Workshop 2: Loops, a brief introduction to AWK, and launching programs and scripts from the terminal (e.g. R, Python, BLAST).

Part I: Loops, statements and bash scripts

If you have a look at past workshops on R and Python, you should remember the basic structure of for loops and if..else statements.

In UNIX terminals (the shell), the for loop is slightly different than in R or Python. For example, try this:

```
for i in {1..10}
do
echo $i
echo "file_number_"$i
echo "file_number_"$i".txt"
done
```

If..else statements are quite convoluted. Here is a short script to test whether a variable is equal, larger, or smaller than another variable. Try it, and alter it to see what happens.

```
VAR1=100
VAR2=20

if [[ $VAR1 -gt $VAR2 ]];
then   echo "VAR1 is greater than VAR2.";

elif [[ $VAR1 == $VAR2 ]];
then   echo "VAR1 is equal to VAR2.";
else   echo "VAR1 is less than VAR2.";

fi
```

You can find below a list of common operators that you can use to perform tests using the UNIX command line:

- `-n VAR` - True if the length of `VAR` is greater than zero.
- `-z VAR` - True if the `VAR` is empty.
- `STRING1 = STRING2` - True if `STRING1` and `STRING2` are equal.
- `STRING1 != STRING2` - True if `STRING1` and `STRING2` are not equal.

- `INTEGER1 -eq INTEGER2` - True if `INTEGER1` and `INTEGER2` are equal.
- `INTEGER1 -gt INTEGER2` - True if `INTEGER1` is greater than `INTEGER2`.
- `INTEGER1 -lt INTEGER2` - True if `INTEGER1` is less than `INTEGER2`.
- `INTEGER1 -ge INTEGER2` - True if `INTEGER1` is equal or greater than INTEGER2.
- `INTEGER1 -le INTEGER2` - True if `INTEGER1` is equal or less than `INTEGER2`.
- `-f FILE` - True is the FILE exists and is a regular file (not a directory or device).

You can have a look here: https://tldp.org/LDP/abs/html/comparison-ops.html for a complete list of comparison operators.
Needless to say how convoluted this is, and we rarely create very complex loops and statements using Terminal scripts.

You can use the read command to interactively inject a value into a variable. For example:

```
read VAR1
```

Type a number, then:

```
echo $VAR1
```

Note that we can run bash scripts using the command bash, followed by the name of the script that contains our instructions. Use nano (or vim) to create a new file named script_bash.sh that will contain the following commands:

```
#!/bin/bash

echo -n "Enter the first number: "
read VAR1
echo -n "Enter the second number: "
read VAR2
echo -n "Enter the third number: "
read VAR3

if [[ $VAR1 -ge $VAR2 ]] && [[ $VAR1 -ge $VAR3 ]]
then
  echo "$VAR1 is the largest number."
elif [[ $VAR2 -ge $VAR1 ]] && [[ $VAR2 -ge $VAR3 ]]
then
  echo "$VAR2 is the largest number."
else
  echo "$VAR3 is the largest number."
fi
```

Then type

```
bash script_bash.sh
```

Follow the instructions on screen. See what happens?

Based on this and what you learned in the previous workshop:

Task 1: Create a script that creates ten folders named folder_1, folder_2, folder_3, etc.

Task 2: Create a bash script that asks for three folders, and creates text files named name.txt within the folders that contain the folder's name.

You can of course play more with these scripts.

Part II: Introduction to basic operations in UNIX and AWK

We can perform operations using the UNIX shell and attribute the result to a variable. For example, low at the following script.

```
((val= 2 + 2))
echo "Total value : $val"
```

You can try a few more operations and use variable names (make them start with a $ sign):

```
a=10
b=10
((val= $a * $b))
echo $val
((val= $a / $b))
echo $val
((val= $a + $b))
echo $val
((val= $a - $b))
echo $val
```

For operations on files (for example, to quickly compute the average of a given column), we can use UNIX, but it is not very straightforward.

Let's consider a file containing the following columns that we will name example.txt:

```
a	1
b	3
c	2
d	2
```

```
total=0

while read -r line; do
    for val in $line; do
      ((total+=val))
   done
done < example.txt
echo $total
```

Note that we introduce the while loop here: while there is something to read in file.txt, the script will add all numerical values found in each line to total. This script will give the sum of all numerical values in the file. Do we need something so complicated? It is unpractical, and prone to errors.

AWK is quite useful to manipulate large text files in the UNIX command line. It is another sort of language (a bit like R or Python).

You can extract rows based on the value of one column. For example, to extract only rows for which the value in the second column is strictly higher than 1, we can type:

```
cat file.txt | awk '$2>1'
```

We can also output columns in a different order, for example:

```
awk '{print $2 "\t" $1}' example.txt
```

Note that « \t » here stands for tabulation, which is the delimiter between columns.

At last, we can use AWK to compute the average of a specific column. For example, let's have a look at this command:

```
awk '{ sum += $2; n++ } END { if (n > 0) print sum / n; }'
example.txt
```

To learn more about AWK, you can consult this very detailed tutorial:

https://www.tutorialspoint.com/awk/awk_basic_syntax.htm

Task: For this task, you will use the split –l command. It takes as an argument a text file, and will divide it into several files with the same number of rows.

```
split –l10 file_for_average.txt file.txt
```

You will use the file file_for_average.txt that comes with this workshop. It contains 100 rows Create a bash script that outputs the average of the fifth column for each of the 10 files (which should be named file.txtaa, file.txtab, etc.) Note that you can include a awk command in your bash loop.

Part III: Launch programs from the terminal: an example with BLAST.

This is probably one of the most important part of these workshops. Most bioinformatician do not use NCBI BLAST when they want to find matches between a sequence of interest and a genome. They will run BLAST locally on their own machine or High Performance Cluster. This is not worth only for BLAST. From the terminal, you can actually launch all the most common programming languages used in bioinformatics (assuming that they have already been installed).

We will try to identify the position of two sequences on chromosome 1 of the human genome. One is an amino-acid sequence for the gene (Gene_amino_acid_seq_to_id.fasta), the other contains the complete gene sequence (including introns, found in Gene_DNA_seq_to_id.fasta)

In the table below you will find a list of commands that can be used to perform a local blast alignment. Note that you find the same descriptions on the NCBI BLAST website: https://blast.ncbi.nlm.nih.gov/Blast.cgi

Query Type	Database Type	BLAST Program
Nucleotide	Nucleotide	**blastn:** Search a nucleotide database using a nucleotide query.
Nucleotide	Protein	**blastx:** Search protein database using a translated nucleotide query.
Protein	Nucleotide	**tblastn:** Search translated nucleotide database using a protein query.
Protein	Protein	**blastp:** Search protein database using a protein query.

We have a reference in Reference_human_genome_chr1.fasta, which covers positions 11,000,000 to 12,000,000 of the human genome (version GRCh38).

The first step consists in creating a database that the program can access. The command for this is makeblastdb:

```
makeblastdb
makeblastdb –h

makeblastdb -in Reference_human_genome_chr1.fasta -dbtype nucl -
input_type fasta -out Reference
```

```
blastn -db Reference -query Gene_DNA_seq_to_id.fasta -outfmt 7 -
out blastn_output.txt

tblastn -db Reference -query Gene_amino_acid_seq_to_id.fasta -
outfmt 7 -out tblasn_output.txt
```

Task: Inspect the two output files. Based on your results, can you identify the position of the gene on the human genome? Can you make a hypothesis about the position of exons? Is it consistent with what you find when browsing ENSEMBL? Try blasting the two sequences on NCBI BLAST. What do you notice?

Task: Navigate the help menu for the two programs. Explore what happens when changing the -outfmt option. Create a script that extracts the e-value, the start and end positions of the query.
Task: Use AWK to filter rows that have an e-value and percentage of identity higher than a threshold that you will specify.