Biopython lesson 1

pip install biopython

```
pip install biopython
```

```
Requirement already satisfied: biopython in c:\users\acer\anaconda3\lib\site-packages (1.85)Note: you may need to restart the kernel to use updated packa
ges.

Requirement already satisfied: numpy in c:\users\acer\anaconda3\lib\site-packages (from biopython) (1.26.4)
```

We're looking at which version it is.

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

```
1.85
```

```
from Bio.Seq import Seq
my_Seq=Seq("AGTACACTGGT")
print(my_Seq)
```

```
AGTACACTGGT
```

In this code, **Bio.Seq** and **Seq** are part of the Biopython library.

1.Bio.Seq – Used to call the **Seq** module of the Biopython library. This module can be used to sequence nucleotide or amino acid sequences.

allows you to process sequences.

2.Seq – The **Seq** class in Biopython . This class represents DNA, RNA, or protein sequences and performs various operations on them.

allows you to perform operations (e.g., finding a complementary sequence, inverting, etc.).

3.my_Seq = Seq("AGTACACTGGT") – Here a variable called **my_Seq** is created and the nucleotide sequence **"AGTACACTGGT" is** assigned to it.

This sequence is converted to a **Seq** object, which means it is now a special object that can be manipulated by Biopython's functions.

4.print(my_Seq) – This prints the contents of **the my_Seq** variable to the screen, meaning you get the result **AGTACACTGGT .**

That is, **Seq** converts DNA/RNA/protein sequences that appear as plain text into a special object for processing with Biopython functions.

This object can perform various operations such as finding the complementary strand of DNA, transcribing, and reversing it.

It is possible to carry out.

```
my_Seq.complement()
```

```
Seq('TCATGTGACCA')
```

In the code, **the my_Seq.complement()** function is a method of the **Seq** object of the **Biopython** library and **returns the complement** of a DNA sequence.

Returns **the (complementary) chain .**

How does it work?

In DNA, each nucleotide pairs with the complementary base opposite it:

•A (Adenine) ÿ T (Thymine)

•T (Thymine) ÿ A (Adenine)

•C (Cytosine) ÿ G (Guanine)

•G (Guanine) ÿ C (Cytosine)

The working principle of the code is as follows:

If **my_Seq** was initially defined like this:

from Bio.Seq import Seq my_Seq = Seq("AGTACACTGGT") print(my_Seq.complement())

Here, when **the my_Seq.complement()** function is called, each nucleotide is replaced with its corresponding complement:

Result:

The complementary sequence printed on the screen:

Seq('TCATGTGACCA')

As a result, the my_Seq.complement() method returns the complementary strand of DNA. This method is used in bioinformatics, especially in genetics.

It is often used in assays and PCR design.

Home chain (my_Seq)	A	G	T	A	C	A	C	T	G	G	T
Complete chainsaw	T	C	A	T	G	T	G	A	C	C	A

```
my_Seq.reverse_complement()
```

```
Seq('ACCAGTGTACT')
```

What Does the my_Seq.reverse_complement() **Method Do?**

my_Seq.reverse_complement() is a method of **the Seq** object **in Biopython .** This method **returns the complement** of a DNA sequence.

Returns **the complementary and reverse chain .**

What is the
difference? •complement() ÿ Returns **the complementary strand** of a
DNA sequence. •reverse_complement() ÿ First finds **the complementary strand ,** then **reverses it.**

Code Execution Process

1.Basic sequence: "AGTACACTGGT"

2.complement() is applied ÿ "TCATGTGACCA"

3. Reverse ÿ "ACCAGTGTACT"

4. The final result is printed:

Seq('ACCAGTGTACT')

[241]:
```python
from Bio import SeqIO
file_path = r"C:\Users\Acer\Downloads\ls_orchid.fasta"  # Fayl adını düzgün yaz
for seq_record in SeqIO.parse(file_path, "fasta"):
    print("ID:", seq_record.id)
    print("Sequence Length:", len(seq_record))
    print("Sequence:", repr(seq_record.seq))  # Nukleotid ardıcıllığını çap edir
```

```
ID: gi|2765658|emb|Z78533.1|CIZ78533
Sequence Length: 740
Sequence: Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC')
ID: gi|2765657|emb|Z78532.1|CCZ78532
Sequence Length: 753
Sequence: Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAG...GGC')
ID: gi|2765656|emb|Z78531.1|CFZ78531
Sequence Length: 748
Sequence: Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGCAG...TAA')
ID: gi|2765655|emb|Z78530.1|CMZ78530
Sequence Length: 744
Sequence: Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAAACAACAT...CAT')
ID: gi|2765654|emb|Z78529.1|CLZ78529
Sequence Length: 733
Sequence: Seq('ACGGCGAGCTGCCGAAGGACATTGTTGAGACAGCAGAATATACGATTGAGTGAA...AAA')
ID: gi|2765652|emb|Z78527.1|CYZ78527
Sequence Length: 718
Sequence: Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGTAG...CCC')
```

This **Python** code reads a **FASTA-formatted** bioinformatics sequencing file using the **SeqIO** module of **the Biopython** library and outputs the following information: **1. The ID number of the sequence (DNA or protein sequence)**
2. The length of the sequence (number of nucleotides or amino acids)
3. The sequence itself

1. SeqIO ÿ This is a module used to read and write **sequencing files** from the Biopython library.

2. file_path = r"C:\Users\Acer\Downloads\ls_orchid.fasta"

- file_path ÿ Indicates the full path of the .fasta format file containing sequence data.
- r" " **(raw string)** ÿ Used to prevent \ characters from being treated as special characters.

3. Reads Sequence File

for seq_record in SeqIO.parse(file_path, "fasta"):

- SeqIO.parse(file_path, "fasta") ÿ Reads the given file **in FASTA format .**
- for seq_record in ... ÿ It processes **each sequence (record)** in the file one by one.

4. Printing Sequence Data

print("ID:", seq_record.id)
- seq_record.id ÿ Extracts the sequence's **FASTA header identifier (ID) .**

print("Sequence Length:", len(seq_record))
len(seq_record) ÿ Returns the length of the sequence (number of nucleotides or amino acids).

print("Sequence:", repr(seq_record.seq))
seq_record.seq ÿ Prints the sequence sequence to the screen.
repr() ÿ Used to display the sequence in full instead of dots ("...").

Code Execution Result (Sample Output)

Suppose a sequence in the ls_orchid.fasta file is as follows:

>gi|2765658|emb|AJ001588.1| Orchidaceae rbcL gene ATGTCACGCTTACCGTGGG
The following code will be displayed:

ID: gi|2765658|emb|AJ001588.1| Sequence Length: 19 Sequence: 'ATGTCACGCTTACCGTGGG'

Bioinformatics Use of Code

ÿ **Read genomic or protein sequences in FASTA format**
ÿ **Obtaining sequencing data for genetic analyses**
ÿ **Checking the length of DNA or protein sequences**
ÿ **Automation of sequence processing and analysis processes**

Note:

•**FASTA** format is widely used in bioinformatics to store **DNA, RNA or protein sequences .** •SeqIO.parse() function is used **for reading** only , if the sequence is to be modified, it must be rewritten with SeqIO.write() function.

[235]:
```python
from Bio import SeqIO

file_path = r"C:\Users\Acer\Downloads\ls_orchid.gbk"  # Düzgün fayl yolu

for seq_record in SeqIO.parse(file_path, "genbank"):  # "gbk" əvəzinə "genbank" yazdıq
    print("ID:", seq_record.id)
    print("Sequence Length:", len(seq_record))
    print("Sequence:", repr(seq_record.seq))
```

```
ID: Z78533.1
Sequence Length: 740
Sequence: Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC')
ID: Z78532.1
Sequence Length: 753
Sequence: Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAG...GGC')
ID: Z78531.1
Sequence Length: 748
Sequence: Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGCAG...TAA')
ID: Z78530.1
Sequence Length: 744
Sequence: Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAAACAACAT...CAT')
ID: Z78529.1
Sequence Length: 733
Sequence: Seq('ACGGCGAGCTGCCGAAGGACATTGTTGAGACAGCAGAATATACGATTGAGTGAA...AAA')
ID: Z78527.1
Sequence Length: 718
Sequence: Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGTAG...CCC')
```

Code

Explanation This **Python** code reads a **GenBank format** (.gbk) sequence file using the **SeqIO** module of **the Biopython** library and outputs the following information: **1. Sequence ID number (DNA or protein sequence) 2. Sequence length (number of nucleotides or amino acids) 3. The sequence itself**

As a result of running the code, the following will appear on the screen:

ID: AJ001588 Sequence Length: 19 Sequence: 'ATGTCACGCTTACCGTGGG'

Bioinformatics Use of Code

- ÿ **Read genomic or protein sequences in GenBank format**
- ÿ **Obtaining sequencing data for genetic analyses**
- ÿ **Automate sequence processing and annotation processes**
- ÿ **Get additional metadata (gene name, features, annotations, etc.) from a GenBank file**

Differences between GenBank and FASTA

Feature	GenBank (.gbk)	FASTA (.fasta)
Data type	Genetic sequencing + annotation	Sequencing only
Format	Multi-line, structured	Simple and compact
Supported data	Sequence, gene annotation, properties, source	Sequence and title (ID)
To read in Biopython	"genebank"	"fast"

This code **is used for GenBank files instead of the FASTA format** and allows for processing additional annotation data.

creates an opportunity.

[193]:
```
from Bio.Seq import Seq
my_seq = Seq("GATCG")
for index, letter in enumerate(my_seq):
    print(index, letter)
print(len(my_seq))
```

```
0 G
1 A
2 T
3 C
4 G
5
```

Building a Loop on Nucleotides

for index, letter in enumerate(my_seq): print(index, letter)

•enumerate(my_seq) ÿ Retrieves each nucleotide of the sequence along with its index.

•for index, letter in ... ÿ Cycle **returns an index and a letter for each nucleotide.**

•print(index, letter) ÿ Prints each nucleotide and its index to the screen.

•**The first five lines** print **each nucleotide** in the sequence and its index number.

•**The last line** displays the total length of the sequence (5).

Bioinformatics Use of Code

- **Processing individual nucleotides and amino acids in DNA/RNA/protein sequences**
- **Analyze the length and composition of the sequence**
- **Perform iteration operations in bioinformatics algorithms**
- **Use the same methods as Python strings when working with sequences**

This code **is a simple example for analyzing DNA sequences** and can be used for more advanced

can be the basis for bioinformatic operations.

```
#strtde
from Bio.Seq import Seq
my_seq=Seq("CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG")
print (len(my_seq))
print (my_seq.count("G"))
my_seq[4:12]
print(my_seq[4:12])
my_seq[3::2]
print(my_seq[3::2])
my_seq[::-1]
print(my_seq[::-1])
str(my_seq)
```

```
70
20
ACAAGGTT
ACAGTCGAGGACGGAGACTGTAAACGAAAGTGGG
GTGAGCTAGTATATAAGACAACAGAGTTGTTACTAGGAAGGCGTCCAAGTGGATGCCTTTGGAACAATGC
'CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG'
```

len(my_seq)

len(my_seq) ÿ Returns the number of nucleotides in the sequence.

ÿ Result: 60 (because the sequence consists of 60 nucleotides)

my_seq.count("G")

my_seq.count("G") ÿ Counts how many times the nucleotide "G" is repeated in the sequence.
ÿ For example, if there are 15 "G"s in the sequence, the result will be 15.

Selecting a Specific Part of a Sequence (Slicing)

my_seq[4:12] print(my_seq[4:12])

•my_seq[4:12] ÿ Selects the section **starting from index 4 to index 12** (not including 12).

•**Since Python indexing starts at 0,** the 4th index will be the letter A.

ÿ **For example, if the sequence is "CGTAACAAGGTTTCC...",** the 4:12 **part** will be "CAAGGTTT" .

Selecting Parts of a Sequence with Certain Steps (Stride)

my_seq[3::2] print(my_seq[3::2])

•my_seq[3::2] ÿ

- •**Starts from index 3** (letter A).
- • Creates a new sequence **by skipping 2 indexes each time .**

•**For example:**

- •Basic sequencing ÿ

  "CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATC

  "GAGTG"

- •3::2 ÿ "AAGTTCGAGGAACCTGGAATGTTAGAAATAAGT"

Reverse the Sequence

my_seq[::-1] print(my_seq[::-1])

- [::-1] ÿ **Completely reverses the sequence.**
- **For example:**
  - Basic sequencing ÿ
    "CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG"
  - Reverse sequencing ÿ
    "GTGAGCTAGTATATAAGACAACTGAAGTTGATCATGGAGGCGTCCAGGAATGGTCCTTGAACTAAGTACG"

Converting a Seq Object to a String

str(my_seq)

- str(my_seq) ÿ Converts the Seq object **to a regular Python string .**
- **This operation makes it easier to work with string methods** (e.g., .replace(), .find(), etc.).

Code Usage Areas

- ÿ **Distinguishing certain sequences in genetic analyses**
- ÿ **Investigating different parts of DNA sequences**
- ÿ **Convert sequences for reverse-complementary analyses**
- ÿ **Determining the number of individual nucleotides in genome studies**

[245]:
```
from Bio.Seq import Seq
from Bio.SeqUtils import GC   # SeqUtils düzgün yazıldı
my_seq = Seq("CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG")
print(GC(my_seq))  # GC faizini çap edir
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[245], line 2
      1 from Bio.Seq import Seq
----> 2 from Bio.SeqUtils import GC   # SeqUtils düzgün yazıldı
      3 my_seq = Seq("CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG")
      4 print(GC(my_seq))

ImportError: cannot import name 'GC' from 'Bio.SeqUtils' (C:\Users\Acer\anaconda3\Lib\site-packages\Bio\SeqUtils\__init__.py)
```

[249]:
```
from Bio.Seq import Seq
from Bio.SeqUtils import gc_fraction  # Yeni funksiya adı
my_seq = Seq("CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG")
gc_content = gc_fraction(my_seq) * 100  # GC faizi üçün 100-ə vururuq
print(f"GC Content: {gc_content:.2f}%")
```

```
GC Content: 44.29%
```

This code **calculates the GC percentage of a DNA sequence** using **the Seq** and **SeqUtils** modules of **the Biopython** library .

What is GC Content?

It represents the total percentage of **Guanine (G) and Cytosine (C)** nucleotides in a DNA sequence .

Formula:

$$GC\% = \left(\frac{G + C}{\ddot{u}muminukleotidsay\imath} \right) \times 100$$

GC content provides information about **the stability and evolution of DNA .**

- Seq ÿ Biopython's object class **for working with DNA, RNA, or protein sequences .**
- gc_fraction ÿ **A function that calculates the percentage of GC in a DNA sequence.**

Note:

Previously, the GC function was used, but since **Biopython version 1.78,** gc_fraction is recommended.

Calculating GC Content

gc_content = gc_fraction(my_seq) * 100 # We multiply by 100 for GC percentage

•gc_fraction(my_seq) ÿ Returns **the total percentage of G and C nucleotides** in the sequence.

•* 100 ÿ To convert from a decimal **to a percentage,** we multiply by 100.

For example:

If the GC fraction is 0.45,

GC percentage: 0.45 * 100 = 45%

•**F-string** (f"") ÿ Used to write output in a formatted way.

•{gc_content:.2f} ÿ Prints the GC percentage **with two digits after the comma .**

•**Screen output (for example, if the GC percentage is 45.67%):**

GC Content: 45.67%

Code Usage Areas

ÿ Calculation of GC content **in genetic analyses**

ÿ **Learn if DNA is stable** (Higher GC percentage ÿ More stable DNA)

ÿ **Analyses of the evolution and adaptation of microorganisms**

ÿ Primer design **in PCR and other genetic engineering applications**

```
•[9]: from Bio.Seq import Seq
      list_of_seqs=[Seq("ACGT"), Seq("AACC"), Seq("TTCC")]
      concatenated=Seq("")
      for s in list_of_seqs:
          concatenated+=s
          pass
          print(list_of_seqs)
          print("Concatenated Sequence:", concatenated)
```

```
[Seq('ACGT'), Seq('AACC'), Seq('TTCC')]
Concatenated Sequence: ACGT
[Seq('ACGT'), Seq('AACC'), Seq('TTCC')]
Concatenated Sequence: ACGTAACC
[Seq('ACGT'), Seq('AACC'), Seq('TTCC')]
Concatenated Sequence: ACGTAACCTTCC
```

Generating a List of DNA Sequences

list_of_seqs = [Seq("ACGT"), Seq("AACC"), Seq("TTCC")]

•list_of_seqs ÿ is a list of DNA sequences.

•List elements:

"ACGT"

"AACC"

"TTCC"

Note:

Seq("ACGT") ÿ This **is not a str type, but a Biopython** Seq **object.**

Creating an Empty Sequence

concatenated = Seq("")

•concatenated ÿ **An empty** Seq **object for the sequences to be concatenated.**

If we **want to print the concatenated sequence,** we need to add this to the end of the code:

print("Concatenated Sequence:", concatenated)

Application Areas of the Code ÿ **Combining multiple DNA sequences** ÿ **Creating primers for genetic analyses** ÿ **Gene splicing and mutation studies**

```
[7]: from Bio.Seq import Seq
     contigs=[Seq("ATG"), Seq("TTCC"), Seq("TTGCA")]
     spacer=Seq("N"*10)
     spacer.join(contigs)
```

```
[7]: Seq('ATGNNNNNNNNNNTTCCNNNNNNNNNNTTGCA')
```

The following steps occur in the code:

1. Three different Seq (sequence) objects are created in a list called contigs:

    •Seq("ATG"): This is a start codon (ATG) sequence.

    •Seq("TTCC"): This is another short-term sequence.

    •Seq("TTGCA"): This is another sequence.

2. A Seq object is created in a variable called spacer, which consists of 10 "N" sequences (Seq("N"*10)), where the letter "N" represents the unknown bases. represents.

3. With the command spacer.join(contigs), the spacer sequence is joined to all sequences in the contigs list. This join operation,

It concatenates sequences and adds a break of 10 "N" between each sequence.

As a result, a new sequence is created that includes 10 "N" letters between the contigs. This method is used to store genetic information or

useful for integrating other data used in bioinformatics.

```
[113]: from Bio.Seq import Seq
       dna_seq=Seq("acgtACGT")
       print(dna_seq)
       print(dna_seq.upper())
       print(dna_seq.lower())
```

```
acgtACGT
ACGTACGT
acgtacgt
```

This code uses the Bio.Seq module of the BioPython library to show how a DNA sequence is represented in different cases.

Below is an explanation of each command:

1.dna_seq = Seq("acgtACGT"):

- This line creates a Seq object and stores the sequence "acgtACGT" in a variable called dna_seq. Here, inside the Seq object, there are lowercase and

  There is a DNA sequence shown in capital letters.

2.print(dna_seq):

- This command prints the dna_seq variable to the screen. The result will be the sequence "acgtACGT" as is.

3.print(dna_seq.upper()):

- This command converts all letters in the dna_seq sequence to uppercase. The upper() method converts lowercase letters in any sequence replaces with uppercase letters. As a result, ACGTACGT will be printed.

4.print(dna_seq.lower()):

- This command converts all letters in the dna_seq sequence to lowercase. The lower() method converts all uppercase letters in any sequence to lowercase. replaces with letters. As a result, acgtacgt will be printed.

The general purpose of this code is to display the different spellings (lowercase and uppercase) of a DNA sequence. The upper() and lower() methods, used to change the writing style of sequences, such changes can be particularly useful in bioinformatic analyses.

```
[129]: from Bio.Seq import Seq
       my_seq=Seq("CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG")
       print(my_seq)
       print(my_seq.complement())
       print(my_seq.reverse_complement())
```

```
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG
GCATTGTTCCAAAGGCATCCACTTGGACGCCTTCCTAGTAACAACTCTGTTGTCTTATATACTAGCTCAC
CACTCGATCATATATTCTGTTGTCTCAACAATGATCCTTCCGCAGGTTCACCTACGGAAACCTTGTTACG
```

This code uses the Bio.Seq module of the BioPython library to generate the complement and reverse complement of a DNA sequence.

The methods used in the code and their purposes are explained below:

1.my_seq = Seq("CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG"):

- •This line creates a Seq object and stores the string "CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG" in a variable called my_seq

  This sequence is a DNA sample.

2.print(my_seq):

- •This command prints the sequence my_seq to the screen. As a result, the DNA sequence is as follows:

  ("CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG") will be displayed.

3.print(my_seq.complement()):

- •complement() method returns the complementary bases of a sequence. The complementary bases of DNA are:
  - •A (Adenine) ÿ T (Thymine)
  - •T (Thymine) ÿ A (Adenine)
  - •C (Cytosine) ÿ G (Guanine)
  - •G (Guanine) ÿ C (Cytosine)

This command prints the complementary form of the sequence my_seq (i.e., replacing each base with its complementary base).

4.print(my_seq.reverse_complement()):

- The reverse_complement() method combines two operations:
  - First, it finds the complementary bases of the sequence.
  - Then it reverses this complementary sequence.

As a result, both the complementary and inverted forms of the sequence are obtained and printed to the screen.

Example:

- my_seq: CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG
- complement(): GCATTTGTTCCAAGGCATCCACTTGGACGCCCTTCCTAGTAACAACACTCTTGTCCTATATACTAGCTCA
- reverse_complement(): GCTCAGTATATCCACTTGTTCAGGCAGTCAAGTGAATCCTTCAGCTGACCTTGGACGGAAGTTAACATGCG

These methods are widely used, especially in bioinformatics, when analyzing and comparing DNA sequences.

Thanks