



Chapter 02. Working with Sequence

Python Programming for Bioinformatics

Robert C. Chi

Agenda

- **Create a Biological Sequence**
- **Basic Operations for Sequences**
- **Functions Applying for Sequences**
- **Biological Operations for Sequences**
- **Modifiable Sequences**
- **Biological Functions for Strings**





CREATE A BIOLOGICAL SEQUENCE

Introduction to Bio.Seq.Seq

Bio.Seq.Seq

(Bio.Seq.Seq is a Read-only Object)

FASTA

```
27656581emb1Z78533.1|C1Z78533 C. irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA
ACAAGGTTTCCTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGAATAAAGCATCGAGTG
GGAGGACCGGTGTACTCAGCTCACCAGGGGGCAATGCTCCCGTGGTGAACCTGATTTGTGTGGG
CCGGAGCGTCCATGGCGGGTTTGAACCTCTAGCCCGGGCGCAGTTGGGGGCCAAGCCATATGAA
CCCGCGAATGGCATTTCTTCCCAAACCCGAGCGGGCGGTGCTGGCGTGGCCCAATGAA
ATTTTGTGACTCTGGCAAACGGGAATCTGGCTCTTTGCATCGGATGGAAGGACGCAAGAAATGGCAT
AAGTGGTGTGAATGCAAGATCCCGTGAACCATGAGCTTTTGAAGCAAGTTGCGCCGAGGCCATCA
GGCTAAGGGCACCGCTGCTTGGGGCGTGGGCTTCTCTCTCCGCAATGCTTGGCCGCAATACAGCC
AGGCGCGGTGGTGGGATGGAAGATTTGGCCCTTGTGCTAGGTGGCGGGTCCAAAGACTGGTGT
TTTGTATGCCCGGAAACCCGCAAGAGGTGGACGATGCTGGCAGCAGCTGCGTGGCAATCCCCCATGTT
GTGTGCTTTGTGGACAGGCGAGGAACCTTCCGAAACCCAAATGAGGGCGGTTGACCGCAATTCGGAT
GTGACCCAGGTGAGGCGGGGACCCCGCTGAGTTACGC
```

GenBank

```
LOCUS       Z78533                740 bp    DNA     linear   PLN 30-NOV-2006
DEFINITION  C. irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA.
ACCESSION   Z78533
VERSION     Z78533.1  GI:2765658
KEYWORDS    5.8S ribosomal RNA; 5.8S rRNA gene; internal transcribed spacer;
            ITS1; ITS2.
SOURCE      Cyripedium irapeanum
ORGANISM    Cyripedium irapeanum
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; Liliopsida; Asparagales; Orchidaceae;
            Cyripedioideae; Cyripedium.
REFERENCE   1
AUTHORS     Cox,A.V., Pridgeon,A.M., Albert,V.A. and Chase,M.W.
TITLE       Phylogenetics of the slipper orchids (Cypripedioideae:
            Orchidaceae): nuclear rDNA ITS sequences
JOURNAL     Unpublished
REFERENCE   2 (bases 1 to 740)
AUTHORS     Cox,A.V.
TITLE       Direct Submission
JOURNAL     Submitted (19-AUG-1996) Cox A.V., Royal Botanic Gardens, Kew,
            Richmond, Surrey TW9 3AB, UK
FEATURES             Location/Qualifiers
     source            1..740
                     /organism="Cyripedium irapeanum"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:49711"
     feature            1..380
                     /note="internal transcribed spacer 1"
                     381..550
                     /gene="5.8S rRNA"
                     381..550
                     /gene="5.8S rRNA"
                     /product="5.8S ribosomal RNA"
                     551..740
                     /note="internal transcribed spacer 2"
```

Different Types of Sequences

```
1 from Bio.Seq import Seq
2
3 # DNA Sequence
4 dna_seq = Seq("ATCG")   Legal Chars: 'ATCG'
5 print(dna_seq)
6
7 # RNA Sequence
8 rna_seq = Seq("UAGC")   Legal Chars: 'UAGC'
9 print(rna_seq)
10
11 # Protein Sequence
12 protein_seq = Seq("MELKILV")   Legal Chars:
13 print(protein_seq)           'ACDEFGHIKLMNPQRSTVWY'
14
15 # Sequence with Unknown Letter 'N' (@GenBank or EMBL)
16 unknown_seq = Seq("ATNNNCG")
17 print(unknown_seq)
18
19 # Sequence with gap letter '-'
20 gap_seq = Seq("ATC--GCA")
21 print(gap_seq)
```

Output:

```
ATCG
UAGC
MELKILV
ATNNNCG
ATC--GCA
```

Notice:

- Although you can add **any character** to `Seq`, **invalid characters** can **cause errors** in the program when performing certain actions with biological meaning. (e.g., Transcription, Translation...etc.)

Practice

- **Construct Different Types of Sequences**

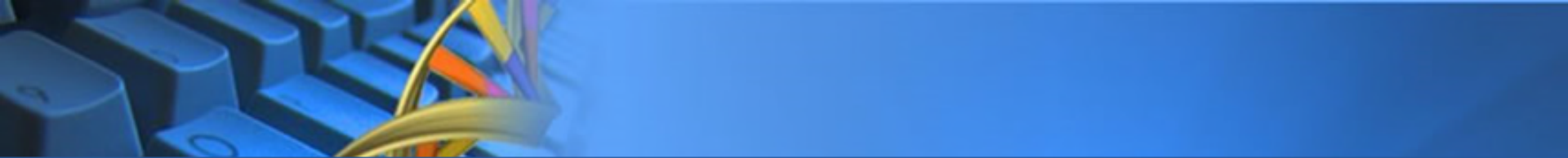
- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

▼ Create Different Types of Sequence

```
[17] 1 from Bio.Seq import Seq
      2
      3 # DNA Sequence
      4 dna_seq = Seq("ATCG")
      5 print(dna_seq)
      6
      7 # RNA Sequence
      8 rna_seq = Seq("UAGC")
      9 print(rna_seq)
     10
     11 # Protein Sequence
     12 protein_seq = Seq("MELKILV")
     13 print(protein_seq)
     14
     15 # Sequence with Unknown Letter 'N' (@GenBank or EMBL)
     16 unknown_seq = Seq("ATNNNGC")
     17 print(unknown_seq)
     18
     19 # Sequence with gap letter '-'
     20 gap_seq = Seq("ATC--GCA")
     21 print(gap_seq)
```

(Solution [URL](#) of this Practice)





BASIC OPERATIONS FOR SEQUENCES

Add Two Sequences

```
1 from Bio.Seq import Seq
2
3 # Add two Nucleic Acid Sequences
4 dna_seq1 = Seq("AATC")
5 dna_seq2 = Seq("CGAT")
6 dna_seq = dna_seq1 + dna_seq2
7 print(dna_seq)
8 print(dna_seq + Seq("CG"))
9 print(dna_seq + "AGCG")
10
11 # Legal, but non-sense
12 protein_seq = Seq("MELKILV")
13 rna_seq = Seq("UAGC")
14 print(protein_seq + dna_seq)
15 print(dna_seq + rna_seq)
```



```
AATCCGAT
AATCCGATCG
AATCCGATAGCG
MELKILVAATCCGAT
AATCCGATUAGC
```


Practice

- **Add Sequences**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

▼ Add

```
✓ [24] 1  from Bio.Seq import Seq
15      2
      3  # Add two Nucleic Acid Sequences
      4  dna_seq1 = Seq("AATC")
      5  dna_seq2 = Seq("CGAT")
      6  dna_seq = dna_seq1 + dna_seq2
      7  print(dna_seq)
      8  print(dna_seq + Seq("CG"))
      9  print(dna_seq + "AGCG")
     10
     11  # Legal, but non-sense
     12  protein_seq = Seq("MELKILV")
     13  rna_seq = Seq("UAGC")
     14  print(protein_seq + dna_seq)
     15  print(dna_seq + rna_seq)
```

(Solution [URL](#) of this Practice)



Multiply Sequences with Integer

Multiply = Repeat

```
1 from Bio.Seq import Seq
2
3 # Multiply with an Integer
4 dna_seq = Seq("AATC") * 3
5 print(dna_seq)
6
7 protein_seq = Seq("MELKILV") * 2
8 print(protein_seq)
```




```
AATCAATCAATC
MELKILVMELKILV
```

Practice

- **Multiply Sequences by Integers**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

▼ Multiply

```
✓ 0s  1 from Bio.Seq import Seq
2
3 # Multiply with an Integer
4 dna_seq = Seq("AATC") * 3
5 print(dna_seq)
6
7 protein_seq = Seq("MELKILV") * 2
8 print(protein_seq)
```

(Solution [URL](#) of this Practice)



Compare Sequences

```
1 from Bio.Seq import Seq
2
3 seq1 = Seq("ATCG")
4 seq2 = Seq("ATCG")
5 seq3 = Seq("GGCC")
6
7 # Compare two Bio.Seq.Seq objects
8 print(seq1 == seq2)
9 print(seq1 != seq3)
10
11 # Compare Bio.Seq.Seq against a string
12 print(seq1 == "ATCG")
13 print(seq1 == "GGCC")
14 print(seq2 != "AATT")
```



```
True
True
True
False
True
```

Practice

- **Comparing Sequences**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

▼ Compare

```
[▶] 1 from Bio.Seq import Seq
2
3 seq1 = Seq("ATCG")
4 seq2 = Seq("ATCG")
5 seq3 = Seq("GGCC")
6
7 # Compare two Bio.Seq.Seq objects
8 print(seq1 == seq2)
9 print(seq1 != seq3)
10
11 # Compare Bio.Seq.Seq against a string
12 print(seq1 == "ATCG")
13 print(seq1 == "GGCC")
14 print(seq2 != "AATT")
```

(Solution [URL](#) of this Practice)



Slice a Sequence

- “**Slicing**” = The Way to Get a “**Sub-Sequence**”

-14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
0 1 2 3 4 5 6 7 8 9 10 11 12 13

“ATCCGATGC**CCAG**”

CCAG
Slicing of 10 ~ 13

Notice:

- Python is “0-based” (0~13)
- NCBI sequences are “1-based” (1~14)

Slice a Sequence

```
1 from Bio.Seq import Seq
2
3 my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")
4
5 # Slice on a Single Character
6 print(my_seq[2])
7 print(my_seq[-1])
8
9 # Slice on a Region of the Sequence
10 print(my_seq[4:12])
11
12 # Slice with start, stop, and step
13 # To get the 1, 2, 3 codon positions of this DNA sequence
14 print(my_seq[0::3])
15 print(my_seq[1::3])
16 print(my_seq[2::3])
17
18 # Reverse the order of sequence
19 print(my_seq[::-1])
```

```
T
C
GATGGGCC
GCTGTAGTAAG
AGGCATGCATC
TAGCTAAGAC
CGCTAAAAGCTAGGATATATCCGGGTAGCTAG
```

Practice

- **Slicing a Sequence**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

```
▼ Slicing
✓ 0s ▶ 1 from Bio.Seq import Seq
      2
      3 my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")
      4
      5 # Slice on a Single Character
      6 print(my_seq[2])
      7 print(my_seq[-1])
      8
      9 # Slice on a Region of the Sequence
     10 print(my_seq[4:12])
     11
     12 # Slice with start, stop, and step
     13 # To get the 1, 2, 3 codon positions of this DNA sequence
     14 print(my_seq[0:3])
     15 print(my_seq[1:3])
     16 print(my_seq[2:3])
     17
     18 # Reverse the order of sequence
     19 print(my_seq[::-1])
```

(Solution [URL](#) of this Practice)



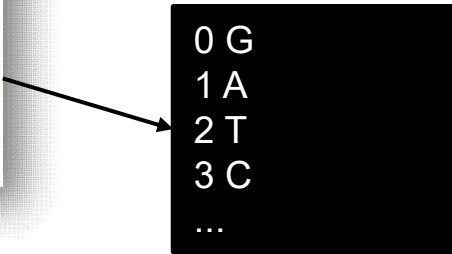
Iterate a Sequence

- **Iteration** = Take **1 letter** a time from the **Sequence**

```
1 from Bio.Seq import Seq
2
3 # Iterate each letter without index
4 my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")
5 for letter in my_seq:
6     print(letter)
7
8 # Iterate each letter with index
9 my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")
10 for index, letter in enumerate(my_seq):
11     print("{} {}".format(index, letter))
```



G
A
T
C
...



0 G
1 A
2 T
3 C
...

Practice

- **Iterating a Sequence**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

Iterate

```
[29] 1  from Bio.Seq import Seq
      2
      3  # Iterate each letter without index
      4  my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")
      5  for letter in my_seq:
      6      print(letter)
      7
      8  # Iterate each letter with index
      9  my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")
     10  for index, letter in enumerate(my_seq):
     11      print("{} {}".format(index, letter))
```

(Solution [URL](#) of this Practice)





FUNCTIONS APPLYING FOR SEQUENCES

Length of a Sequence

```
1 from Bio.Seq import Seq
2 my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")
3                                     32 Characters
4 # len(): Get the length of a sequence
5 print(Len(my_seq))
```



32

Practice

- **Get the Length of a Sequence**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

```
▼ Length  
✓ [30] 1 from Bio.Seq import Seq  
0s     2 my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")  
       3  
       4 # len(): Get the length of a sequence  
       5 print(len(my_seq))
```

(Solution [URL](#) of this Practice)



Change the Case

```
1 from Bio.Seq import Seq
2 my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")
3
4 # Change to lower case and save back to my_seq
5 my_seq = my_seq.lower()
6 print(my_seq)
7
8 # Change my_seq to upper case
9 print(my_seq.upper())
10 print(my_seq)
```

gatcgatgggcctatataggatcgaaaatcgc

GATCGATGGGCCTATATAGGATCGAAAATCGC
gatcgatgggcctatataggatcgaaaatcgc

Practice

- **Change the Case of Sequences**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

▼ Change the Case

```
✓ [32] 1  from Bio.Seq import Seq
0s      2  my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")
        3
        4  # Change to lower case and save back to my_seq
        5  my_seq = my_seq.lower()
        6  print(my_seq)
        7
        8  # Change my_seq to upper case
        9  print(my_seq.upper())
       10  print(my_seq)
```

(Solution [URL](#) of this Practice)



Check for Containing

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 my_dna = Seq("ATATGAAATTTGAAAA")
5
6 # Check for containing by Bio.Seq.Seq
7 print(Seq("AAA") in my_dna)
8
9 # Check for containing by regular strings
10 print("AAA" in my_dna)
```

- 1 Create a Sequence
- 2 Sequence <compare> Sequence
- 3 String <compare> Sequence

Practice

- **Check for Containing**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

- ▼ Check for Containing

```
✓ ▶ 1 from Bio.Seq import Seq
    2
    3 # Create a Sequence
    4 my_dna = Seq("ATATGAAATTTGAAAA")
    5
    6 # Check for containing by Bio.Seq.Seq
    7 print(Seq("AAA") in my_dna)
    8
    9 # Check for containing by regular strings
   10 print("AAA" in my_dna)
```

(Solution [URL](#) of this Practice)



Count the Occurrence

- **Non-overlapped Counting**

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 my_seq = Seq("AAAATCCGCGATAGC")
5
6 # Non-overlapped Counting
7 print(my_seq.count("A"))
8 print(my_seq.count("AA"))
9 print(my_seq.count("AA", 2, -1)) # index= 2~-1
```

`.count("A")`

AAAATCCGCGATAGC

→ 6

`.count("AA")`

AAAATCCGCGATAGC

→ 2

`.count("AA", 2, -1)`

2 → -1

AAATCCGCGATAGC

→ 1

Count the Occurrence

- **Overlapped Counting**

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 my_seq = Seq("AAAATCCGCGATAGC")
5
6 # Overlapped Counting
7 print(my_seq.count_overlap("A"))
8 print(my_seq.count_overlap("AA"))
9 print(my_seq.count_overlap("AA", 2, -1))
```

`.count_overlap("A")`

AAAATCCGCGATAGC

→ 6

`.count_overlap("AA")`

AAAATCCGCGATAGC

→ 3

`.count_overlap("AA", 2, -1)`

2 → -1

AAATCCGCGATAGC

→ 1

Count the Occurrence

- **Count for GC%**

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 my_seq = Seq("AAAATCCGCGATAGC")
5
6 # Count for GC%
7 gc_percent = float(my_seq.count("G") + my_seq.count("C")) / Len(my_seq)
8 print("{0:.2%}".format(gc_percent))
9
10 from Bio.SeqUtils import GC
11 gc_percent = GC(my_seq)
12 print("{0:.2f}%".format(gc_percent))
```

Manual calculation



Using the library

Practice

- **Count the Occurrence**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

- ▼ Count for Occurrence

```
[13] 1 from Bio.Seq import Seq
      2
      3 # Create a Sequence
      4 my_seq = Seq("AAAATCCGCGATAGC")
      5
      6 # Non-overlapped Counting
      7 print(my_seq.count("A"))
      8 print(my_seq.count("AA"))
      9 print(my_seq.count("AA", 2, -1)) # index= 2~-1
     10
     11 # Overlapped Counting
     12 print(my_seq.count_overlap("A"))
     13 print(my_seq.count_overlap("AA"))
     14 print(my_seq.count_overlap("AA", 2, -1))
     15
     16 # Count for GC%
     17 gc_percent = float(my_seq.count("G") + my_seq.count("C"))/len(my_seq)
     18 print("{0:.2%}".format(gc_percent))
     19
     20 from Bio.SeqUtils import GC
     21 gc_percent = GC(my_seq)
     22 print("{0:.2f}%".format(gc_percent))
```

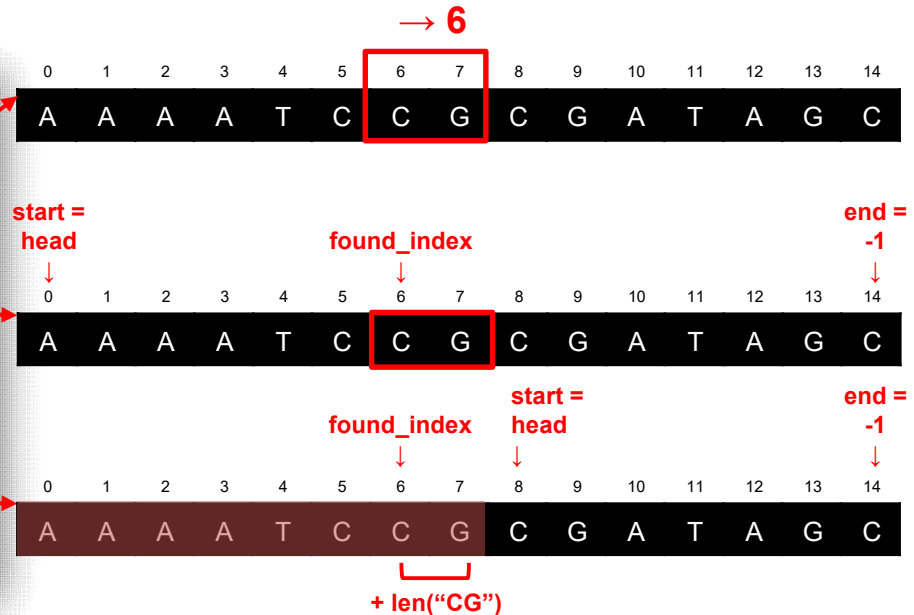


(Solution [URL](#) of this Practice)

Find Sequence Patterns

- Find Forwardly: `.find()`

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 my_seq = Seq("AAAATCCGCGATAGC")
5
6 # Find forward for the first occurrence
7 print("First Occurred:", my_seq.find("CG"))
8
9 # Find forward for all the occurrence
10 occur = my_seq.count("CG") → 2 (6~7, 8~9)
11 head = 0
12 while occur > 0:
13     found_index = my_seq.find("CG", start=head, end=-1)
14     print("Found at:", found_index)
15     occur -= 1
16     head = found_index + len("CG")
17     print("occur: {} head: {}".format(occur, head))
```



Practice

- **Find Sequence Patterns Forwardly**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 my_seq = Seq("AAAATCCGCGATAGC")
5
6 # Find forward for the first occurrence
7 print("First Occurred:", my_seq.find("CG"))
8
9 # Find forward for all the occurrence
10 occur = my_seq.count("CG")
11 head = 0
12 while occur > 0:
13     found_index = my_seq.find("CG", start=head, end=-1)
14     print("Found at:", found_index)
15     occur -= 1
16     head = found_index + len("CG")
17     print("occur: {} head: {}".format(occur, head))
```

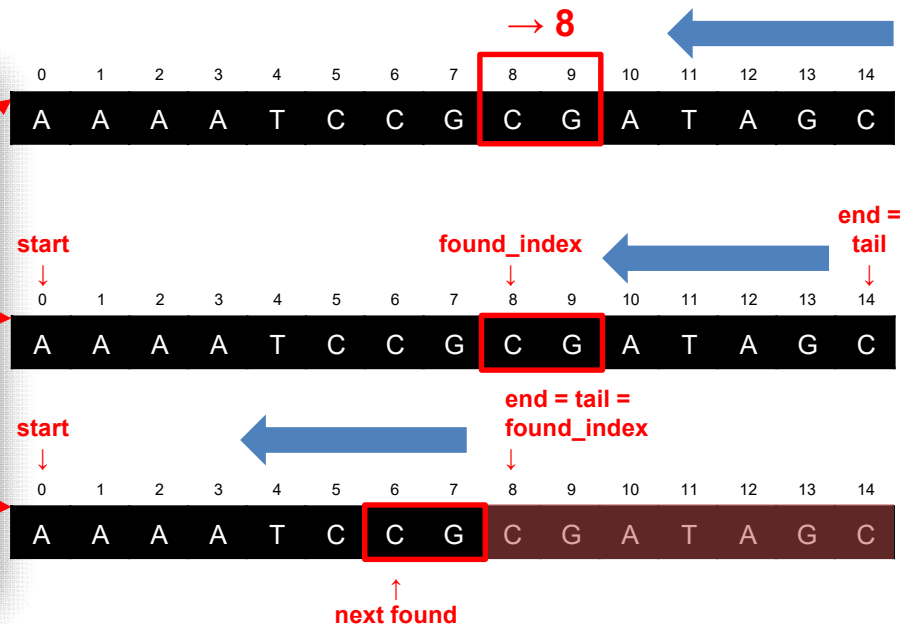
(Solution [URL](#) of this Practice)



Find Sequence Patterns

- Find Backwardly: `.rfind()`

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 my_seq = Seq("AAAATCCGCGATAGC")
5
6 # Find backward for the first occurrence
7 print("Last Occurred:", my_seq.rfind("CG"))
8
9 # Find backward for all the occurrence
10 occur = my_seq.count("CG") → 2 (6~7, 8~9)
11 tail = len(my_seq)
12 while occur > 0:
13     found_index = my_seq.rfind("CG", start=0, end=tail)
14     print("Found at:", found_index)
15     occur -= 1
16     tail = found_index
17     print("occur: {} tail: {}".format(occur, tail))
```



Practice

- **Find Sequence Patterns Backwardly**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 my_seq = Seq("AAAATCCGCGATAGC")
5
6 # Find backward for the first occurrence
7 print("Last Occurred:", my_seq.rfind("CG"))
8
9 # Find backward for all the occurrence
10 occur = my_seq.count("CG")
11 tail = len(my_seq)
12 while occur > 0:
13     found_index = my_seq.rfind("CG", start=0, end=tail)
14     print("Found at:", found_index)
15     occur -= 1
16     tail = found_index
17     print("occur: {} tail: {}".format(occur, tail))
```

(Solution [URL](#) of this Practice)



Find Sequence Patterns

- **Check for Start Codons & Stop Codons**

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 mRNA = Seq("AUGGCCAUUGUAAUUGGGCCGCUGAAAGGGUGCCCGAUAGUUG")
5
6 # Check for start codon "AUG" (Met/Methionine)
7 print(mRNA.startswith("AUG"))
8
9 # Check for start codon from specific index
10 print(mRNA.startswith("AUG", start=12))
11
12 # Check for other possible start codons of E. coli
13 print(mRNA.startswith(("AUG", "GUG", "UUG", "AUU", "CUG")))
14
15 # Check for possible stop codons: "UAG" (amber), "UAA" (orche), "UGA" (opal)
16 print(mRNA.endswith(("UAG", "UAA", "UGA")))
17 print(mRNA.endswith(("UAG", "UAA", "UGA"), end=-3))
```

← Check Start Codon

← Check Start Codon
at Specific Location

← Check all Start Codons

← Check all Stop Codons

Practice

- **Check for Start Codons & Stop Codons**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

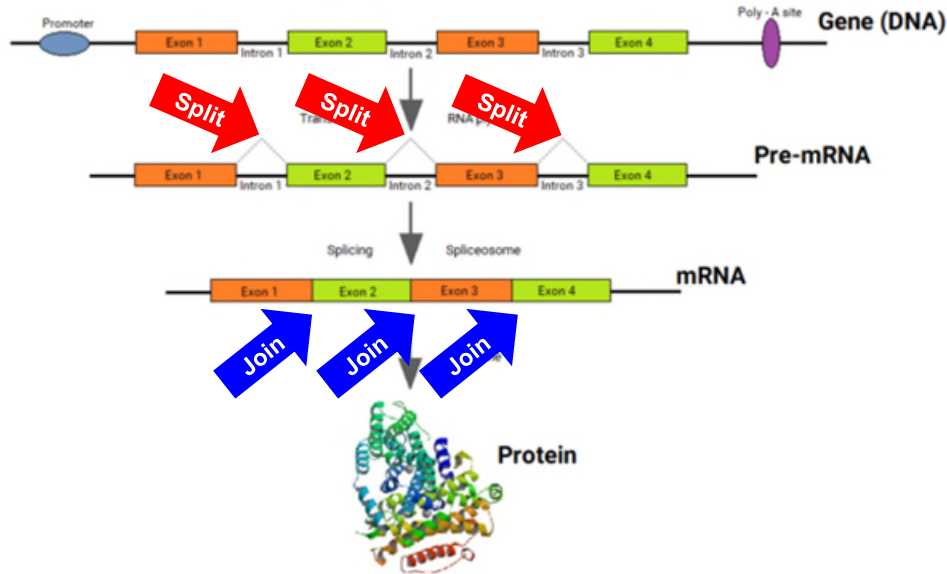
```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 mRNA = Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAGUUG")
5
6 # Check for start codon "AUG" (Met/Methionine)
7 print(mRNA.startswith("AUG"))
8
9 # Check for start codon from specific index
10 print(mRNA.startswith("AUG", start=12))
11
12 # Check for other possible start codons of E. coli
13 print(mRNA.startswith(("AUG", "GUG", "UUG", "AUU", "CUG")))
14
15 # Check for possible stop codons: "UAG" (amber), "UAA" (orche), "UGA" (opal)
16 print(mRNA.endswith(("UAG", "UAA", "UGA")))
17 print(mRNA.endswith(("UAG", "UAA", "UGA"), end=-3))
```

(Solution [URL](#) of this Practice)



Split & Join Sequences

- **Biological Meanings of Split & Join**



Split & Join Sequences

- **Split a Sequence**

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 Pre_mRNA = Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAGUUG")
5
6 # Split at Introns
7 Introns = "GGCC"
8 Exons = Pre_mRNA.split(Introns)
9 print(Exons)
```

AUGGGCAUUGUAAUGGGCGCUGAAAGGGUGCCCGAUAGUUG



[Seq('AU'), Seq('AUUGUAAUG'), Seq('GCUGAAAGGGUGCCCGAUAGUUG')]

Split & Join Sequences

- **Join Sequences**

Exons = [Seq('AU'), Seq('AUUGUAAUG'), Seq('GCUGAAAGGGUGCCCGAUAGUUG')]

```
1 # Join all Exons with spacers
2 spacer = Seq("")
3 mRNA1 = spacer.join(Exons)
4 print(mRNA1)
5
6 spacer = Seq("N"*3)
7 mRNA2 = spacer.join(Exons)
8 print(mRNA2)
```

mRNA1 =

AUAUUGUAAUGGCUGAAAGGGUGCCCGAUAGUUG

mRNA2 =

AUNNNAUUGUAAUGNNNGCUGAAAGGGUGCCCGAUAGUUG

Practice

- **Split & Join Sequences**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

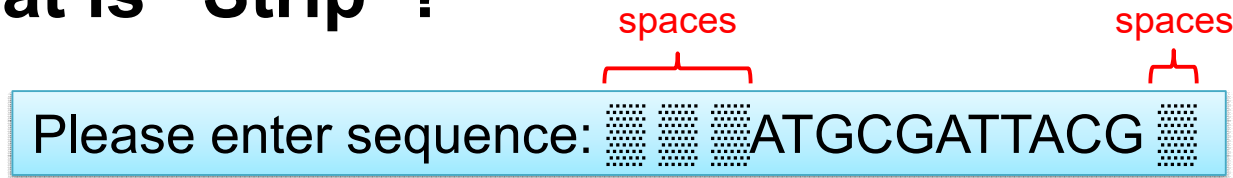
```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 Pre_mRNA = Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAGUUG")
5
6 # Split at Introns
7 Introns = "GGCC"
8 Exons = Pre_mRNA.split(Introns)
9 print(Exons)
10
11 # Join all Exons with spacers
12 spacer = Seq("")
13 mRNA1 = spacer.join(Exons)
14 print(mRNA1)
15
16 spacer = Seq("N"*3)
17 mRNA2 = spacer.join(Exons)
18 print(mRNA2)
```

(Solution [URL](#) of this Practice)



Strip Spaces or Other Symbols

- What is “Strip”?

Please enter sequence: 



ATGCGATTACG

- Why “Strip”?
 - Remove unwanted characters from both sides of the sequence.

Strip Spaces or Other Symbols

```
1 from Bio.Seq import Seq
2
3 # Strip spaces enter by users
4 my_seq = Seq(input("Please enter sequence: "))
5 print("Before Strip:", my_seq)
6 print("After Strip:", my_seq.strip(" "))
7
8 # Strip unwanted characters
9 my_seq = Seq("---ATGCGACCGA-")
10 print("Before Strip:", my_seq)
11 print("After Strip:", my_seq.strip("-"))
```

Please enter sequence:   ATGGCGAG 

Before Strip:   ATGGCGAG 

After Strip: ATGGCGAG

Before Strip: ---ATGCGACCGA-

After Strip: ATGCGACCGA

Practice

- **Strip Spaces or Other Symbols**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

```
1 from Bio.Seq import Seq
2
3 # Strip spaces enter by users
4 my_seq = Seq(input("Please enter sequence: "))
5 print("Before Strip:", my_seq)
6 print("After Strip:", my_seq.strip(" "))
7
8 # Strip unwanted characters
9 my_seq = Seq("---ATGCGACCGA-")
10 print("Before Strip:", my_seq)
11 print("After Strip:", my_seq.strip("-"))
```

(Solution [URL](#) of this Practice)





BIOLOGICAL OPERATIONS FOR SEQUENCES

Ungap a Sequence

- What is “**Ungap**”?

- Return a copy of the sequence **without** the **gap** character(s).

-ATA--TGAAAT-TTGAAAA



ATATGAAATTTGAAAA

CGGGTAG=AAAAAA



CGGGTAGAAAAAA

Ungap a Sequence

```
1 from Bio.Seq import Seq
2
3 # Ungapped a Sequence
4 my_seq = Seq("-ATA--TGAAAT-TTGAAAA")
5 print("Before Ungapped:", my_seq)
6 print("After Ungapped:", my_seq.ungap())
7
8 # Ungapped with specific character
9 my_seq = Seq("CGGGTAG=AAAAAA")
10 print("Before Ungapped:", my_seq)
11 print("After Ungapped:", my_seq.ungap("="))
```

Before Ungapped: -ATA--TGAAAT-TTGAAAA
After Ungapped: ATATGAAATTTGAAAA

Before Ungapped: CGGGTAG=AAAAAA
After Ungapped: CGGGTAGAAAAAA

Practice

- **Ungap a Sequence**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

```
1 from Bio.Seq import Seq
2
3 # Ungapped a Sequence
4 my_seq = Seq("-ATA--TGAAAT-TTGAAAA")
5 print("Before Ungapped:", my_seq)
6 print("After Ungapped:", my_seq.ungap())
7
8 # Ungapped with specific character
9 my_seq = Seq("CGGGTAG=AAAAAA")
10 print("Before Ungapped:", my_seq)
11 print("After Ungapped:", my_seq.ungap("="))
```

(Solution [URL](#) of this Practice)



Transcription

- What is **Transcription**?

DNA coding strand (aka Crick strand, strand +1)

5' ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG 3'

|||||

3' TACCGGTAACATTACCCGGCGACTTTCCCACGGGCTATC 5'

DNA template strand (aka Watson strand, strand -1)

Forward
Transcription



Backward
Transcription

5' AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG 3'

Single stranded messenger RNA

Transcription

- **Forward Transcription**

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG") # 5'→3'
5 template_dna = coding_dna.complement() # 3'→5'
6 mRNA = coding_dna.transcribe()
7
8 # Forward Transcription
9 print("5' {} 3' : Coding Strand".format(coding_dna))
10 print("3' {} 5' : Template Strand".format(template_dna))
11 print("5' {} 3' : mRNA".format(mRNA))
12
13 # Transcribe Protein Sequence
14 print("Transcribe Protein Sequence:", Seq("MAIVMGR").transcribe())
```

5' ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG 3' : Coding Strand
3' TACCGGTAACATTACCCGGCGACTTTCCCACGGGCTATC 5' : Template Strand
5' AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG 3' : mRNA

Transcribe on Protein Sequence
will be ignored

Transcribe Protein Sequence: MAIVMGR

Transcription

- **Backward Transcription**

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG") # 5'→3'
5 template_dna = coding_dna.complement() # 3'→5'
6 mRNA = coding_dna.transcribe()
7
8 # Backward Transcription
9 print("5' {} 3' : mRNA".format(mRNA))
10 print("5' {} 3' : Backward Transcription".format(mRNA.back_transcribe()))
11
12 # Backward Transcription of Protein Sequence
13 print("Backward Transcription of Protein Sequence:", Seq("MAIVMGR").back_transcribe())
```

5' AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG 3' : mRNA
5' ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG 3' : Backward Transcription

**Backward Transcribe on
Protein Sequence will be ignored**

Backward Transcription of Protein Sequence: MAIVMGR

Practice

- **Transcription**

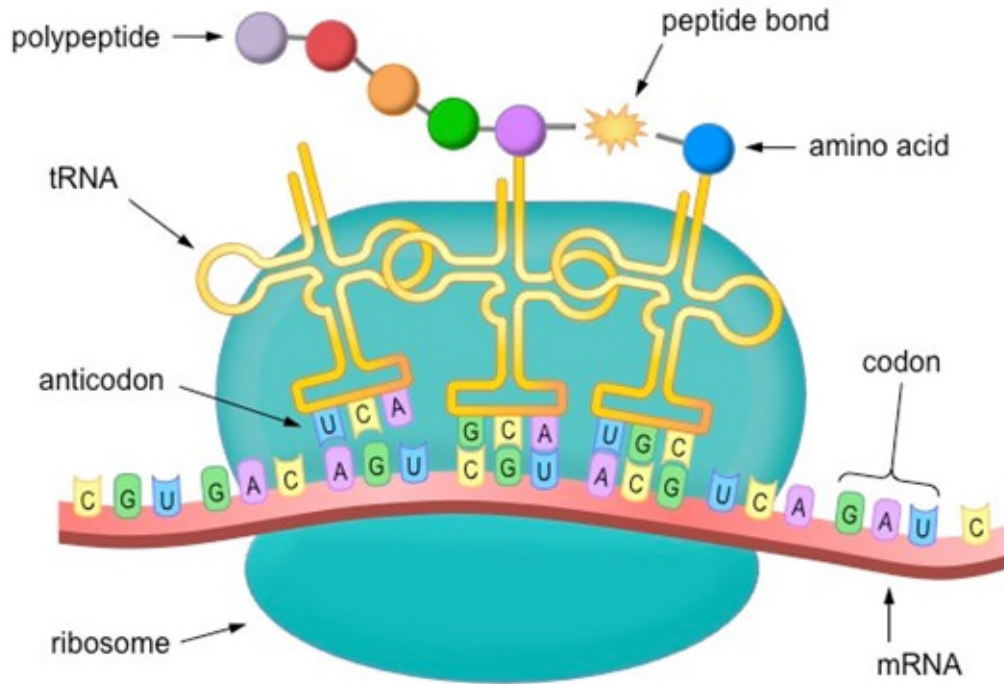
- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG") # 5'→3'
5 template_dna = coding_dna.complement() # 3'→5'
6 mRNA = coding_dna.transcribe()
7
8 # Forward Transcription
9 print("5' {} 3' : Coding Strand".format(coding_dna))
10 print("3' {} 5' : Template Strand".format(template_dna))
11 print("5' {} 3' : mRNA".format(mRNA))
12
13 # Transcribe Protein Sequence
14 print("Transcribe Protein Sequence:", Seq("MAIVMGR").transcribe())
15
16 # Backward Transcription
17 print("5' {} 3' : mRNA".format(mRNA))
18 print("5' {} 3' : Backward Transcription".format(mRNA.back_transcribe()))
19
20 # Backward Transcription of Protein Sequence
21 print("Backward Transcription of Protein Sequence:", Seq("MAIVMGR").back_transcribe())
```



(Solution [URL](#) of this Practice)

Translation



Translation Table (from NCBI)

1. The standard code
2. The vertebrate mitochondrial code
3. The yeast mitochondrial code
4. The mold, protozoan, and coelenterate mitochondrial code and the mycoplasma/spiroplasma code
5. The invertebrate mitochondrial code
6. The ciliate, dasycladacean and hexamita nuclear code
7. The kinetoplast code
9. The echinoderm and flatworm mitochondrial code
10. The euplotid nuclear code
11. The bacterial, archaeal and plant plastid code
12. The alternative yeast nuclear code
13. The ascidian mitochondrial code
14. The alternative flatworm mitochondrial code
15. The *Blepharisma* nuclear code
16. The chlorophycean mitochondrial code
21. The trematode mitochondrial code
22. The *Scenedesmus obliquus* mitochondrial code
23. The *Thraustochytrium* mitochondrial code
24. The Pterobranchia mitochondrial code
25. The candidate division SR1 and gracilibacteria code
26. The *Pachysolen tannophilus* nuclear code
27. The karyorelict nuclear code
28. The *Condylostoma* nuclear code
29. The *Mesodinium* nuclear code
30. The peritrich nuclear code
31. The *Blastocrithidia* nuclear code
33. The Cephalodiscidae mitochondrial code

Translation

- Syntax of `.translate()` Function

```
.translate(table='Standard', stop_symbol='*', to_stop=False, cds=False)
```

- table: Specify which codon table to use (string or integer)
- stop_symbol: Specify which character to represent the stop codon.
- to_stop: Translation is stopped at the first stop codon or not.
- cds: Sequence is started with start codon, ended with stop codon, and is a multiple of 3 or not.

Translation

- Translate with DNA + Standard Table

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 coding_dna = Seq("ATGGCCATTGTGATGGGCCGCTGAAAGGGTGCCCGATAG")
5 print("5' {} 3' : Coding Strand".format(coding_dna))
6
7 mRNA = coding_dna.transcribe()
8 print("5' {} 3' : mRNA".format(mRNA))
9
10 # Translate from DNA with the Standard Table
11 polypeptides = coding_dna.translate()
12 print("Translate (DNA, Standard):", polypeptides)
```

Create a Sequence

```
5' ATGGCCATTGTGATGGGCCGCTGAAAGGGTGCCCGATAG 3' : Coding Strand
5' AUGGCCAUUGUGAUGGGCCGUGAAAGGGUGCCCGAUAG 3' : mRNA
```

Translate with DNA + Standard Table

```
Translate (DNA, Standard): MAIVMGR*KGAR*
```

Translation

- **Other Operations of Translation**

```
14 # Change the Symbol of Stop Codons
15 polypeptides = coding_dna.translate(stop_symbol="@")
16 print("Translate (DNA, Stop='@'):", polypeptides)
17
18 # Translate until Stop Codons
19 polypeptides = coding_dna.translate(to_stop=True)
20 print("Translate (DNA, Until Stop):", polypeptides)
21
22 # Translate from mRNA with the Standard Table
23 polypeptides = mRNA.translate()
24 print("Translate (RNA, Standard):", polypeptides)
```

Change the Symbol

Translate (DNA, Stop='@'): MAIVMGR@KGAR@

Translate until Stop Codon

Translate (DNA, Until Stop): MAIVMGR

Change the Symbol

Translate (RNA, Standard): MAIVMGR*KGAR*

Translation

- Translate with Other Table

```
26 # Translate with Table=2 (Vertebrate Mitochondrial)
27 polypeptides = coding_dna.translate(table=2)
28 print("Translate (DNA, Table=2):", polypeptides)
29
30 # Translate with Table=2, GTG=Start Codon, Fully Coding Sequence (CDS)
31 sub_CDS = coding_dna[9:]
32 print("5' {} 3' : Sub-Coding Sequence".format(sub_CDS))
33 # polypeptides = sub_CDS.translate(cds=True) # Error: GTG is not a Start Codon
34 polypeptides = sub_CDS.translate(table=2, cds=True)
35 print("Translate (DNA, Table=2, CDS):", polypeptides)
```

Translate (DNA, Table=2): MAIVMGRWKGAR*

5' GTGATGGGCCGCTGAAAGGGTGCCCGATAG 3' : Sub-Coding Sequence

Translate (DNA, Table=2, CDS): MMGRWKGAR

Practice

```
1 from Bio.Seq import Seq
2
3 # Create a Sequence
4 coding_dna = Seq("ATGGCCATTGTGATGGGCCGCTGAAAGGGTGCCCGATAG")
5 print("5' {} 3' : Coding Strand".format(coding_dna))
6
7 mRNA = coding_dna.transcribe()
8 print("5' {} 3' : mRNA".format(mRNA))
9
10 # Translate from DNA with the Standard Table
11 polypeptides = coding_dna.translate()
12 print("Translate (DNA, Standard):", polypeptides)
13
14 # Change the Symbol of Stop Codons
15 polypeptides = coding_dna.translate(stop_symbol="@")
16 print("Translate (DNA, Stop='@'):", polypeptides)
17
18 # Translate until Stop Codons
19 polypeptides = coding_dna.translate(to_stop=True)
20 print("Translate (DNA, Until Stop):", polypeptides)
21
22 # Translate from mRNA with the Standard Table
23 polypeptides = mRNA.translate()
24 print("Translate (RNA, Standard):", polypeptides)
25
26 # Translate with Table=2 (Vertebrate Mitochondrial)
27 polypeptides = coding_dna.translate(table=2)
28 print("Translate (DNA, Table=2):", polypeptides)
29
30 # Translate with Table=2, GTG=Start Codon, Fully Coding Sequence (CDS)
31 sub_CDS = coding_dna[9:]
32 print("5' {} 3' : Sub-Coding Sequence".format(sub_CDS))
33 # polypeptides = sub_CDS.translate(cds=True) # Error: GTG is not a Start Codon
34 polypeptides = sub_CDS.translate(table=2, cds=True)
35 print("Translate (DNA, Table=2, CDS):", polypeptides)
```

- **Translation**

- **Write** and **Run** the left codes on a Colab page called “**SeqObjects.ipynb**”:



(Solution [URL](#) of this Practice)



MODIFIABLE SEQUENCES

Problem of Modifying Sequence

```
1 # Create a Sequence
2 from Bio.Seq import Seq
3 my_seq = Seq("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA")
4
5 # Mutation from "T" to "G" at Index = 5
6 my_seq[5] = "G"
```

Error!

TypeError: 'Seq' object does not support item assignment

→ **Bio.Seq.Seq** is an “**Immutable**” object
(**Immutable** = Not allow to change it **partially**)

Practice

- **Test for the Problem of Bio.Seq.Seq Modification**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

```
1 # Create a Sequence
2 from Bio.Seq import Seq
3 my_seq = Seq("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA")
4
5 # Mutation from "T" to "G" at Index = 5 → Error!!
6 my_seq[5] = "G"
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-37-19a818f9f44a> in <module>()
      4
      5 # Mutation from "T" to "G" at Index = 5 → Error!!
----> 6 my_seq[5] = "G"

TypeError: 'Seq' object does not support item assignment
```



(Solution [URL](#) of this Practice)

Solution: Bio.Seq.MutableSeq

- **About Bio.Seq.MutableSeq**
 - A Sequence object that can be modified partially.
- **Possible Operations**

Comparison: ==, !=, <, <=, >, >=

Length: len()

Partially Change: seq[5] = 'T'

Partially Delete: del seq[5]

Concatenation (Add): seq = seq + "GCG"

Duplication (Multiple): seq = seq * 3

Insert: seq.insert(8,'G')

Remove: seq.remove('A')

Count: seq.count("ATG")

Reverse: seq.reverse()

Complement:
seq.complement()
seq.reverse_complement()

Conversion of MutableSeq

String → MutableSeq

```
str_seq = "GCCATTGTA"  
mutable_seq = MutableSeq(str_seq)
```

MutableSeq → String

```
mutable_seq = MutableSeq("GCCATTGTA")  
str_seq = str(mutable_seq)
```

Seq → MutableSeq

```
seq = Seq("GCCATTGTA")  
mutable_seq = MutableSeq(seq)  
mutable_seq = seq.tomutable()
```

MutableSeq → Seq

```
mutable_seq = MutableSeq("GCCATTGTA")  
seq = Seq(mutable_seq)  
seq = mutable_seq.toseq()
```

Practice

- **Example for Bio.Seq.MutableSeq**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

```
1 # Create a Sequence
2 from Bio.Seq import Seq
3 my_seq = Seq("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA")
4 print("5' {} 3' : Original DNA".format(my_seq))
5
6 # Conversion for Bio.Seq.MutableSeq
7 from Bio.Seq import MutableSeq
8 mutable_seq = MutableSeq(my_seq)
9
10 # Mutation
11 mutable_seq[5] = "G"
12 print("5' {} 3' : DNA after Mutation".format(mutable_seq))
13
14 # Reverse
15 mutable_seq.reverse()
16 print("3' {} 5' : DNA after Mutation & Reverse".format(mutable_seq))
```

```
5' GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA 3' : Original DNA
5' GCCATGGTAATGGGCCGCTGAAAGGGTGCCCGA 3' : DNA after Mutation
3' AGCCCGTGGGAAAGTCGCCGGGTAATGGTACCG 5' : DNA after Mutation & Reverse
```



(Solution [URL](#) of this Practice)



BIOLOGICAL FUNCTIONS FOR STRINGS



Biological Functions for Strings

- **What do these functions do?**
 - When you want to perform some **Biological Operations** (e.g., transcribe, translate...etc.) directly on a "string" **instead of** using **Seq** or **MutableSeq** objects.
- **What functions are available?**
 - complement()
 - reverse_complement()
 - transcribe()
 - back_transcribe()
 - translate()

Example

```
1 from Bio.Seq import complement, reverse_complement
2 from Bio.Seq import transcribe, back_transcribe, translate
3
4 # Create a Sequence
5 str_coding = "ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG"
6 str_template = complement(str_coding)
7 str_template_reverse = reverse_complement(str_coding)
8 print("5' {} 3' : Coding Strand".format(str_coding))
9 print("3' {} 5' : Template Strand".format(str_template))
10 print("5' {} 3' : Template Strand".format(str_template_reverse))
11
12 # Transcription
13 str_mRNA = transcribe(str_coding)
14 print("5' {} 3' : mRNA".format(str_mRNA))
15 print("5' {} 3' : Backward Transcription".format(back_transcribe(str_mRNA)))
16
17 # Translation
18 str_protein = translate(str_coding, table=2, stop_symbol="@")
19 print("Protein Sequence:", str_protein)
```

```
5' ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG 3' : Coding Strand
3' TACCGGTAACATTACCCGGCGACTTCCACGGGGCTATC 5' : Template Strand
```

```
5' CTATCGGGCACCCTTTCAGCGGCCATTACAATGGCCAT 3' : Template Strand
```

```
5' AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG 3' : mRNA
```

```
5' ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG 3' : Backward Transcription
```

```
Protein Sequence: MAIVMGRWKGAR@
```

Practice

- **Biological Functions for Strings**

- **Write** and **Run** the following codes on a Colab page called “[SeqObjects.ipynb](#)”:

```
1 from Bio.Seq import complement, reverse_complement
2 from Bio.Seq import transcribe, back_transcribe, translate
3
4 # Create a Sequence
5 str_coding = "ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG"
6 str_template = complement(str_coding)
7 str_template_reverse = reverse_complement(str_coding)
8 print("5' {} 3' : Coding Strand".format(str_coding))
9 print("3' {} 5' : Template Strand".format(str_template))
10 print("5' {} 3' : Template Strand".format(str_template_reverse))
11
12 # Transcription
13 str_mRNA = transcribe(str_coding)
14 print("5' {} 3' : mRNA".format(str_mRNA))
15 print("5' {} 3' : Backward Transcription".format(back_transcribe(str_mRNA)))
16
17 # Translation
18 str_protein = translate(str_coding, table=2, stop_symbol="@")
19 print("Protein Sequence:", str_protein)
```

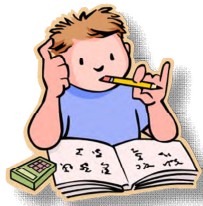
```
5' ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG 3' : Coding Strand
3' TACCGGTAACATTACCCGGCGACTTTCCCACGGGCTATC 5' : Template Strand
5' CTATCGGGCACCCCTTTCAGCGGCCATTACAATGGCCAT 3' : Template Strand
5' AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG 3' : mRNA
5' ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG 3' : Backward Transcription
Protein Sequence: MAIVMGRWKGAR@
```



(Solution [URL](#) of this Practice)

Homework

- **Parse the FASTA file of E. Coli**
 - Please go to the URL: <https://bit.ly/3wHQ6lu>
 - Read the requirements listed on the above URL.
 - Create your own Colab page and start to implement the requirements.
 - When you finished, click “Share” button at the upper-right.
 - Click “Get Link” > “Copy Link” from the pop-up window.
 - Send your page’s link to [Google Classroom](#) as the result of this homework.



Summary

- **Create a Sequence**
 - `dna_seq = Seq("ATCG")`
- **Basic Operations**
 - Add: `Seq("AATC") + Seq("CGAT")`
 - Multiple: `Seq("AATC") * 3`
 - Compare: `Seq("ATCG") != Seq("GGCC")`
 - Slicing: `Seq("GATCGA")[0:4:2]`
 - Iterate:
 - `for letter in my_seq:`
 - `for index, letter in enumerate(my_seq):`
- **Functions Applying for Sequences**
 - Length: `len(my_seq)`
 - Case: `my_seq.lower()`, `my_seq.upper()`
 - Contain: `Seq("AAA") in my_dna`
 - Count:
 - Non-overlapped: `my_seq.count("AA")`
 - Overlapped: `my_seq.count_overlap("AA")`
 - GC%: `Bio.SeqUtils.GC(my_seq)`
 - Find: `find()`, `rfind()`, `startswith()`, `endswith()`
 - Split & Join:
 - `Pre_mRNA.split("GGCC")`
 - `"".join([Seq('AU'), Seq('AUUGUT')])`
 - Strip: `my_seq.strip("-")`



Summary

- **Biological Operations**
 - Ungap: `my_seq.ungap("-")`
 - Transcription:
 - `coding_dna.transcribe()`
 - `protein.back_transcribe()`
 - Translation: `coding_dna.translate(table=2, stop_symbol="@")`
- **Bio.Seq.MutableSeq**
 - Create: `mutable_seq = MutableSeq(my_seq)`
 - Mutation: `mutable_seq[5] = "G"`
 - Reverse: `mutable_seq.reverse()`
- **Biological Functions for Strings**
 - `complement()`
 - `reverse_complement()`
 - `transcribe()`
 - `back_transcribe()`
 - `translate()`

