EXAM PYTHON (90 + 3EC POINTS)

- This is an examination. It is to be your work and your work alone.
- No exchange of information with another human entity in any form is acceptable.
- Reading Google documents to find out how to solve a problem is not acceptable, including using chatGPT.
- It is ok to use class material, notes, your programs, and any other notes you have written for class.
- You can use the online python documentation pages listed at the end of the Python Notes.

These instructions are general for all the scripts that you will write in python3.

```
Instructions
#class header

#Q1
var1=124
print(var1)

#Q2
var2=3+var1
print(var2)
```

```
Deadline and late penalty
The deadline is 8:30pm with 3 min grace period.
After 8:33pm, the penalty is 2 points for each late minute.
Gradescope Exam2 will close at 8:40m.

IF YOU HAVE TECHNICAL ISSUES DURING THE EXAM LET THE PROFESSOR KNOW
```

Policy for cheating: sharing code on the exam or using chatGPT or using google to find out how to solve a problem, is unacceptable and will earn you a 0 and take you straight to the ethics board. Do not share any study documents either!

We give partial points in case of syntax errors.

We **will subtract -0.5 points for missing or incorrect header and/or Q&A for each script.**

After you download the Exam2-F2023.zip from Canvas, unzip the file if necessary, and a directory called **Exam2-F2023** will be created. Change into **Exam2-F2023** and start your work in that directory.

We take points off for using python concepts or modules out of class material.

1. **(23)** In the provided script **ex1-f23-argv.py** a dictionary D has been defined. Notice that it is a nested dictionary.

```
D={'colors': ['blue','red','yellow'],
   'numbers': [{'integer':[1,2,3]} , {'float':[1.2,3.4]}]}
```

Edit the script and do the following:

Q0. As a comment include this sentence as Q0 for this script.
I promise not to communicate with another human being in any way about this exam.

Q1. (4) One integer number is provided from the command line, like this:

```
python3 ex1-f23-argv.py 3
```

Store that integer number in a variable called **num**. Variable **num** should be int type.

If you do not know how to do this, store one number of your choice in variable num for a loss of points on this part.

Q2. (6) Access nested values in D, and append the number stored in **num** to the list of integer numbers in dictionary D and print D to screen. Notice that one list contains float numbers, and another integer numbers.

Q3. (7) Randomly pick a color from the list of colors in D and repeat that color **num** times and print to screen.

Q4. (6) Use list comprehension and a function to calculate the sum of numbers from 1 to **num** (1 and num included) and print the total sum to screen, with some text. See output below. Do not use concatenation in the print function.

If the number 3 is entered from command line, like in the example in Q1, the output of the script should be (the color name can vary):

```
{'colors': ['blue', 'red', 'yellow'], 'numbers': [{'integer': [1, 2, 3, 3]},
{'float': [1.2, 3.4]}]}
blueblueblue
The sum is 6
```

2. **(38)** In a molecule of an element, two or more atoms belonging to the same element can be chemically bonded together. The number of atoms present in a molecule of an element is known as the **atomicity** of that element.
Based on the atomicity, molecules of elements are classified as:

- Monoatomic molecules. Their atomicity is 1. For example, all noble gases such as helium (He), neon (Ne) etc.
- Diatomic molecules. Their atomicity is 2. For example, the elements hydrogen (H2), nitrogen (N2), Oxygen (O2), etc.
- Triatomic molecules. Their atomicity is 3. For example, ozone (O3)
- Polyatomic molecules. Their atomicity is greater than 3. For example, the elements phosphorus (P4) and sulfur (S8).

The provided data set **data-elements.csv** contains information for some elements. The fields are organized as follow:

1st Atomic Number
2nd Name
3rd (Symbol)
4th Atomicity

The first line of the file is a header line, reporting the fields.

A module called **weight.py** is provided, and contains a dictionary called **Dw**, whose keys are the symbols, and the values the corresponding atomic weights.

In a script called **ex2-f23-elements.py** do the following:

Q1. (2) Import dictionary **Dw** from the **weight** module. Use function import.

Q2. (3) Read in the set **data-elements.csv** and store it in a list called **L**. Print **L** to screen.

Q3. (3) In one line of code, remove the header line (first line of the file) from list L, and print that header line in uppercase. For this use a list method and a string method in one line of code. The output should be.

```
ATOMIC NUMBER,NAME,(SYMBOL),ATOMICITY
```

If you cannot do this, for a loss of points on this part, copy and paste this code below:
```
L=L[1:]
```

Q4. (6) Make a dictionary called **Datomicity** out of list **L**. The dictionary should be of this form:

```
Datomicity={symbol: atomicity}
```

The keys are the symbols, and the values the atomicities in integer type.
Note that, the symbols in the data set and so in list L are surrounded by parenthesis like this (symbol), and you should not include parenthesis in the keys.
Print **Datomicity** to screen.

```
{'H': 2, 'He': 1, 'Li': 1, 'Be': 1, 'B': 1, 'C': 1, 'N': 2, 'O': 2,
'F': 2, 'Ne': 1, 'Na': 1, 'Mg': 1, 'Al': 1, 'Si': 1, 'P': 4, 'S': 8,
'Cl': 2, 'Ar': 1, 'K': 1, 'Ca': 1, 'Sc': 1, 'Ti': 1, 'V': 1, 'Cr': 1,
'Mn': 1, 'Fe': 1, 'Co': 1, 'Ni': 1, 'Cu': 1, 'Zn': 1}
```

If you cannot do this, use the output reported above and define Datomicity for a loss of points on this part.

Q5. (3) How many triatomic molecules are there? Use type conversion and a method to count. Do not use loops or comprehension. Print the result to screen like this.

```
There are 0 triatomic molecules.
```

Q6. (5) **Use dictionary comprehension** to make another dictionary called **Ddiatomic** out of **Datomicity. Ddiatomic** should have this structure and contain information only on diatomic molecules (atomicity = 2).

```
Ddiatomic={symbol: weight}
```

The keys are the symbols, and the values the corresponding molecular weights.
To calculate the molecular weight, use this formula:

molecular weight = atomic weight * atomicity

For the atomic weights use **Dw**, whose keys are the symbols, and values the atomic weights.
Print **Ddiatomic** to screen.

```
{'H': 2.016, 'N': 28.014, 'O': 31.998, 'F': 37.99680632, 'Cl': 70.9}
```

Q7. (3) Make a list of the keys of **Ddiatomic** and store it in a variable of your choice.
Make a list of the values of **Ddiatomic** and store it in a variable of your choice.
To make the lists use type conversion and do not use loops/comprehension.

Q8. (8) Make one for loop over both lists created in Q7 and print this formatted output.

```
H2    2.02
N2   28.01
O2   32.00
F2   38.00
Cl2  70.90
```

The number of spaces can vary, but the two fields should be aligned as reported above.
Do not hardcode the number 2 in the first field, for example in H2; instead, make use of the element's atomicity from dictionary Datomicity.

Q9. (5) Find the atomicity and the symbol of the polyatomic molecule with the largest molecular weight.
Polyatomic molecules have atomicity > 3.

Print the symbol, and the atomicity as reported below.
```
S8
```

Use the **Datomicity** dictionary, list comprehension and methods. No loops. Do not hardcode neither S nor 8.

In comment lines explain the steps you take to solve this problem.

3. **(29)** The area A of a rhombus is defined as:

$$A = \frac{d_1 d_2}{2} \quad \text{Eq1}$$

where d_1 and d_2 are the lengths of the two diagonals and they must satisfy this condition:
d1 > 0 and d2 > 0

Write a script called **ex3-f23-rhomb.py** and in it do the following:

Q1. (5) Make a function called ***rhombarea****,* which takes 2 parameters, d1 and d2, and calculates and returns the area reported in the Eq1 formula.

Q2. (12) Make another function called ***valid_entries***. This function:

- takes no parameters
- uses a while loop to prompt the user to enter the lengths of the two diagonals (d1, and d2), and checks that the two numbers are valid, i.e., both greater than 0 (d1 and d2 > 0).
  In the while loop use only **one input function** to prompt the user to enter **two numbers separated by a comma**. See output below.
- returns two valid diagonals d1 and d2.

Alternative - If you cannot make the function do the following for a loss of 4 points:
Make a while loop that error-check the users input (the user should enter two numbers separated by comma) and build two lists of 4 valid entries, one list for d1 and another for d2.
Instead of using the *valid_entries* function, use the two lists you made for next question.

If you cannot make the alternative, for a loss of points, make these two lists, and use them for next question.
```
d1=[0.5, 1.0, 2.0,3.0]
d2=[0.5, 2.0, 3.0, 4.0]
```

Q3. (12) Use one for loop and the two functions and write into a file called **valid.txt** four valid pairs of diagonals and the corresponding areas.
As an example – if the user enters:

```
Enter two numbers, both > 0 separated by a comma: -0.5,0.5
Enter two numbers, both > 0 separated by a comma: 0.5,0.5
Enter two numbers, both > 0 separated by a comma: 1,2
Enter two numbers, both > 0 separated by a comma: 2,3
Enter two numbers, both > 0 separated by a comma: 3,4
```

valid.txt should be:

```
d1          d2          A
0.5         0.5         0.1
1.0         2.0         1.0
2.0         3.0         3.0
3.0         4.0         6.0
```

Extra credit (3) This problem is all or nothing, there is no partial credit.

The inverse of the mathematical constant *e* can be approximated as follows:

$$\frac{1}{e} \approx \left(1 - \frac{1}{n}\right)^n \quad \text{with n=1,2,…,N}$$

Write a script **EC-f23.py** that will loop through values of *n* until the difference between the approximation and the actual value is less than 0.0001. The script should then print out the built-in value of e^{-1} and the approximation to 4 decimal places and print the value of *n* required for such accuracy.

```
The built-in value of 1/e is 0.3679
The approximation is 0.3678
The value of n is 1840
```

```
Submission to Gradescope Exam2 the following:
ex1-f23-argv.py
ex2-f23-elements.py
ex3-f23-rhomb.py
EC-f23.py (if you made it)
DO NOT SUBMIT A ZIP FILE.
```