ACCESS BIOINFORMATICS DATABASES WITH BIO-PYTHON

This project is aimed to deploy python-based programming pipelines and scripts to automate biological data retrieval and analysis.

1. NCBI

This section uses the NCBI module from Biopyton to perform BLAST (Basic Local Alignment Search Tool). This finds regions of local similarity between sequences from the NCBI databse. It is a preliminary analysis in identifying the strain, species, or the source, to which our isolated sequence belongs to or is closely related.

Import Modules

In [19]:
```
from Bio.Blast import NCBIWWW
from Bio import SeqIO, SearchIO
```

1.1. Nucleotide BLAST

First, I performed nucleotide BLAST using the nucleotide sequence stored in the nuc_seq.fasta file. As shown, the sequence is 774 base pairs long:

In [20]:
```
nuc_record = SeqIO.read("nuc_seq.fasta", format = "fasta")
len(nuc_record)
```

Out[20]:

774

We can access various parts of the sequences file. The file is composed of two lines: the description and the sequence.

1. The description:

```
In [21]: nuc_record.description
```

```
Out[21]:
'MT598137.1 Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/h
uman/IRN/PN-2142-S/2020 surface glycoprotein (S) gene, partial cds'
```

Our sequence is a gene encoding surface glycoprotein SARS-CoV-2 isolate.

2. The sequence:

In [22]:
```
nuc_record.seq
```

Out[22]:

```
Seq('ATCGCTCCAGGGCAAACTGGAAAGATTGCTGATTATAATTATAAATTACCAGAT...GGT')
```

Now we perform blast!

The module takes in three parameters:

1. 'blastn'which stands for BLAST nucleotide.
2. The database used it 'nt' which stands for nucleotide.
3. 'nuc_record.seq' which is our record.

The SEarchIO Biopython module is used to read in the results of our BLAST.

```
In [23]: result_handle = NCBIWWW.qblast("blastn","nt",nuc_record.seq)
         blast_result = SearchIO.read(result_handle,"blast-xml")
```

```
C:\Users\fxy40\anaconda3\lib\site-packages\Bio\SearchIO\_legacy\__init__.py:12:
BiopythonDeprecationWarning: The 'Bio.SearchIO._legacy' module for parsing BLAST
plain text output is deprecated and will be removed in a future release of Biopy
thon. Consider generating your BLAST output for parsing as XML or tabular format
instead.
  warnings.warn(
```

I used the Python indexing technique to print out the first two results:

```
In [24]: print(blast_result[0:2])
```

```
Program: blastn (2.14.1+)
  Query: No (774)
         definition line
 Target: nt
   Hits: ----  -----  ------------------------------------------------------------
            #  # HSP  ID + description
         ----  -----  ------------------------------------------------------------
            0      1  gi|2529195153|gb|OR223350.1|  Severe acute respiratory ...
            1      1  gi|2529195140|gb|OR223349.1|  Severe acute respiratory ...
```

Usually, the first hit the the BLAST constitutes the best match. Hence, we will fetch more details about the first sequence from the BLAST results.

```
In [25]: Seq = blast_result[0]
         print(f"Sequence ID: {Seq.id}")
print(f"Sequence Description: {Seq.description}")

details = Seq[0]
print(f"E-value: {details.evalue}")
```

```
Sequence ID: gi|2529195153|gb|OR223350.1|
Sequence Description: Severe acute respiratory syndrome coronavirus 2 isolate SA
RS-CoV-2/human/USA/WA-UW143/2020 ORF1ab polyprotein (ORF1ab), ORF1a polyprotein
(ORF1ab), surface glycoprotein (S), ORF3a protein (ORF3a), envelope protein (E),
membrane glycoprotein (M), ORF6 protein (ORF6), ORF7a protein (ORF7a), ORF7b (OR
F7b), ORF8 protein (ORF8), nucleocapsid phosphoprotein (N), and ORF10 protein (O
RF10) genes, complete cds
E-value: 0.0
```

A low value for a sequence of considerable length is considered as optimal. Here, the E-valye is 0.0. Therefore, the sequence is an exct or a very closely related match to our sequence.

```
In [26]: print(f"alignment:\n{details.aln}")
```

```
alignment:
Alignment with 2 rows and 774 columns
ATCGCTCCAGGGCAAACTGGAAAGATTGCTGATTATAATTATAA...GGT No
ATCGCTCCAGGGCAAACTGGAAAGATTGCTGATTATAATTATAA...GGT gi|2529195153|gb|OR223350.1|
```

This gives us the alignment of the two sequences, with the top one being the query sequence which we used, and the bottom the result which we got.

1.2. Protein BLAST

In [27]:
```
prot_record = SeqIO.read("prot_seq.fasta", format="fasta")
len(prot_record)
```

Out[27]:

258

Length of the protein sequence: 258 amino acids.
This module also takes three parameters:

1. 'blastp' stands for BLAST protein.
2. The database used it PDB.
3. 'prot_record.seq' is our protein record.

In [28]:
```
result_handle = NCBIWWW.qblast("blastp", "pdb", prot_record.seq)
blast_result = SearchIO.read(result_handle, "blast-xml")
```

The first two results are fetched by running the next two cells:

```
In [29]: print(blast_result[0:2])
```

```
Program: blastp (2.14.1+)
  Query: unnamed (258)
         protein product
 Target: pdb
   Hits: ----  -----  ------------------------------------------------------------
            #  # HSP  ID + description
         ----  -----  ------------------------------------------------------------
            0      1  pdb|7CAB|A  Chain A, Spike glycoprotein [Severe acute r...
            1      1  pdb|7R4I|A  Chain A, Spike glycoprotein [Severe acute r...
```

Our protein sequence corresponds to two matches which have the greatest similarity, among which we fetch the first result:

```
In [30]: Seq = blast_result [0]
         print(f"Sequence ID: {Seq.id}")
print(f"Sequence Description: {Seq.description}")

details = Seq[0]
print(f"E-value: {details.evalue}")
```

```
Sequence ID: pdb|7CAB|A
Sequence Description: Chain A, Spike glycoprotein [Severe acute respiratory synd
rome coronavirus 2]
E-value: 0.0
```

To check for alignment, I ran this cell which returned the alignment of our sequence against the results fetched using BLAST.

```
In [31]: print(f"alignment:\n {details.aln}")
```

```
alignment:
 Alignment with 2 rows and 258 columns
IAPGQTGKIADYNYKLPDDFTGCVIAWNSNNLDSKVGGNYNYLY...PIG unnamed
IAPGQTGKIADYNYKLPDDFTGCVIAWNSNNLDSKVGGNYNYLY...PIG pdb|7CAB|A
```