py**. Implement the empty functions in the starter code to solve the following parts.

**PART A:** Write a function called read_schedules() that reads course schedules from the 3 given files and stores them in a dictionary of dictionaries:

```
data = {    "COMP110":  { "Name": "Introduction to Programming with Matlab",
                          "Instructor": "Emre Kutukoglu",
                          "Location": "SCI103",
                          "Days": "TuTh",
                          "Start Time": "8:30",
                          "End Time": "9:45"},
            "COMP125":  { "Name": "Programming with Python",
                          "Instructor": "Ayca Tuzmen",
                          "Location": "ENGZ50",
                          "Days": "TuTh",
                          "Start Time": "13:00",
                          "End Time": "14:15"},
            ....
        }
```

For the outer dictionary data, the key should be the course ID (e.g.: "COMP110", "COMP125", "ELEC204") and the value should be a dictionary. For the inner dictionary, the keys should be "Name", "Instructor", "Location", "Days", "Start Time", "End Time"; and the values should be the corresponding information for that course as shown above.

The return value of read_schedules() should be the main dictionary data.

**Hint**: You should first read instructors.txt when creating the outer dictionary data, and later populate the location and time details using locations.txt and times.txt.

**PART B:** Some courses included in instructors.txt are missing location assignments or time assignments, i.e., they are included in instructors.txt but not in locations.txt or times.txt. Write a function find_unscheduled(data) that:

- Takes as input the main dictionary data constructed above
- Has two return values:
    - First return value is the list of courses that have no location
        - Example: ["COMP306", "INDR460"]
    - Second return value is the list of courses that have no timeslot
        - Example: ["ELEC422", "MECH435", …]

**PART C:** Write a function clean_schedule(data, courses_to_remove) such that:

- data is the main dictionary constructed in Part A
- courses_to_remove is the list of courses that should be removed from data
  - Example: [“ELEC422”, “MECH435”, …]
- Your function should return the resulting dictionary after the courses are removed from it

**Caution**: This function must return the resulting dictionary, do not just modify the dictionary in-place.

**PART D:** Write a function find_instructor(data, courseID) such that:

- data is the main dictionary constructed in Part A
- courseID is a string containing the ID of one course
  - Example: courseID = “COMP125” or courseID = “MECH433”
- Your function should return the instructor who is teaching that course
  - If the given courseID does not exist in data, your function should return the following string: “NA”

**PART E:** Write a function find_subj_courses(data, subject) such that:

- data is the main dictionary constructed in Part A
- subject is the 4-letter subject area
  - Example: subject = “COMP” or subject = “ELEC” or subject = “MECH”
- Your function should return the list of all courses from that subject area
  - Example: [“COMP110”, “COMP125”, “COMP131”, …, “COMP437”]

**PART F:** Write a function build_schedule(data, courses) such that:

- data is the main dictionary constructed in Part A
- courses is a list containing course IDs that a student is interested in taking
  - Example: courses = [“COMP125”, “MECH301”, “COMP305”, “ELEC301”]
- File output: Your function should create and write to a file called “student.txt” the schedule of this student. For the above example, contents of student.txt will be:

  COMP125,ENGZ50,TuTh,13:00,14:15
  MECH301,SOSZ27,MoWe,10:00,11:15
  COMP305,SOSB07,MoWe,8:30,9:45
  ELEC301,SNA159,MoWe,13:00,14:15
- If the given courses have a time conflict, i.e., they overlap, then the student cannot take the given list of courses simultaneously. In this case, your program should write only the following to student.txt (course IDs, locations, times should not be written):

  ** Time conflict **

# Submission and Grading

Solve each question in its own file. **Do not change the names of the files. Do not change the headers of the given functions (function names, function parameters).**

When you are finished, compress your Hw4 folder containing all of your answers. The result should be a SINGLE compressed file (file extension: .zip, .rar, .tar, .tar.gz, or .7z). Upload this compressed file to Blackboard.

Follow instructions, input-output formats, return values closely. Your code may be graded by an autograder, which means any inconsistency will be automatically penalized.

**<u>You may receive 0 if one or more of the following is true:</u>**

- You do not follow the submission instructions, i.e., you submit something other than a single compressed file.
- Your compressed file does not contain your Python files or the file names are wrong.
- Your compressed file is corrupted.
    - **After you submit, you should download your submission from Blackboard to make sure it is not corrupted and it has the latest version of your code.**
- Your code contains syntax errors.
- Your code does not run without errors the way that it is submitted.
    - We should not have to open your file and comment/uncomment parts of it in order to run your code.
- Your code does not terminate, e.g.: it contains infinite loops.

You are only going to be graded based on your Blackboard submission. We will not accept homework via e-mail or other means.

**Best of luck and happy coding!**