## Topics Covered in Beginners:

- print()
- Comments
- Data types (strings, integers, floats)
- Variables
- Addition, subtraction, multiplication, division
- Errors
- Character Art
- Creating a function (with parameters)
- Return values/output
- Conditionals (if, else, pass, elif)
- Comparison operators (>, <, >=, <=, ==, !=)
- Casting variables to a different data type
- Print using escape sequence
- While loops/breaks

## Beginners End of Course Assignment:

**Part 1: Splicing data and calculating PSI**

With RNA-sequencing data, we can look at many different biological outputs: motif analysis to look for enrichment or depletion of particular binding sites, expression data and pathway analysis to see if any particular system seems overwhelmed or activated and splicing analysis to see if there are shifts in the alternative splicing of different groups. We can measure splicing by looking at PSI values (or percent spliced in) of any given exon and then we can compare these PSI values to see if they are different in our groups. For this part of the assignment, we will calculate PSI for two different groups and see how they compare:

1. Create a function called 'calculate_psi' that takes two input parameters, which will be lists of PSI values from two groups
2. Outside of this function define and create two lists:
    a. Controls = [ ]
    b. Affected = [ ]
    c. Fill these lists with several different float values ranging from 0 to 1 (ex: .55, .34, .15, etc). Look at step 5 when thinking about what kind of values you want to fill your lists with.
3. Inside of your function calculate the mean of each list and assign this to a new variable ('control_mean' and 'affected_mean'). You can do this by adding all of the values of your list together and dividing it by the length of your list
    a. Check to make sure all of the values in your lists are floats before performing this operation on them, or else it will not work properly
    b. Print these two values out to check if they appear correct
    c. **For added difficulty:** look up the numpy.mean function. To use this in your script, you just need to 'import numpy' in the beginning to make sure this

package is available to you. Often we import things with a particular shorthand so we can call them later on, for example if I used ‘import numpy as np’ when I call the .mean function later on I would say ‘np.mean()’

4. Next ask if these two values are different from each other using ‘!=’. If they are different return the statement ‘My groups have different splicing for Gene1’. Else, return ‘There is no difference in splicing for Gene1’
5. Go back to the values in your list and change these to test your function and make sure it is working. Try out these scenarios:
    a. Make the values in the first list the exact same as the second list, your function should tell you there is no difference.
    b. Make all of the values in one list extremely high (.91, .89, .99, etc) and the values in the other extremely low (.09, .11, .05, etc), your function should report that they are different
    c. Use completely random values in both lists when you are sure your function is working and see what the result is!
6. **For added difficulty:** you can play around with this aspect of the function and have it only return the first statement if the difference between the two groups meets a particular size requirement first using ‘>’ or ‘>=’.
    a. Subtract the two means and assign this value to a new variable
    b. Take the absolute value of this using abs() and ask if it is larger than a certain threshold. Often in splicing analysis we use a baseline difference of .1 or greater as a first pass value

**Part 2: Finding barcodes within sequence data**

While processing some sequencing data, we may want to look for particular barcodes and collapse these sequences or group them together. For this part of the assignment, we will work on looking for a particular pattern or barcode in our sequences:

1. Create a function called ‘find_barcode’. This function will take two input parameters: a barcode sequence of your choice maybe 5-8 nucleotides long and a list of short sequences
2. Outside of the function, define these two variables:
    a. barcode = (a barcode string) (ex: ‘ATGCA’)
    b. sequences = [ ] (fill list with several short sequence strings, ex: ‘ATGCATGAGTGATC’, ‘CTGCTGCTGCTG’, ‘GCTGAGTCAGTC’, etc). Make sure your list contains at least one or two sequences that start with your barcode, or that end with the reverse complement of your barcode if you are attempting the more difficult version of this assignment
3. Now read in these variables and make sure everything is converted to capital letters using .upper()
    a. We often convert things in the beginning as it is unclear what type of data input you will be working with and you need the pattern to be consistent for sequence comparison

4. Now loop through your list one sequence at a time and ask if it starts with the barcode you input using .startswith() (you may need to google this function first)
    a. To loop through your list, you can use a for loop in the format of ‘for item in list:’ and then perform your function on each ‘item’
5. If a sequence starts with that barcode, append it to a new list
6. Have your function return this list of sequences that contain your barcode
7. **For added difficulty:** try to see if the reverse complement of your barcode is at the end of the sequence instead. For this you will need to:
    a. Import a Biopython module to help you perform this operation:
        i. from Bio.Seq import Seq
    b. Called Seq on your barcode and assign this to a new variable (ex: bc = Seq(barcode))
    c. Now use the .reverse_complement() function to create the reverse complement of your barcode (ex: reverse_bc = bc.reverse_complement())
    d. Look for the reverse complement of your barcode at the end of your sequence using .endswith()
    e. Append these sequences to a different list and return it as well
8. **For added difficulty:** Adjust your function or make a new one, to read in two lists, one of barcodes and one of sequences.
    a. Now loop through each barcode in your barcode list and check every sequence for this barcode before moving onto the next barcode in the list. To do this, you will need to have two nested ‘for’ loops.
    b. For each barcode, return the sequences that contained this barcode.