Day 4: Functions and Input/Output practicals

Let's start by importing from the **data** module:

In [1]:
```
from data import blast_out,human_sequence,IUPAC_codes
```

Problem 1:

The ***human_sequence*** variable imported from the **data** module is a cDNA sequence string resembling a typical aligned sequence, with gaps spread throughout.

a) Define a function capable of taking any sequence string as input, and that prints a new sequence without gaps. The function must be able to take the gap symbol as an argument (e.g., "-" or "?", etc)

In [ ]:
```

```

b) Define a function similar to the one created for a), but capable of printing a new sequence without gaps entirely OR without gaps only at the 3' and 5' extremities of the sequence.

Tips:

- You can use a boolean-like variable as an argument of the function, e.g. "zero_gaps = True" for when you want the function to remove gaps from both ends, and "zero_gaps = Right/Left" when, you want to remove them from only one side).

In [ ]:
```

```

c) Define a function capable of finding any motif in a sequence, and returning a tuple with the starting and ending positions of its first appearance. If the motif does not exist, print an error message. Find the following motifs in the ***human_sequence*** variable:

- "GGGTTCACTT"
- "GATCA"
- "AACAT"
- "GGTGTGGGGGG"

Tips:

- The function can be defined with two arguments: One for the sequence, and another for the motif

In [ ]:
```

```

d) Modify the function created in c), so that it can take a variable number of motifs as arguments and return a dictionary with the motifs as *keys* and the tuple with the positions as *values*.

(This is a tough one)

In [ ]:
```

```

Problem 2

The list blast_out from the examples module contains a short table of blast results in a simplified tabular format with the following fields per hit:

query sequence \t ***subject sequence*** \t ***identification percentage*** \t ***e-value***

a) Define a function that returns a list containing only the blast hits with an e-value below 1×10^{-5} . This e-value cut-off should be the default, but the function must be flexible enough to be called with different cut-off values.

Tips:

- Python is able to recognize and interpret numbers in scientific notation when converted with float(), e.g. float(1e-7);
- Define the function using two arguments: one for a list, and the other for the e-value number;
- You can use the .split() method of strings to separated the different field in each blast hit)

In [ ]:

b) Based on the list returned by the function in a), define a function that sorts the blast hits according to their identification percentage into three lists of high (100-90%), moderate (90-60%) and low (below 60%) identity percentage. Return those lists as separate variables and compute the number of hits in each identity percentage class.

Tips:

- Once again you can use the .split() method of strings, but now we are interested in the "identification percentage" field, instead of the "e-value" field);
- Remember that functions can return multiple values (separated by commas)

In [ ]:

c) Modify the function in a) so that it can take a file object as an argument, instead of a list.

In [ ]:

c) Open the "blast_out.txt" file in read mode and use the previous function to return a list of blast hits with e-values below 1×10^{-15} .

In [ ]:

d) Create a script that includes the function defined in a). When executed through the terminal, the script should prompt the user for:

- the path and filename of the blast output file that is going to be read
- the desired e-value cutoff
- the name of the output filename

The script must open and read the provided input file, and write all blast hits above the desired e-value cutoff to a new file with the name provided by the user.

Tips:

- Use the raw_input() and input() functions to collect information from the user/keyboard - Their values can be stored in variables to be used latter;

In [ ]:

Problem 3:

Use python to open the "My_fasta.fas" file in read mode.

a) Define a function that returns a dictionary with the fasta headers as *keys* and their sequence as the corresponding *value*.

Tips:

- For this exercise, we ran out of tips. Sorry.

In [ ]:

b) Define a function that performs some quality checks for each sequence. Check if:

- all sequences are of the same size. If not, the function should print a message informing which taxa have sequences of different length.
- there are no illegal characters in each sequence. If there are, the function should print a message informing which taxon's sequence has problems and what is the illegal character. Use the *IUPAC_codes* list from the *data* module to check for illegal characters.

In [ ]:

c) Create a function that uses the dictionary created in a) and collapses taxa with identical sequences into the same haplotype. Write these unique haplotypes to a new file (My_fasta_collapsed.fas) in fasta format, and write the correspondance between haplotype and taxon name in

another file (My_haplotype_list.txt).

In [ ]: