Introducción a Biopython

El proyecto de Biopython es una asociación internacional de desarrolladores del lenguaje Python con aplicación a biología molecular computacional. Es de libre acceso y está muy bien documentado, para mayor profundidad consulte el Recetario de Biopython. Para la instalación y requisitos consultar
biopython/biopython

```
module load python/3.6.0_miniconda-4.3.11_gcc-11.2.0
conda create -n biopython
conda activate biopython
conda install -c conda-forge biopython
```

```
  Cell In[1], line 1
    module load python/3.6.0_miniconda-4.3.11_gcc-11.2.0
           ^
SyntaxError: invalid syntax
```

Debemos salir y volver a entrar a apolo para chequear la instalación. Una realizamos el login, comenzamos python y evocamos biopython

```
python
```

```
import Bio
print(Bio.__version__)
```

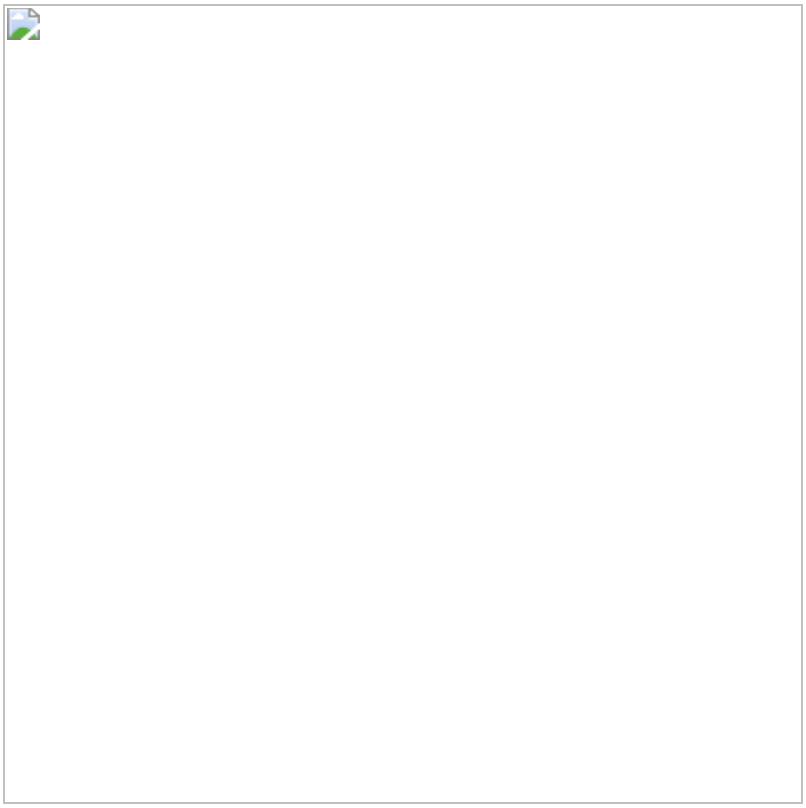
Si sale un error después de llamar la librería es porque hay un error de instalación. De lo contrario, está bien por el momento y empezamos a trabajar con secuencias.

Secuencias

Aunque una secuencia es un conjunto de letras, biopython tiene un formato especial para definir una secuencia

```
from Bio.Seq import Seq
my_seq = Seq("AGTACACTGGT")
my_seq
my_seq.complement()
my_seq.reverse_complement()
```

Para entender por qué es importante el complemento y el complemento reverso, recordemos brevemente cómo es la síntesis de proteínas



Vamos a utilizar el archivo descargado previamente "Sars_cov.dna.fa". Para esto vamos a entrar al directorio donde está almacenado en el PC, y desde allí vamos a realizar *save copy* "scp" del archivo a nuestro directorio en el *home* de Apolo

```
scp Sars_cov.dna.fa lsalazarj@apolo.eafit.edu.co:/home/lsalazarj
```

```
from Bio import SeqIO

dna_file = "Sars_cov.dna.fa"

for record in SeqIO.parse(dna_file, "fasta"):
    print(record)
```

Skip to main content

La función SeqIO.parse abre el archivo **input_file** y lo intepreta como un archivo fasta. Como solo hay una secuencia, el for loop solo itera una vez y muestra ("print") los atributos de la secuencia. Compárelo con el archivo fasta para entender que significa cada línea.

Ahora vamos a extraer la secuencia y calcular su tamaño

```
for record in SeqIO.parse(dna_file, "fasta"):
    print(record.seq)
    print(len(record.seq))
```

Nota: La indentación (la sangría jerárquica luego de una declaración) es muy importante en Python. Si no se respeta, el programa arroja un error

De manera similar se hace con un archivo de múltiples secuencias, por ejemplo el archivo de proteínas. Repita la copia de archivo como se hizo anteriormente, con el de las proteínas. Ahora vamos a iterar sobre las secuencias del archivo "Sars_cov.pep.fa" y a desplegar los identificadores

```
prot_file = "Sars_cov.prot.fa"

for record in SeqIO.parse(prot_file, "fasta"):
    print(record.id)
```

En este archivo hay 12 proteínas de diferentes tamaños y con funciones distintas. Vamos a imprimir su id, su tamaño y su descripción:

```
for record in SeqIO.parse(prot_file, "fasta"):
    print(record.id,len(record.seq),record.description)
```

La descripción tiene mucha información importante pero esta no es muy clara, entonces vamos a extraer el fragmento que contiene la función, vamos a dividir la descripción en items, utilizando la función `split.()` y luego le damos el número (**index**) del item entre de interés `[]` (es el item 9 pero como python comienza a contar desde 0 entonces es el 8)

```
for record in SeqIO.parse(prot_file, "fasta"):
    print(record.id,len(record.seq),record.description.split()[8])
```

Skip to main content

Escribir secuencias un archivo

Con frecuencia estamos interesados en un subconjunto de los genes o proteínas para realizarun análisis a profundidad de ellos. En este caso vamos a seleccionar las glicoproteínas que son aquellas que codifican la espícula. Esta espícula ("spike") es la encargada de mediar la entrada del virus hacia el hospedero y tiene una tasa de mutación mas alta que los otros genes, lo que presumiblemente dió origen a cepas mas virulentas (https://en.wikipedia.org/wiki/Coronavirus_spike_protein)

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord

for record in SeqIO.parse(prot_file, "fasta"):
    if (record.description.split()[9] == "glycoprotein"):
        print(record)
```

Hasta aquí hemos seleccionado las secuencias que contienen el término "glycoprotein". Para escribirlas en un archivo en un formato adecuado de secuencias vamos a utilizar la función de biopython `Bio.SeqIO.write()` y a cambiar un poco el estilo.

```
glycoproteins = (record for record in SeqIO.parse(prot_file, "fasta") if (record.descri
SeqIO.write(glycoproteins, "sars_covid19_glycoproteins.fa", "fasta")
```

Donde eperas que esté el archivo? Chequéalo mirando el directorio actual ("get current working directory"). Verifica que si se haya generado con el nombre y formatos correctos, y la información que deseamos que contenga.

```
import os

os. getcwd()
os.listdir()
```

Por último, vamos a escribir los comando que utilizamos en un script de python (con extensión ".py") para dejar registro de lo que hicimos y pueda ser repetible

```
#!/usr/local/bin/

# Script por Laura Salazar Jaramillo
# Genera un subconjunto de glicoproteinas de SARS-covid 19

# Importamos las librerías necesarias
import os
import sys

import Bio
from Bio import SeqIO
from Bio.Seq import Seq

# Archivos de entrada (inputs)
dna_file = "Sars_cov.dna.fa"
prot_file = "Sars_cov.prot.fa"


# Listamos el tamano del genoma
for record in SeqIO.parse(dna_file, "fasta"):
    print("La longitud en nc del genoma es:",len(record.seq))

# Listamos los id y longitudes de las proteínas
for record in SeqIO.parse(prot_file, "fasta"):
    print(record.id,len(record.seq),record.description.split()[9])

# Seleccionamos las glicoproteínas y las guardamos en un archivo aparte
glycoproteins = (record for record in SeqIO.parse(prot_file, "fasta") if (record.descri
SeqIO.write(glycoproteins, "sars_covid19_glycoproteins.fa", "fasta")
```

Este script se puede ejecutar en la terminal escribiendo los comandos

```
python glycoproteins.py > glycoproteins.out
```

Ejercicio Evaluable 1) Ejecute en Apolo el script de python via Slurm 2) Suba el script de python (arch.py), el de slurm y el output (arch.out) por la asignación de tareas