HW8 (100 POINTS)

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

```
Instructions for Python3 Homework submission

#class header

#Q1
var1=124

#Q2
var2=3+var1
```

```
This HW has 2 parts, HW8-part1 and HW8-part2:

All parts are due Monday March 25th 11:59pm.

You will get 4 extra credit points if you submit HW8-part1 by Sunday March
17th 11:59pm.

You will get 4 extra credit points if you submit HW8-part2 by Sunday March
17th 11:59pm.
```

Policy: Work on the HW problems on your own. It is not allowed to share code in anyway. The use of chatGPT is not allowed. Using internet to find out how to solve problems is not allowed.

General Grading rules we will follow:

- We do not give points for instructions but take off points for not following them.
- We take all points off for a question if your code has syntax errors (comment out the code for partial credit).
- We grade the code, not the output of the code. Even if the output is correct, hard-cording, redundant logic, code that has not purpose in the program will not earn full credits.
- We will take points off if you use out-of-class material.
- We will take points off if you do not follow directions when a question specifies to use a certain command or write the code in a certain way.

HW8-part1 (60)

1 (30) The provided python module **proteins.py** contains a dictionary *d_weight*, where each key is one character (i.e., a one letter code for an amino acid), and the value is the corresponding amino-acid weight (w_n) in g/mol. A protein sequence is a string of characters, e.g., MGVNAVHW

Q1 (1) Add a module docstring, which describes what this module is about. Write it before the dictionary *d_weight*. Keep in mind that this module contains functions to compute different parameters of a protein sequence. You can briefly mention which parameters.

Q2 (5) Make a function called ***freq_am*** that counts the relative frequency of amino acids in a protein sequence.
This function should use a string of several characters and a single character passed to the function and return the relative frequency of that character in that string. The relative frequency (Fc) is the number of times a specific character is repeated in a string (*Nc*), divided by the total number of characters in the string (*N*). Do not use a for loop.

$$Fc=Nc/N$$

Also write a short function docstring describing what the function does.

Q3 (5) Make a function called ***aromaticity*** that calculates the aromaticity of a protein sequence. According to Lobry & Gautier 1994, the aromaticity is the relative frequency of occurrence of aromatic amino acids, i.e., the sum of relative frequencies of amino acids Phenylalanine (F), Tyrosine (Y) and Tryptophan (W).
This function takes one parameter, i.e., one string of characters, and returns the aromaticity of the protein sequence. **Make use of function *freq_am()* to calculate the aromaticity.**

$$Aromaticity=F_W+F_Y+F_F$$

Also write a short function docstring describing what the function does.

Q4 (6) Make a function called ***weight*** that calculates the molecular weight of a protein sequence. This function takes one parameter (a protein sequence) and returns the molecular weight M_w of that protein.

The formula below reports the sum of the weights of all amino acids in a protein sequence. N is the total number of amino acids (i.e., total number of characters in a string) and w_n is the weight of an amino acid. Note that the weight of an amino acid is stored in the dictionary *d_weight*.

$$M_w = \sum_{n=1}^{N} w_n$$

Also write a short function docstring describing what the function does.

Q5. (13)

- (5) Test the functions by using the if __name__== "__main__": statement and within:
  - (2) Define a variable called s1 which will store a string of characters **provided from the command line**.
    This string MGVNAVHWFRKGLRLH should be provided from the command line. See below.
  - (1) Calculate the frequency of the amino-acid (character) W in string s1, and store result in variable *s1_fa*
  - (1) Make a formatted print statement reporting the result to screen:

    `The frequency of W is 0.06`
  - (1) Calculate the aromaticity of string s1 and store result in variable *s1_aro*
  - (1) Make a formatted print statement reporting the result:

    `The aromaticity is 0.1`
  - (1) Calculate the total weight of the protein sequence stored in *s1*
  - (1) Make a formatted print statement reporting the result:

    `Weight=2191.53 g/mol`

Run the script and provide this string from the command line.

`MGVNAVHWFRKGLRLH`

DO NOT USE THE INPUT FUNCTION BUT USE AN ATTRIBUTE FROM A MODULE DONE IN CLASS.

The output should be:

```
The frequency of W is 0.06
The aromaticity is 0.1
Weight=2191.53 g/mol
```

2. (30) The provided script **use-proteins.py** contains a dictionary of protein sequences of the flavoprotein family.

The dictionary is organized as follows:

```
D={ID: sequence}
```

The keys are the protein identification codes ID and the values are the corresponding sequences.

Q1. (2) Import the module **proteins.**

Q2. (3) Make a list of keys out of the dictionary D. Name the list ID. The use of loop or comprehension will not earn you all points.

Q3. (5) Make a list of aromaticity of the sequences. If you use loops, you will lose 2 points.

Q4. (5) Make a list of weights of the sequence. If you use loops, you will lose 2 points.

Q5. (10) Use **one for loop** to print this formatted output, which includes the protein identification code, the aromaticity, and the weight.

```
Q96524 0.09 15835.57
P97784 0.11 15991.62
Q9R194 0.11 15482.02
O77059 0.08 15876.19
Q49AN0 0.10 15819.27
```

Q6. (5) Print the identification code of the protein with the largest weight. Points will be deduced if you use flow control or comprehension.

Print the weight in scientific notation with one decimal digit.
You can achieve this with multiple lines of code.

```
P97784 1.6e+04 g/mol
```

Upload the following files to Gradescope HW8-part1:
proteins.py
use-proteins.py

HW8-part2 (40)

1 (40) The provided file **global-emission.dat** contains data on global carbon dioxide (CO2) emissions by region from 1990 to 2012**.**
The regions are Europe, Canada, United States, Latin America, Africa, Asia, Australia, and Oceania. Emissions are expressed in million metric tons of carbon dioxide equivalents (MMT CO2 Eq). The data file is organized as follows:

Field 1: The region where data were recorded.
Field 2-23: The CO2 emissions from 1990 to 2012.

Write a program called **global_emission.py** that analyzes this data file.

Use cat at the IPython console to see how the data are organized within the file, and what the field separator is. Do not write this in the script.

Q1 (3) In Python there is a built-in module called **statistics** (like the math module), which contains functions to calculate statistical parameters. You should import the functions **mean** (which calculates the average) and **stdev** (which calculate the standard deviation). You will use these functions later. You can use the function or the generic import.

Q2 (4) Read in all the lines of the provided data file **global-emission.dat**, and store all the lines in a list called L.

Q3 (10) Make a dictionary called *D* where each key is a region, and each value is a list of float numbers that contains all emission values recorded from 1990 to 2012 in that region.
In addition to that, in a comment line, write your strategy for solving this problem.

If you cannot do this part, import the dictionary from the Dback.py script, and use it for the next questions.

You can also explore and use the map function to solve this problem.

As an example:
If you have a list of strings:

```
L=['1.3','2.5','3.0']
```

```
list(map(float, L))
```
map will apply float to each element of L, and return an object, which you can convert in list.

Q4 (10) Use dictionary comprehension to make a dictionary called *Dres* out of dictionary *D,* where the keys are the region names, and the values are lists containing the mean and standard deviation of the emission values.

Each key: value pair in *Dres* is the following.

```
Dres: {region name: [mean value, std value]}
```

Q5 (5) Loop over dictionary *Dres* and print this formatted output:

```
Europe                   6248.5    436.6 (MMT CO2 Eq)
Asia                    10538.8   3501.6 (MMT CO2 Eq)
United States            5433.2    317.8 (MMT CO2 Eq)
Canada                    514.5     46.2 (MMT CO2 Eq)
Latin America            1310.4    245.8 (MMT CO2 Eq)
Africa                    856.9    168.5 (MMT CO2 Eq)
Australia and Oceania     374.3     53.7 (MMT CO2 Eq)
```

The exact number of spaces between fields does not matter, but the fields should be aligned.

Q6 (8) Write the same output as in Q5 into a file called **results-emission.dat** with the *write* method, but you should add one line at the beginning reporting the fields.

```
Region name             average   stddev     Unit
Europe                   6248.5    436.6 (MMT CO2 Eq)
Asia                    10538.8   3501.6 (MMT CO2 Eq)
United States            5433.2    317.8 (MMT CO2 Eq)
Canada                    514.5     46.2 (MMT CO2 Eq)
Latin America            1310.4    245.8 (MMT CO2 Eq)
Africa                    856.9    168.5 (MMT CO2 Eq)
Australia and Oceania     374.3     53.7 (MMT CO2 Eq)
```

You can see the contents of the file by using cat at the IPython console or at bash terminal.

Upload the following files to Gradescope HW8-part2:
global-emission.py