Module B: Python module

You can choose from the following three tasks:

1. Bioinformatics Tasks *(recommended)*
2. Extra Python Exercises
3. Basic Python Concepts

Whichever task you choose, you'll be building essential Python skills! Happy coding!

---

1. **Bioinformatics Tasks** (recommended)

   These exercises focus on string manipulations in real-life bioinformatics scenarios. You will work with TSV files, extracting necessary data before performing proper analysis. This is a great way to apply Python to practical biological data handling.

2. **Extra Python Exercises** (For those who missed them earlier)

   If you didn't complete the morning exercises, you can do them now. These exercises are more advanced and cover essential Python concepts like:

   ✓*While loops*
   ✓*Working with directories*

   These topics are commonly used in Python coding, making them valuable for improving your skills.

3. **Basic Python Concepts** (For those wanting a simpler start)

   If you need more foundational Python practice, these easier exercises will help reinforce pythonic thinking. You'll work on:

   ✓*User input*
   ✓*Tuples*
   ✓*More list exercises*

   These tasks are designed to build confidence before moving on to more advanced topics.

Bioinformatic Tasks

*Use the **Advanced Python Handout** as a guide for these exercises. It provides key concepts, examples, and best practices to help you complete the tasks efficiently.*

Exercise 1: Unwanted characters

*When working with text files and strings, special characters like **newline (\n)** and **tab (\t)** are always present in the background, even if they aren't visible in a basic text editor. These **escape characters** are crucial for text formatting but can cause unexpected blank lines or spacing issues if not handled properly in scripts.*

Exercise 1A: When printing lines from a file using a loop, each line already contains a hidden newline character (\n) at the end. Since print() adds another newline by default, this results in extra blank lines in the output. For this exercise you will be working with **samplesheet_without_headers.txt** from day2/Samplesheets. Modify the code below to remove the unwanted trailing \n when printing the file content:

```
file = open('/path/to/file/samplesheet_without_headers.txt')
for line in file:
    print(line)
```

Exercise 2: More string manipulations

As a bioinformatician, a common task is to open a file, extract relevant data, and store it in a variable for further analysis. This process is fundamental to bioinformatics workflows and is widely used across different applications.

Exercise 2A: Using Spyder, open **samplesheet_with_headers.txt** from Day2/samplesheets, iterate through every line, add the SampleID (second column) of each line to a list called SampleIDs, and print this list once complete.

Exercise 2B: Modify your code from Exercise 2A to skip the header row before extracting SampleIDs.

Basic Python Concepts

*Use the **Advanced Python Handout** (page 2-8) as a guide for these exercises. It provides key concepts, examples, and best practices to help you complete the tasks efficiently.*

Exercise 3: Tuples, tuples, tuples!

Like lists, tuples store multiple objects in Python. However, unlike lists, tuples are immutable, meaning they cannot be changed after creation.

Exercise 3A: Create a tuple called my_diary that includes these items:
"Pizza", 3.0, 3000, ["pizza shopping list"], "Why is it always food?"

Exercise 3B: Print the fifth element from my_diary.

Exercise 3C: Why would you ever use a tuple instead of a list? Discuss with your partner

Exercise 4: More fun with lists!

Lists are essential in Python for managing and processing data. Their versatility, dynamic nature, and ease of use make them a fundamental part of Python programming.

Exercise 4A: Create a list of pizza toppings:
leos_pizza_toppings = ["pepperoni", "jalapenos", "pepperoni", "tomatoes", "honey"]

Exercise 4B: Use a built-in function to count how many times pepperoni appears in the list.

Exercise 4C: Remove the excessive pepperoni using another built-in function.

Exercise 4D: Change one of the toppings, to something you prefer instead.

Exercise 4E: Create a list of pizza toppings named *my_pizza_toppings*, then combine it with *leos_pizza_toppings* to form a new list called *our_pizza_toppings*.

Exercise 4F: Life is easier when there is structure. Sort our_pizza_toppings in alphabetical order.

Exercise 5: User input

User input allows a program to accept data from users during runtime, enabling interaction through the keyboard or other input devices.

Exercise 5A: Prompt the user to enter their name by asking, "Who is talking to me?"

Exercise 5B: Use the provided name to generate a response and display a message back to the user.

Extra Python Exercises:

Extra Exercises 1: While Loop

Exercise E1A: Create a variable called **sheep** and set it to 0. Using a while loop, print the number of **sheep** and increase the amount of **sheep** by 1 until you reach 23.

Exercise E1B: Copy and paste the duck list below into Spyder:

```
duck_list=["duck", "duck", "duck", "duck", "duck", "duck", "duck", "goose", "duck", "duck", "duck"]
```

Exercise E1C: Using a *for-loop* to iterate through duck_list, count the number of **ducks** and stop when you reach **goose**. Print "Goose!" along with the number of ducks counted before the goose.

Exercise E1D: Repeat exercise b, but use a *while-loop* instead of a for loop.

Tip: use len()

Extra Exercises 2: Making a function

Exercise E2A: Modify the previous if-statement from Exercise 4A by turning it into a function called higher_than(X). The function should take an input number X and return one of the following messages:

"X is lower than 68"
"X is higher than 68"
"both numbers are 68"

```
# Example result from Exercise 4A:
num = 70 # You can change this number to test different values

# If statement to check if the number is greater than 68
if num > 68:
    print(num, "is greater than 68")
```

Exercise E2B: Improve the function and make it take two inputs (integer or float) so that running higher_than(X,Y) will return one of the following:

"X is not higher than Y"
"X is higher than Y"
"Both numbers are X" *(if X and Y are the same)*

Extra Exercises 3: Sets

Exercise E3A: Convert *duck_list* to a **set** and observe the difference.

Extra Exercises 4: Dictionaries

Exercise E4A: Create a dictionary called *favorite_foods* with your top three favorite foods as keys and their corresponding deliciousness levels (on a scale of 1-10) as values. Print the dictionary to see your favorite foods and how delicious they are.

Exercise E4B: Choose one food and print its deliciousness level along with an explanatory message by retrieving the information from the dictionary.
Example: The deliciousness level of broccoli is 10

Exercise E4C: Modify the deliciousness level to make it even more delicious and make a new explanatory message.
Example: Actually, the deliciousness level of broccoli is more like 11

Exercise E4D: Add your least favorite food to the dictionary along with its corresponding deliciousness levels and update the explanatory message.
Example: I don't care for pizza; I only find its deliciousness level to be 3

Exercise E4E: Make a dictionary called *my_dict* with a key called **number** and a key called **food** that each have an empty list as value.

Exercise E4F: Make a *for-loop* that iterates through the *burger_list* created in Exercise 3A and append all string values (burger ingredients) to the **food**-key and all numerical values to the **number-**key.

```
# Example:
burger_list = ["lettuce", "cheese", "tomato", "bacon", "onion", 7, 3.14, 42, 10, 99.9]
```