HW6 (100 POINTS)

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

```
Instructions for Python3 Homework submission
Do NOT use python 2, make sure you are using python 3 and above.

#class header
#Q1
var1=124

#Q2
var2=3+var1
```

```
This HW has 3 parts, HW6-part1, HW6-part2, and HW6-part3 and has 3
separate submissions on Gradescope.

All parts are due Sunday March 10th 11:59pm

If you submit:
• HW6-part1 by Wed March 6th 11:59pm you will get 3 EC points.
• HW6-part2 by Friday March 8th 11:59pm you will get 3 EC points.
• HW6-part3 by Saturday March 9th 11:59pm you will get 2 EC points.
```

Policy: Work on the HW problems on your own. It is not allowed to share code in anyway. The use of chatGPT, and in general 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.

HW6-part1 (35)

1. **(11)** The provided script **ex6-planets.py** contains variables that store different parameters of our planets.

```
name=['MERCURY','VENUS','EARTH','MARS','JUPITER','SATURN','URANUS','NE
PTUNE', 'PLUTO']

temp=[167, 464, 15, -65, -110, -140, -195, -200, -225]
orb_vel=[47.4,35.0,29.8,24.1,13.1,9.7,6.8,5.4,4.7]
num_moons=[0,0,1,2,92,83,27,14,5]
```

name stores the names of the planets.
temp stores the corresponding average temperature over the whole planet's surface in degrees C (Celsius).
orb_vel stores the average velocity of the planet as it orbits the Sun, in kilometers per second (km/s). Planets orbit around the sun at very high velocity.
num_moons stores the number of IAU (International Astronomical Union) confirmed moons orbiting the planet. New moons are still being discovered.

Edit **ex6-planets.py** and do the following:

Use the variables, one for loop and one if statement to select planets with:

velocity >=10 and number of moons > 1

and print the planet's name, orbital velocity, number of moons, and temperature on screen. Format the output to obtain – the number of spaces can vary, but the fields should be aligned as reported below:

```
MARS      24.1  km/s   2   -65 C
JUPITER   13.1  km/s  92  -110 C
```

You should loop over multiple lists at the same time.

2. **(24)** The genetic code is the set of rules by which information encoded within genetic material (DNA or mRNA sequences) is translated into proteins (amino-acid sequences) by living cells. The code defines how sequences of nucleotide triplets, called codons, specify which amino acid will be added next during protein synthesis. The genetic code is highly similar among all organisms and can be expressed in a simple table with 64 entries. A three-nucleotide codon in a DNA sequence specifies a single amino acid.

List *codons* and list *amino* are provided in the script **ex6-genetic-code.py.** List *codons* reports the 3 DNA letters (codons) and list *amino* reports the corresponding amino acid one-letter codes (i.e., the first codon corresponds to the first amino acid letter code in the list, etc. …). We did not include the stop codon (or termination codon), which is a nucleotide triplet that signals the termination of translation into proteins.

Dictionary *Dwh* contains information on the hydrophobicity and weight of each amino acid and has this structure:

```
Dwh={one letter-code:[hydrophobicity, weight]}
```

Ldna contains a list of codons.

Edit the provided script **ex6-genetic-code.py** and add python3 code to do the following:

Q1. (6) Make a dictionary called *gencode* out of the two lists *codons* and *amino*. For this use only built-in functions. No loops, no comprehension.

In *gencode*, the keys should be the 3 DNA letters (codons), and the values the corresponding amino-acid 1 letter codes. **Print only the final dictionary to screen**.

If you do not know how to make the dictionary, for a loss of all points on this part, copy and paste the dictionary *gencode* from the provided file back.py.

Q2. (8) Translate the DNA list of codons *Ldna* into a string of amino acids called *seq*.
Use one for loop over list *Ldna* and dictionary *gencode* to create the string *seq* within the for loop. **Print only the final string *seq* to screen**, which should be:

```
KMFPSALVNALSLS
```

If you do not know how to do this, make this variable *seq*, for a loss of all points on this part.

```
seq='KMFPSALVNALSLS'
```

Q3. (10) Use one for loop to calculate the total weight of the amino acid sequence *seq* by using dictionary *Dwh*.

To calculate the total weight, use this formula:

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

The formula 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 *Dwh*.

After you calculate the total weight, print a formatted statement reporting only the total weight in g/mol. Format the total weight with 2 decimal precision and print out the unit g/mol.

```
The total weight is 1711.95 g/mol
```

Your script should produce this output.

```
{'GAG': 'E', 'TGC': 'C', 'GCA': 'A', 'AGT': 'S', 'TTA': 'L', 'GGA':
'G', 'CAT': 'H', 'CAC': 'H', 'GGC': 'G', 'CCC': 'P', 'ACA': 'T',
'ACT': 'T', 'GAT': 'D', 'CTA': 'L', 'TTC': 'F', 'GCT': 'A', 'GAC':
'D', 'CCT': 'P', 'CGC': 'R', 'GCC': 'A', 'AAA': 'K', 'GGT': 'G',
'CTT': 'L', 'AGC': 'S', 'GTA': 'V', 'ATT': 'I', 'GAA': 'E', 'CAA':
'Q', 'CCG': 'P', 'GCG': 'A', 'ATC': 'I', 'TTG': 'L', 'CAG': 'Q',
'AAT': 'N', 'AAC': 'N', 'CGT': 'R', 'ATA': 'I', 'TAT': 'Y', 'TAC':
'Y', 'AGG': 'R', 'ACG': 'T', 'TGT': 'C', 'AGA': 'R', 'CTC': 'L',
'TCC': 'S', 'TGG': 'W', 'CCA': 'P', 'TCG': 'S', 'TCA': 'S', 'GGG':
'G', 'CGA': 'R', 'ATG': 'M', 'AAG': 'K', 'CTG': 'L', 'CGG': 'R',
'GTG': 'V', 'GTC': 'V', 'TCT': 'S', 'ACC': 'T', 'TTT': 'F', 'GTT':
'V'}
KMFPSALVNALSLS
The total weight is 1711.95 g/mol
```

Upload your files to Gradescope HW6-part1:
ex6-planets.py
ex6-genetic-code.py

HW6-part2 (30)

1. (22) Clouds are generally classified as high, middle, or low level. The height of the cloud is the determining factor, but the ranges vary depending on the temperature. For example, in tropical regions the classifications may be based on the following height ranges (given in feet):

low	(0,6500.0]	greater than 0 and smaller than or equal to 6500
middle	(6500.0,20000.0]	greater than 6500 and smaller than or equal to 20000
high	> 20000.0	greater than 20000.

The height of a cloud is a real number, as it represents a physical measurement in terms of distance above the Earth's surface.

Write a script called **ex6-clouds.py** that does the following:

Q1. (10) Make a while loop to error-check the user's input. The user should enter a positive number.

Use the input function, and ask the user, for example:
"Enter a positive number for the height of the cloud: "

The while loop should prompt the user over and over until the number entered by the user is valid (i.e. positive, greaten than 0). Your logic should not be redundant.
Assume the user only enters numbers.

Q2. (12) Make an appropriate if statement to classify the height of the cloud (i.e., the valid number entered by the user in Q1). For each class make a print statement. For the low clouds, display to screen *this is a low cloud*, for middle cloud, display *this is a middle cloud* etc.

2. (8) In a script called **ex6-while-dict.py** do the following:
Use one while loop to create a dictionary D of **4 key:value pairs**.
A key should be a random integer number between 1-10 (1 and 10 included), and the corresponding value should be a color name entered by the user.

The colors (the values) in the dictionary D should be unique. Do not make any lists. Your task is to build the dictionary within the for loop. Import the necessary module we did in class.

In the while loop, a dictionary like this should be created:

```
{3: 'yellow', 5: 'green', 1: 'blue', 10: 'red'}
```

Keep in mind that the keys in a dictionary are unique!

Upload your files to Gradescope HW6-part2:
ex6-clouds.py
ex6-while-dict.py

HW6-part3 (35)

1. **(17)** The formula for exponential growth or decay is:

$$A = A_o e^{(k\,t)} \qquad Eq.1$$

where:

- A is the final number of a quantity you're dealing with (e.g., money, bacteria growing in a petri dish, radioactive decay of an element)
- A_0 is the number of that same quantity at time 0
- k is the growth or decay rate
- t is time.

A certain type of bacteria, given a favorable growth medium, doubles in population every 6.5 hours, i.e., the rate is **k = 0.10661/h**.

We will calculate the number of bacteria within the [1,36] hour range by using list comprehension.

In a script called **ex6-bacteria-new.py** do the following:

Q1. (1) Import a module we did in class to calculate the formula.

Q2. (1) Make these variables

```
k=0.1066
Ao=100.0
```

Q3. (4) Make a list of integer numbers called *times* from 1 to 36 (1 and 36 included). Print *times* to screen.

Q4. (8) Use list comprehension to make a new list called *numbac*. The new list *numbac* should contain the number of bacteria for each time in the range 1-36 hours. In list comprehension you should implement the formula in Eq1. Print *numbac* to screen.

Q5. (3) How many bacteria are there after 13 hours? To answer this question, use lists *times* and *numbac,* a method to get the index and use the index to access that value. Do not hardcode the result, and do not use loops. Print the result to screen in scientific notation, with 2 decimal precisions. The output should be:

```
4.00e+02
```

2. (5) In a script called **ex6-str-len.py**, create a list of integers *word_len*, which contains the length of each word in string *sentence*, but only if the word is not the word *the*. Use type conversion and list comprehension to make a list of strings out of *sentence*. If you use loops, you will lose all points for this part. Print *word_len* to screen.

Define and use this variable:

```
sentence = "the quick brown fox jumps over the lazy dog"
```

3. (13) The provided script **ex6-dcompre.py** contains some variables. Edit the script and do the following:

Q1. (6) Sometimes you want to extract specific keys from a dictionary. This is easily accomplished using dictionary comprehension. Make a new dictionary called *Dsel* out of dictionary D containing only the key:value pairs of the *selectedkeys*. Only use dictionary comprehension. Print *Dsel* to screen.

```
D = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F'}
selectedkeys = [0, 2, 5]
```

Q2. (7) Use dictionary comprehension, and ChainA and ChainB:

```
ChainA={'prot1':'VGFTYU','prot2':'NHGTYU','prot3':'UIJYHTUI'}
ChainB={'prot1':'WSEQRT','prot2':'JUHYTR','prot3':'NHYUTO'}
```

to make a new dictionary called ChainAB:

```
{'prot1': 'VGFTYUWSEQRT', 'prot2': 'NHGTYUJUHYTR', 'prot3': 'UIJYHTUINHYUTO'}
```

You should concatenate values having the same key in ChainA and ChainB. You should use a dictionary method within the comprehension. Print *ChainAB t*o screen.

Upload to Gradescope HW8-part2
ex6-bacteria-new.py
ex6-str-len.py
ex6-dcompre.py