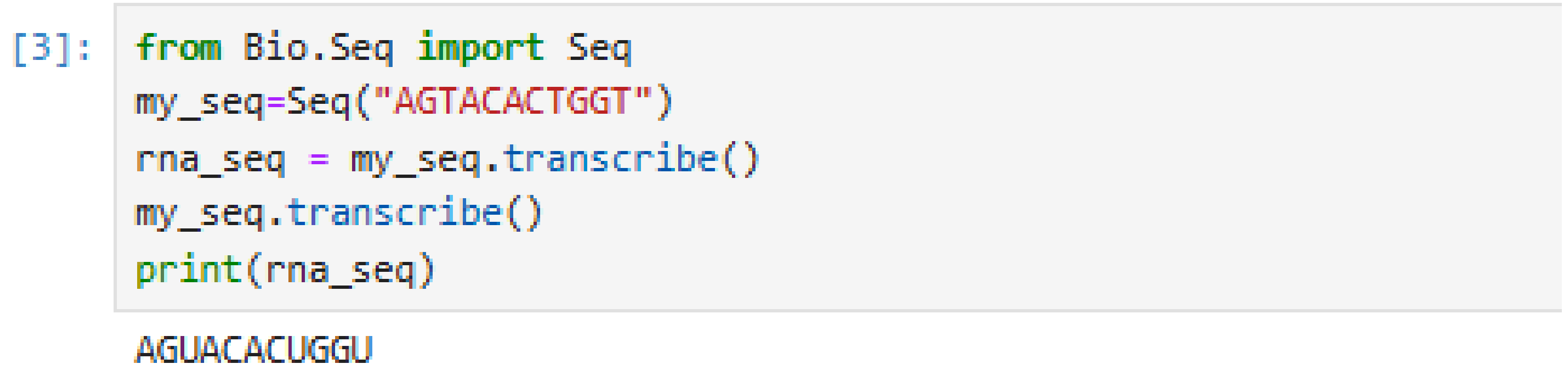
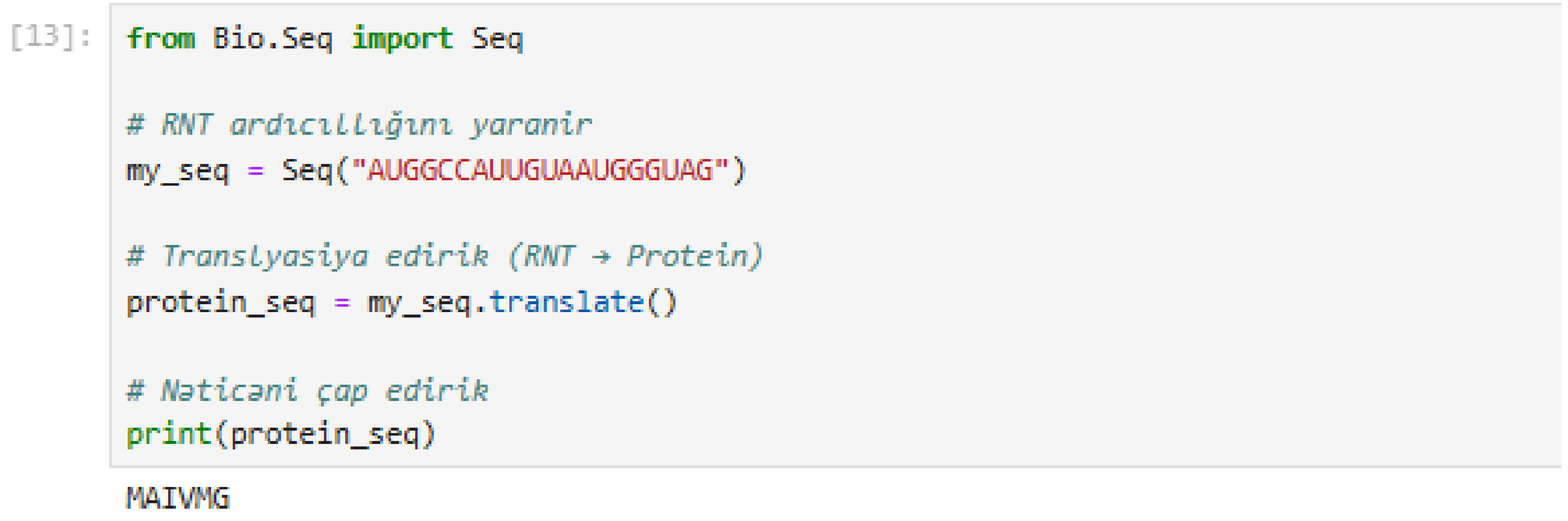
Task 1. Transcription to – Use the transcribe() method to transcribe a given DNA sequence use to the RNA sequence by translate . For example : Seq ("AGTACACTGGT").transcribe()



```
[3]: from Bio.Seq import Seq
     my_seq=Seq("AGTACACTGGT")
     rna_seq = my_seq.transcribe()
     my_seq.transcribe()
     print(rna_seq)

     AGUACACUGGU
```

- **Seq ("AGTACACTGGT")** – This is the DNA sequence creates .
- **transcribe()** – This method **T** the base **U** with replacement RNA sequencing creates .
- **print(rna_seq)** – Transcription RNA sequence to the screen print does .

Task 2. Broadcast to – Translate a given RNA sequence using the translate() method use by doing relevant to protein sequence translate . For example : Seq ("AUGGCCAUUGUAAUGGGUAG").translate()



```
[13]: from Bio.Seq import Seq

      # RNT ardıcıllığını yaranir
      my_seq = Seq("AUGGCCAUUGUAAUGGGUAG")

      # Translyasiya edirik (RNT → Protein)
      protein_seq = my_seq.translate()

      # Nəticəni çap edirik
      print(protein_seq)
```

```
MAIVMG
```

Description :

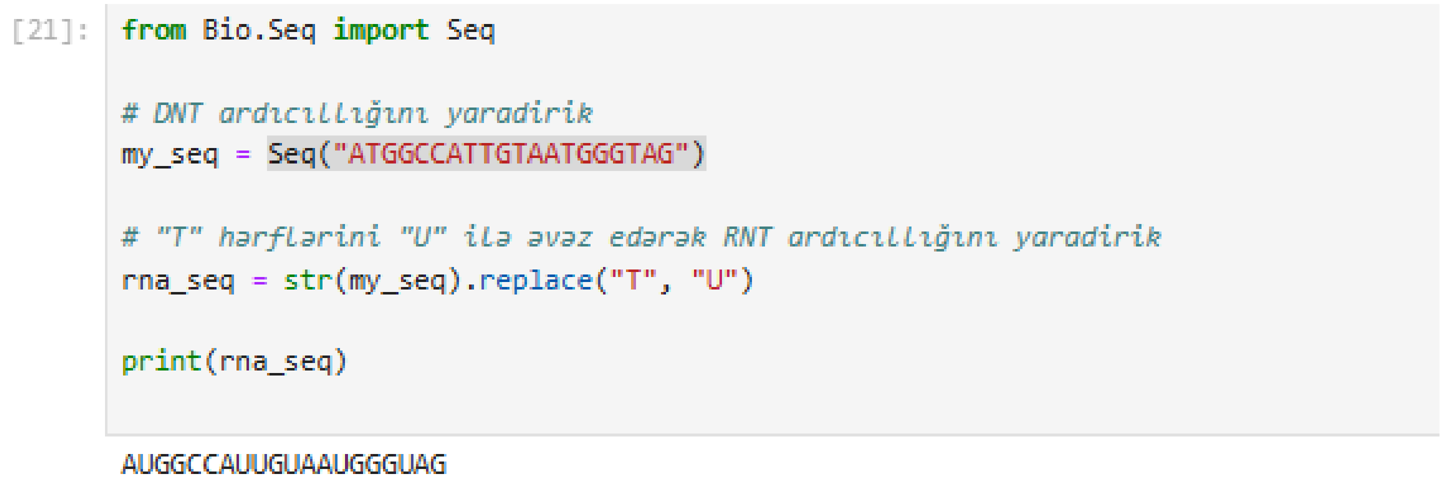
1.Seq ("AUGGCCAUUGUAAUGGGUAG") – It is an RNA sequence .

2.translate() – This method 3 codons amen to acids converts .

3.print(protein_seq) – Protein sequence print does .

- **M** → Methionine (AUG start codon)
- **A** → Alanine (GCC)
- **P** → Proline (AUU)
- **I** → Isoleucine (GUA)
- **Stop codon** → * symbol with is displayed)

Task 3. Difference between DNA and RNA – from the replace() method use by replacing the letters T in the DNA sequence with U replacement do and the result print please .



```
[21]: from Bio.Seq import Seq

      # DNT ardıcıllığını yaradirik
      my_seq = Seq("ATGGCCATTGTAATGGGTAG")

      # "T" hərflərini "U" ilə əvəz edərək RNT ardıcıllığını yaradirik
      rna_seq = str(my_seq).replace("T", "U")

      print(rna_seq)
```

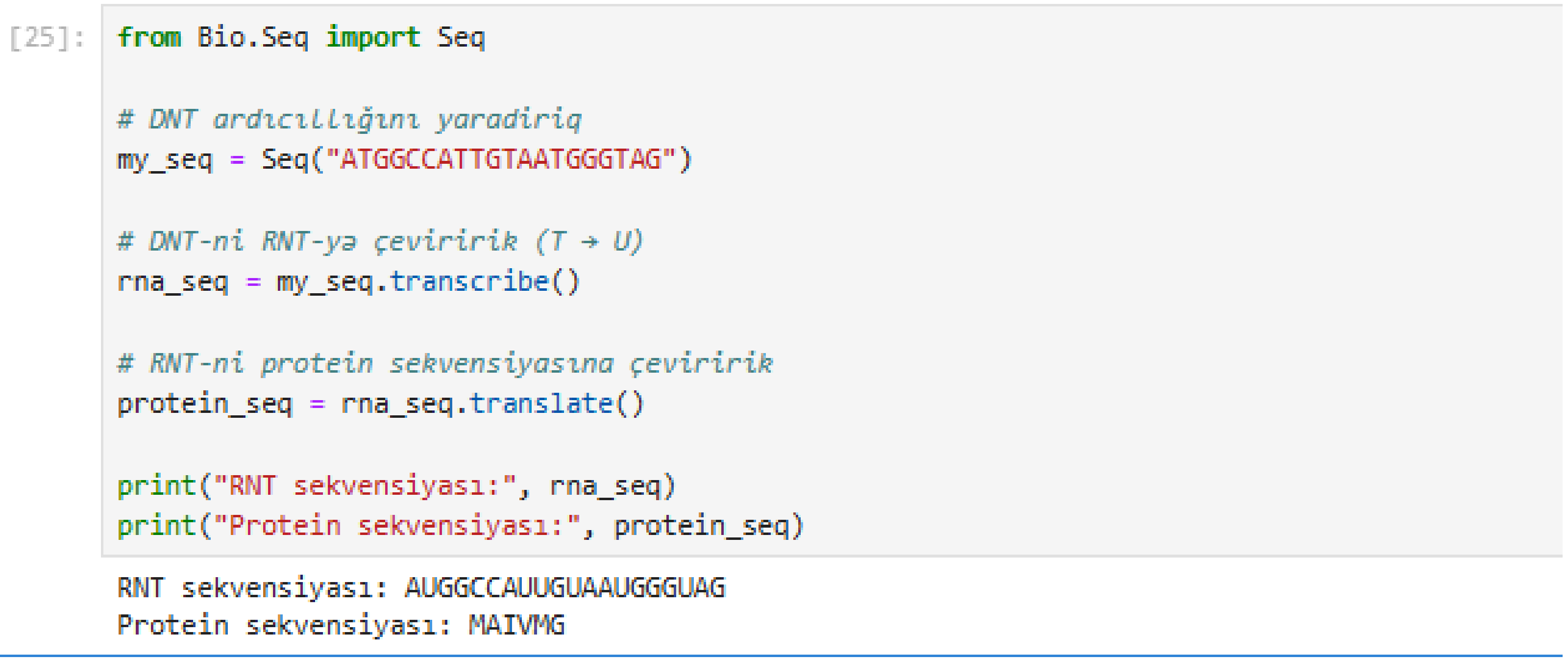
```
AUGGCCAUUGUAAUGGGUAG
```

Biopython 's Seq replace() method on object no , because **Seq Python str object like directly changeable can't** .

This instead , in the DNA sequence **"T"** letters **"U"** with replacement to do for From **the Python str.replace () method use can you can** . But this street to format by turning you have to do .

Description :
1.Seq ("ATGGCCATTGTAATGGGTAG") – DNA sequence creates .
2.str (my_seq).replace("T", "U") – str () method with Convert **DNA sequence to string format** converts and then **"T" letters with "U" replacement does** .
3.print(rna_seq) – Obtained RNA sequencing print does .

Task 4. DNA sequencing amen to acids convert – Convert a given DNA sequence first to RNA , then and protein sequence turn it over .



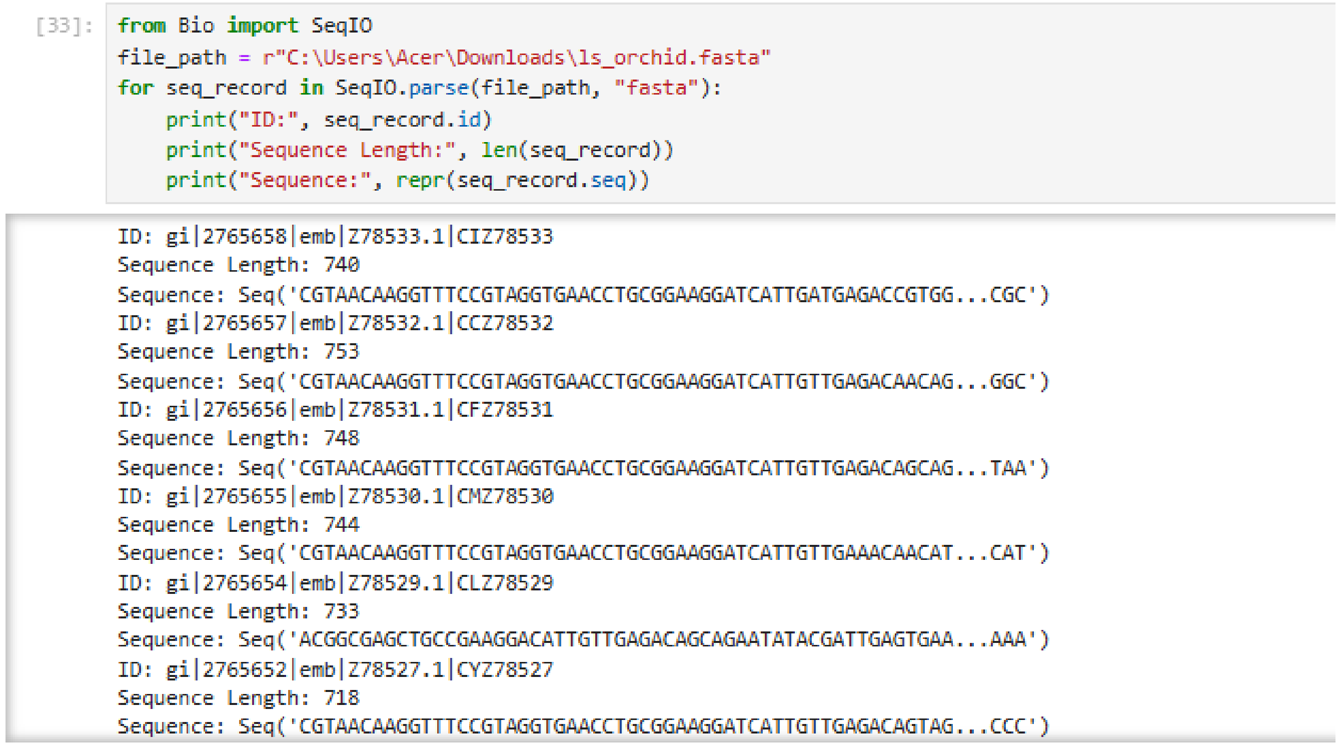
```
[25]: from Bio.Seq import Seq

      # DNT ardıcıllığını yaradiriq
      my_seq = Seq("ATGGCCATTGTAATGGGTAG")

      # DNT-ni RNT-yə çeviririk (T → U)
      rna_seq = my_seq.transcribe()

      # RNT-ni protein sekvensiyasına çeviririk
      protein_seq = rna_seq.translate()

      print("RNT sekvensiyası:", rna_seq)
      print("Protein sekvensiyası:", protein_seq)
```

```
RNT sekvensiyası: AUGGCCAUUGUAAUGGGUAG
Protein sekvensiyası: MAIVMG
```

- **"AUG"** → Methionine (**M** , start codon)
- **"GCC"** → Alanine (**A**)
- **"AUU"** → Isoleucine (**I**)
- **"UGU"** → Cysteine (**C**)
- **"AAU"** → Asparagine (**N**)
- **"GGG"** → Glycine (**G**)
- **"UAG"** → STOP codon (**"*"** with is displayed)

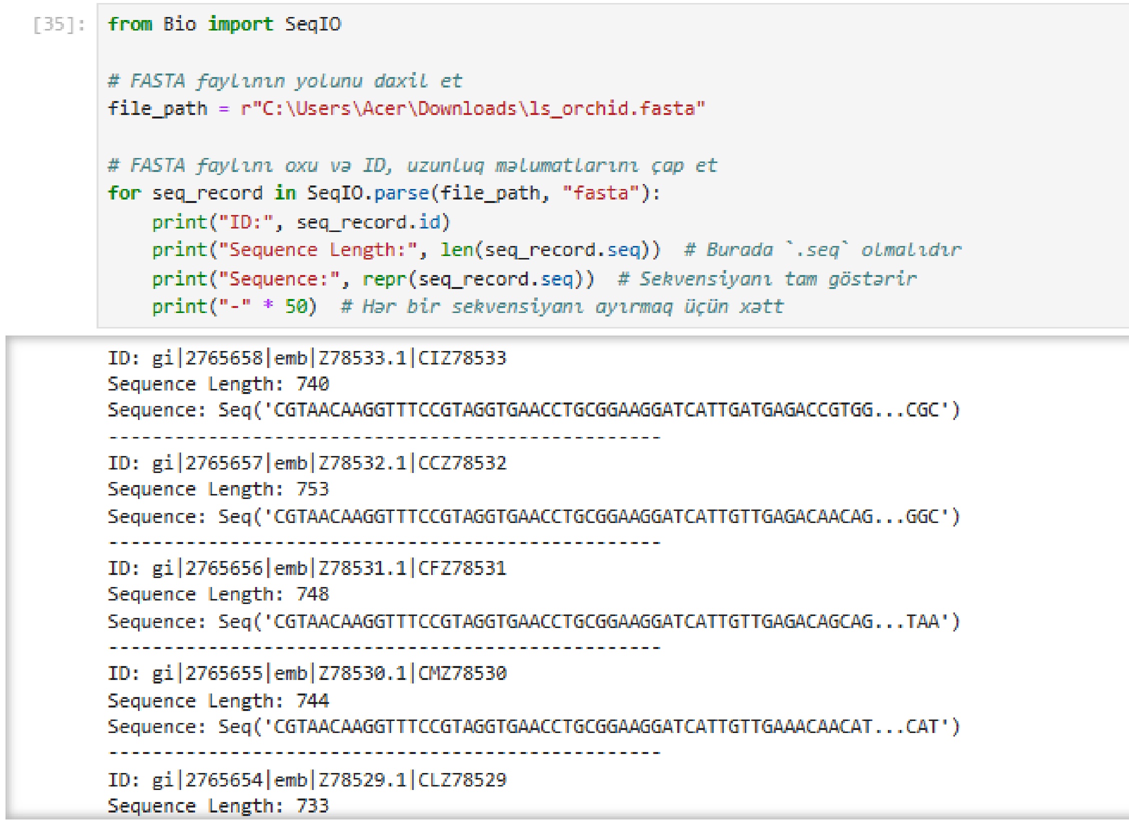
- **Seq ("ATGGCCATTGTAATGGGTAG")** – DNA sequence Seq at the facility appointment does .
- **transcribe()** – transcribes DNA into RNA converts (T bases to U replacement does).
- **translate()** – RNA amen to acids converts .
- **print()** – Both RNA and also protein sequences print does .

Task 5. From the FASTA File Information Extract – From the SeqIO.parse () method use by doing . fast in the file which all sequence IDs and their lengths print please .



```
[33]: from Bio import SeqIO
      file_path = r"C:\Users\Acer\Downloads\ls_orchid.fasta"
      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))
```

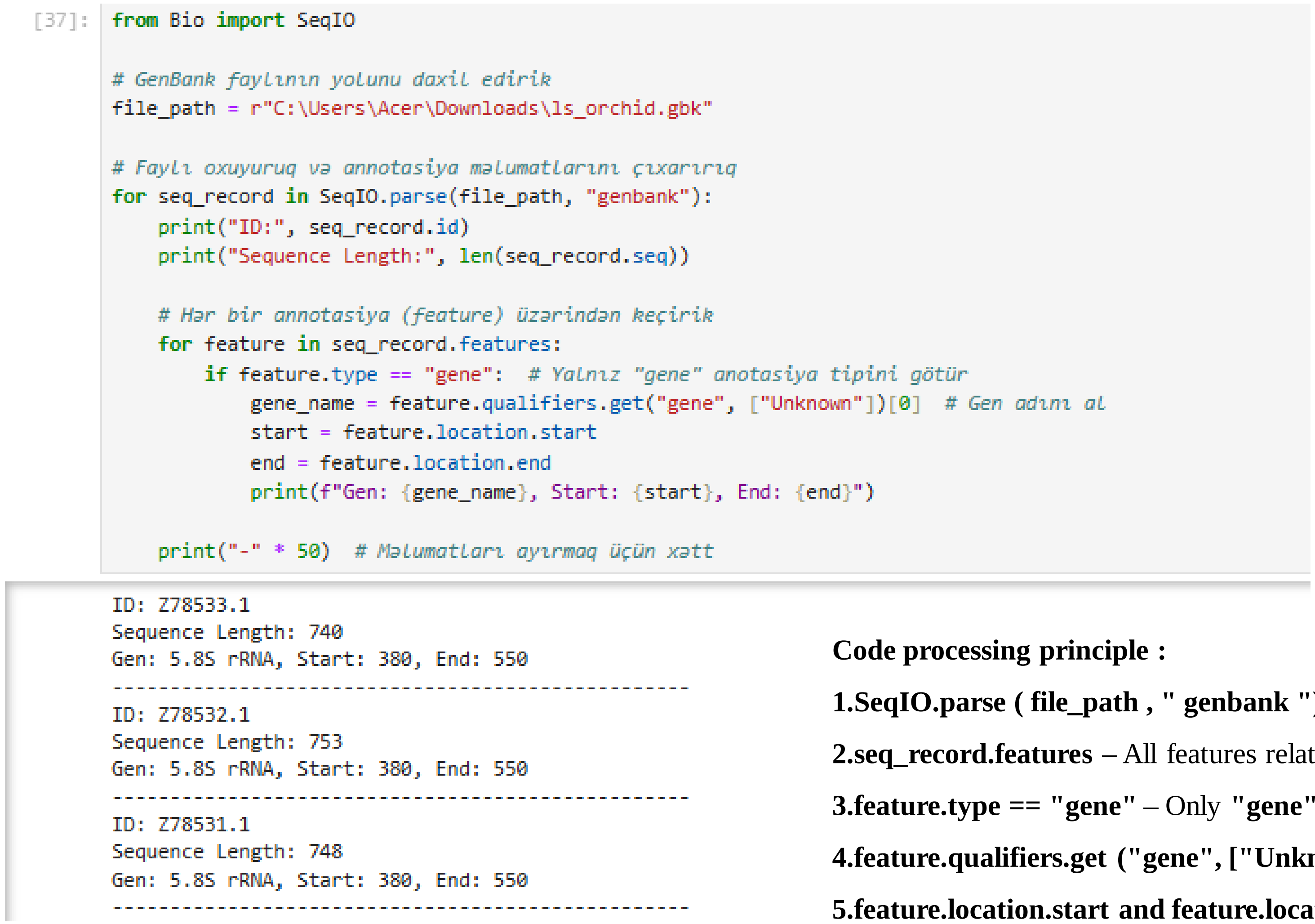
```
ID: gi|2765658|emb|Z78533.1|CIZ78533
Sequence Length: 740
Sequence: Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC')
ID: gi|2765657|emb|Z78532.1|CCZ78532
Sequence Length: 753
Sequence: Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAG...GGC')
ID: gi|2765656|emb|Z78531.1|CFZ78531
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')
```



```
[35]: from Bio import SeqIO

      # FASTA faylının yolunu daxil et
      file_path = r"C:\Users\Acer\Downloads\ls_orchid.fasta"

      # FASTA faylını oxu və ID, uzunluq məlumatlarını çap et
      for seq_record in SeqIO.parse(file_path, "fasta"):
          print("ID:", seq_record.id)
          print("Sequence Length:", len(seq_record.seq))  # Burada `.seq` olmalıdır
          print("Sequence:", repr(seq_record.seq))  # Sekvensiyanı tam göstərir
          print("-" * 50)  # Hər bir sekvensiyanı ayırmaq üçün xətt
```

```
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
```

Task 6. GenBank From file Annotations – . gbk file by reading , genes their names and their start-stop positions remove .



```
[37]: from Bio import SeqIO

      # GenBank faylının yolunu daxil edirik
      file_path = r"C:\Users\Acer\Downloads\ls_orchid.gbk"

      # Faylı oxuyuruq və annotasiya məlumatlarını çıxarırıq
      for seq_record in SeqIO.parse(file_path, "genbank"):
          print("ID:", seq_record.id)
          print("Sequence Length:", len(seq_record.seq))

          # Hər bir annotasiya (feature) üzərindən keçirik
          for feature in seq_record.features:
              if feature.type == "gene":  # Yalnız "gene" anotasiya tipini götür
                  gene_name = feature.qualifiers.get("gene", ["Unknown"])[0]  # Gen adını al
                  start = feature.location.start
                  end = feature.location.end
                  print(f"Gen: {gene_name}, Start: {start}, End: {end}")

          print("-" * 50)  # Məlumatları ayırmaq üçün xətt
```

```
ID: Z78533.1
Sequence Length: 740
Gen: 5.8S rRNA, Start: 380, End: 550
--------------------------------------------------
ID: Z78532.1
Sequence Length: 753
Gen: 5.8S rRNA, Start: 380, End: 550
--------------------------------------------------
ID: Z78531.1
Sequence Length: 748
Gen: 5.8S rRNA, Start: 380, End: 550
--------------------------------------------------
```

Code processing principle :

1.SeqIO.parse (file_path , " genbank ") – GenBank file is reading .

2.seq_record.features – All features related to the sequence annotations keeper It is a list .

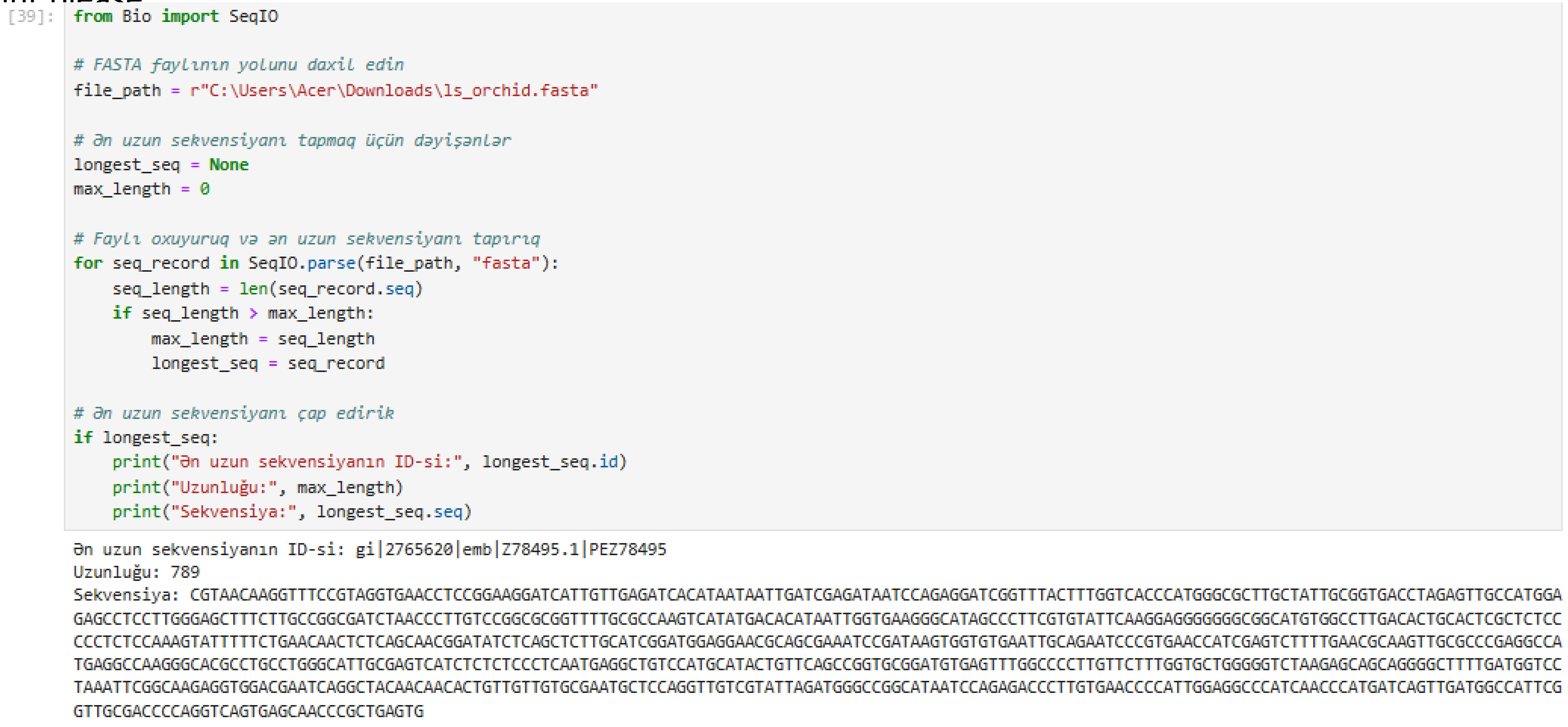
3.feature.type == "gene" – Only **"gene"** annotation type (others , for example , CDS and either tRNA into account not ava

4.feature.qualifiers.get ("gene", ["Unknown"])[0] – Gene name takes (if the gene name if not , as "Unknown" shows).

5.feature.location.start and feature.location.end – Gene **beginning and ending coordinates** removes .

6.Print made information every sequencing for separately is displayed .

Task 7. In the FASTA File The most Long Sequence Find – . fasta in the file which from sequences the most long one choose and print please



```
[39]: from Bio import SeqIO

      # FASTA faylının yolunu daxil edin
      file_path = r"C:\Users\Acer\Downloads\ls_orchid.fasta"

      # Ən uzun sekvensiyanı tapmaq üçün dəyişənlər
      longest_seq = None
      max_length = 0

      # Faylı oxuyuruq və ən uzun sekvensiyanı tapırıq
      for seq_record in SeqIO.parse(file_path, "fasta"):
          seq_length = len(seq_record.seq)
          if seq_length > max_length:
              max_length = seq_length
              longest_seq = seq_record

      # Ən uzun sekvensiyanı çap edirik
      if longest_seq:
          print("Ən uzun sekvensiyanın ID-si:", longest_seq.id)
          print("Uzunluğu:", max_length)
          print("Sekvensiya:", longest_seq.seq)
```

```
Ən uzun sekvensiyanın ID-si: gi|2765620|emb|Z78495.1|PEZ78495
Uzunluğu: 789
Sekvensiya: CGTAACAAGGTTTCCGTAGGTGAACCTCCGGAAGGATCATTGTTGAGATCACATAATAATTGATCGAGATAATCCAGAGGATCGGTTTACTTTGGTCACCCATGGGCGCTTGCTATTGCGGTGACCTAGAGTTGCCATGGA
GAGCCTCCTTGGGAGCTTTCTTGCCGGCGATCTAACCCTTGTCCGGCGCGGTTTTGCGCCAAGTCATATGACACATAATTGGTGAAGGGCATAGCCCTTCGTGTATTCAAGGAGGGGGGGCGGCATGTGGCCTTGACACTGCACTCGCTCTCC
CCCTCTCCAAAGTATTTTTCTGAACAACTCTCAGCAACGGATATCTCAGCTCTTGCATCGGATGGAGGAACGCAGCGAAATCCGATAAGTGGTGTGAATTGCAGAATCCCGTGAACCATCGAGTCTTTTGAACGCAAGTTGCGCCCGAGGCCA
TGAGGCCAAGGGCACGCCTGCCTGGGCATTGCGAGTCATCTCTCTCCCTCAATGAGGCTGTCCATGCATACTGTTCAGCCGGTGCGGATGTGAGTTTGGCCCCTTGTTCTTTGGTGCTGGGGGTCTAAGAGCAGCAGGGGCTTTTGATGGTCC
TAAATTCGGCAAGAGGTGGACGAATCAGGCTACAACAACACTGTTGTTGTGCGAATGCTCCAGGTTGTCGTATTAGATGGGCCGGCATAATCCAGAGACCCTTGTGAACCCCATTGGAGGCCCATCAACCCATGATCAGTTGATGGCCATTCG
GTTGCGACCCCAGGTCAGTGAGCAACCCGCTGAGTG
```

Code processing principle :

1.SeqIO.parse (file_path , " fasta ") – parse FASTA file is reading .

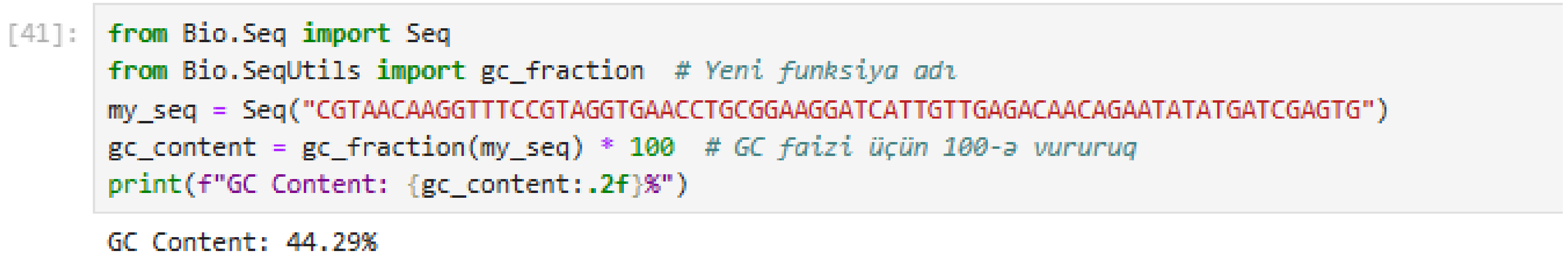
2.max_length and longest_seq variables – Most long sequencing and the length to keep for use is done .

3.Loop with all to sequences we look at :

- Every sequence We check the length (len (seq_record.seq)) .
- If current sequencing **for now until found the most if it is long** , it remember we keep .

4.The most long sequencing print we are doing .

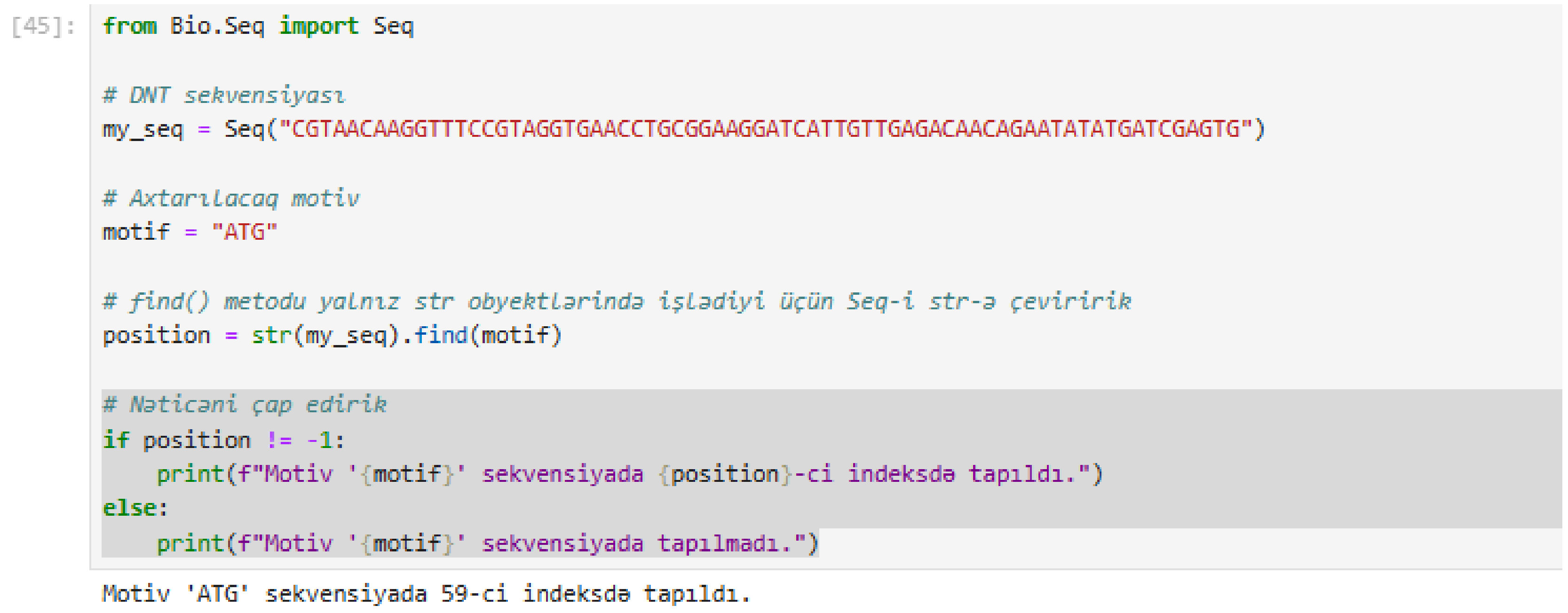
Task 8. GC content calculate – from the gc_fraction () method use GC content of the DNA sequence by with interest calculate .



```
[41]: 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%
```

Task 9. DNA sequencing motive find – from the find() method use by doing in a given DNA sequence certain one whether the motif (e.g. "ATG") is present check .



```
[45]: from Bio.Seq import Seq

      # DNT sekvensiyası
      my_seq = Seq("CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG")

      # Axtarılacaq motiv
      motif = "ATG"

      # find() metodu yalnız str obyektlərində işlədiyi üçün Seq-i str-ə çeviririk
      position = str(my_seq).find(motif)

      # Nəticəni çap edirik
      if position != -1:
          print(f"Motiv '{motif}' sekvensiyada {position}-ci indeksdə tapıldı.")
      else:
          print(f"Motiv '{motif}' sekvensiyada tapılmadı.")
```

```
Motiv 'ATG' sekvensiyada 59-ci indeksdə tapıldı.
```

Code processing principle :

1.str (my_seq).find(motif)

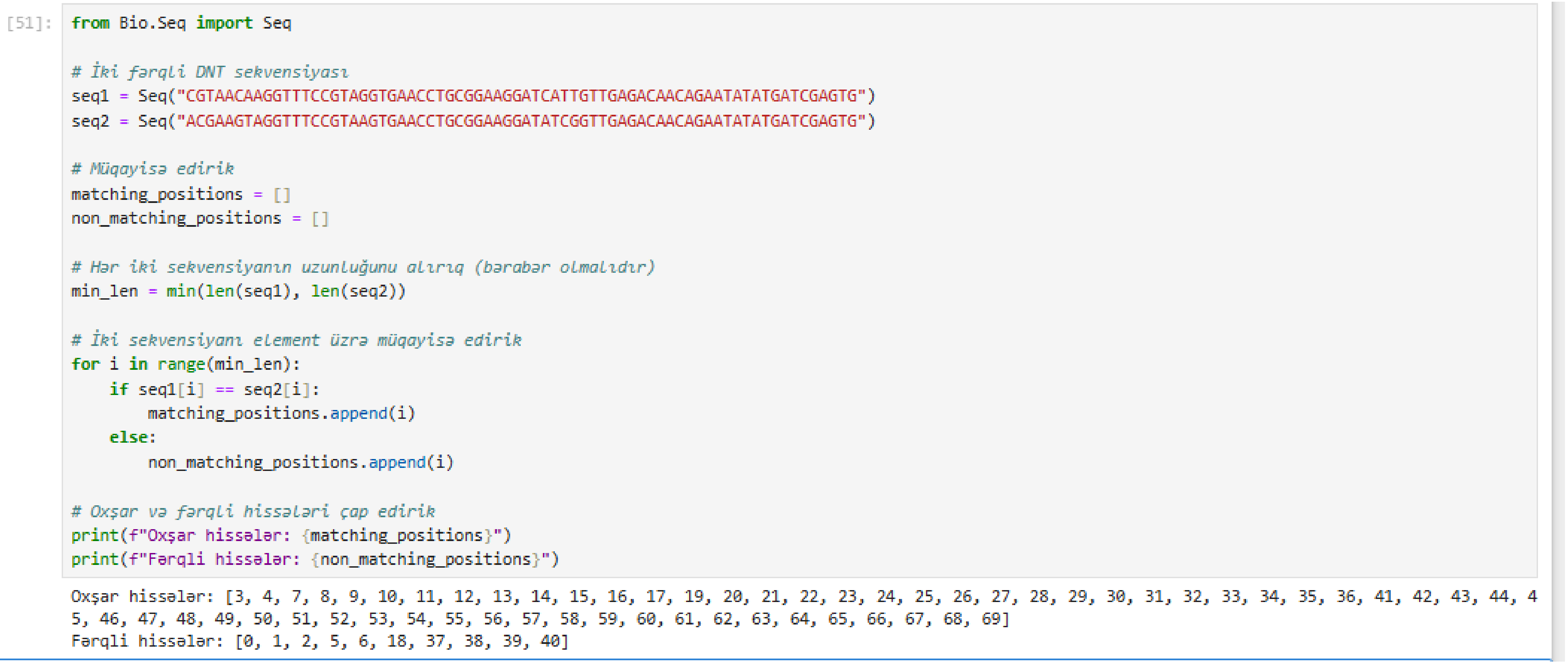
- Seq object street to format we convert (str (my_seq)).
- find(motif) function motive index returns .
- If motive If not , it returns -1 .

2.Condition checked (if position != -1)

- If if the index is not -1 , the motif **first found place** print we are doing .
- Reverse in case of " not found " message We show .

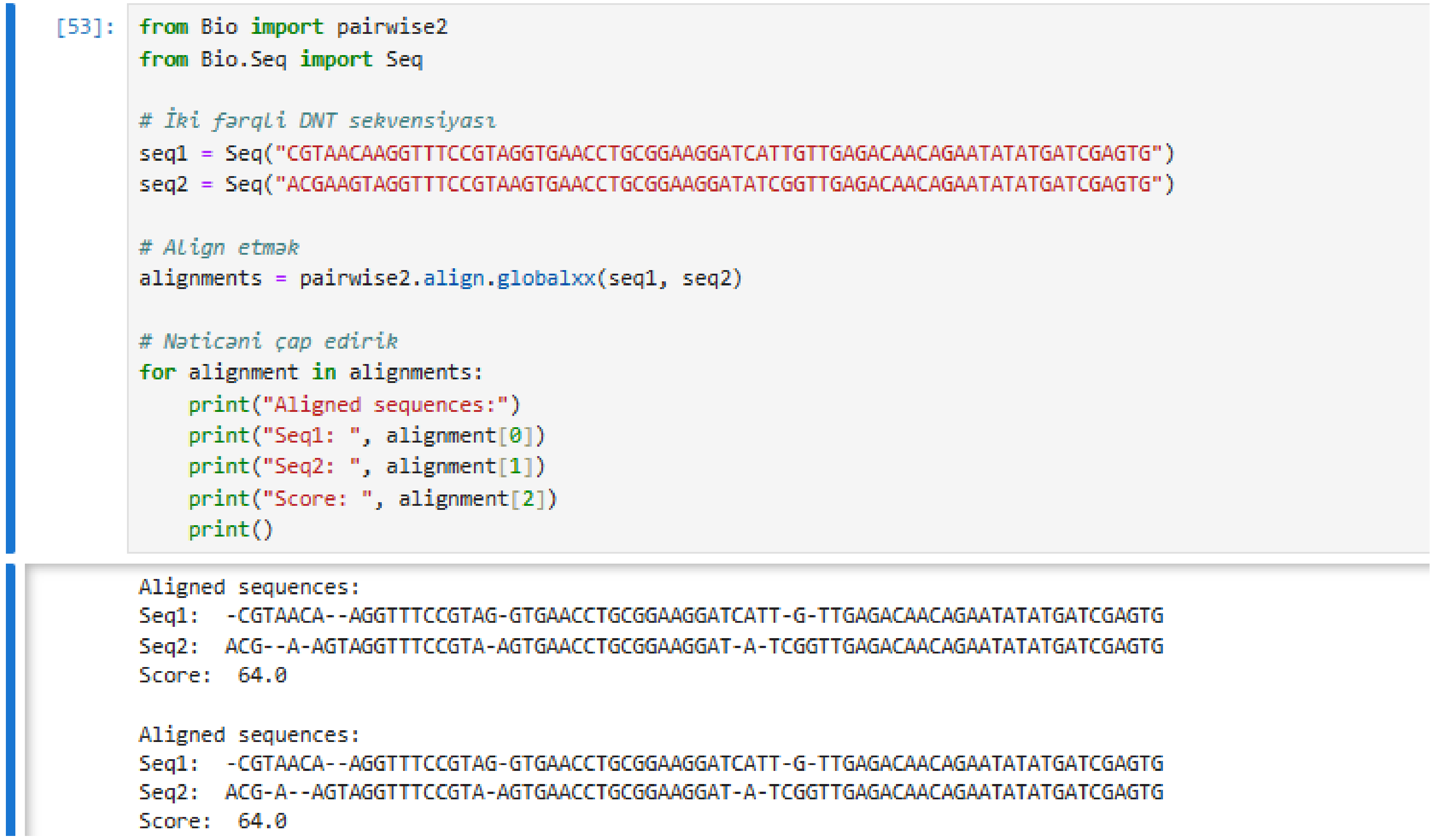
Task 10. DNA sequences comparison to do – Two different DNA sequence comparison by doing similar and different parts print please .

Two different DNA sequence comparison while doing **similar** and **different** parts to find for align() and either just the == operator use can You can . This for **Seq from the objects** and or from the pairwise2 module use to do It will be .



```
[51]: from Bio.Seq import Seq

      # İki fərqli DNT sekvensiyası
      seq1 = Seq("CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG")
      seq2 = Seq("ACGAAGTAGGTTTCCGTAAGTGAACCTGCGGAAGGATATCGGTTGAGACAACAGAATATATGATCGAGTG")

      # Müqayisə edirik
      matching_positions = []
      non_matching_positions = []

      # Hər iki sekvensiyanın uzunluğunu alırıq (bərabər olmalıdır)
      min_len = min(len(seq1), len(seq2))

      # İki sekvensiyanı element üzrə müqayisə edirik
      for i in range(min_len):
          if seq1[i] == seq2[i]:
              matching_positions.append(i)
          else:
              non_matching_positions.append(i)

      # Oxşar və fərqli hissələri çap edirik
      print(f"Oxşar hissələr: {matching_positions}")
      print(f"Fərqli hissələr: {non_matching_positions}")
```

```
Oxşar hissələr: [3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 41, 42, 43, 44, 4
5, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69]
Fərqli hissələr: [0, 1, 2, 5, 6, 18, 37, 38, 39, 40]
```



```
[53]: from Bio import pairwise2
      from Bio.Seq import Seq

      # İki fərqli DNT sekvensiyası
      seq1 = Seq("CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG")
      seq2 = Seq("ACGAAGTAGGTTTCCGTAAGTGAACCTGCGGAAGGATATCGGTTGAGACAACAGAATATATGATCGAGTG")

      # Align etmək
      alignments = pairwise2.align.globalxx(seq1, seq2)

      # Nəticəni çap edirik
      for alignment in alignments:
          print("Aligned sequences:")
          print("Seq1: ", alignment[0])
          print("Seq2: ", alignment[1])
          print("Score: ", alignment[2])
          print()
```

```
Aligned sequences:
Seq1:  -CGTAACA--AGGTTTCCGTAG-GTGAACCTGCGGAAGGATCATT-G-TTGAGACAACAGAATATATGATCGAGTG
Seq2:  ACG--A-AGTAGGTTTCCGTA-AGTGAACCTGCGGAAGGAT-A-TCGGTTGAGACAACAGAATATATGATCGAGTG
Score:  64.0

Aligned sequences:
Seq1:  -CGTAACA--AGGTTTCCGTAG-GTGAACCTGCGGAAGGATCATT-G-TTGAGACAACAGAATATATGATCGAGTG
Seq2:  ACG-A--AGTAGGTTTCCGTA-AGTGAACCTGCGGAAGGAT-A-TCGGTTGAGACAACAGAATATATGATCGAGTG
Score:  64.0
```

Code processing principle :

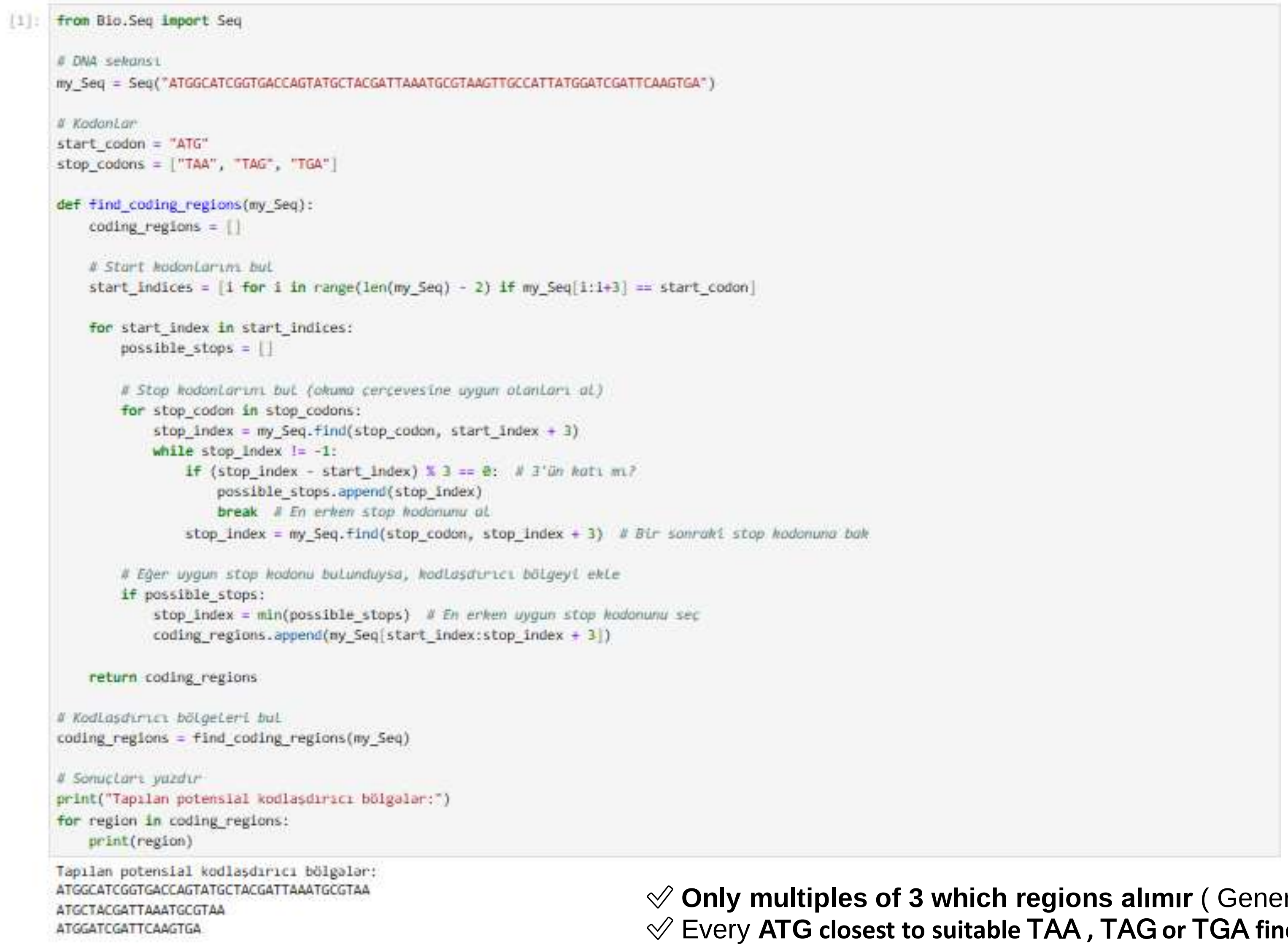
1.Simple comparison :

- •DNA sequences positions on comparison we are doing .
- •Similar and different which indices **to the list** additional we are doing .
- •As a result similar and different parts We are taking out .

2.Pairwise alignment:

- •pairwise2.align.globalxx(seq1, seq2) function , **more exactly** one in the way two sequencing **align** does and the most good finds the match .
- •At the exit **aligned** sequences and **result value** is displayed .

Task 11. Given In DNA sequencing potential encoder regions (ATG start codon with TAA , TAG and or TGA stop codons with ending Find the regions .



```
[1]: from Bio.Seq import Seq

     # DNA sekansı
     my_Seq = Seq("ATGGCATCGGTGACCAGTATGCTACGATTAAATGCGTAAGTTGCCATTATGGATCGATTCAAGTGA")

     # Kodonlar
     start_codon = "ATG"
     stop_codons = ["TAA", "TAG", "TGA"]

     def find_coding_regions(my_Seq):
         coding_regions = []

         # Start kodonlarını bul
         start_indices = [i for i in range(len(my_Seq) - 2) if my_Seq[i:i+3] == start_codon]

         for start_index in start_indices:
             possible_stops = []

             # Stop kodonlarını bul (okuma çerçevesine uygun olanları al)
             for stop_codon in stop_codons:
                 stop_index = my_Seq.find(stop_codon, start_index + 3)
                 while stop_index != -1:
                     if (stop_index - start_index) % 3 == 0:  # 3'ün katı mı?
                         possible_stops.append(stop_index)
                         break  # En erken stop kodonunu al
                     stop_index = my_Seq.find(stop_codon, stop_index + 3)  # Bir sonraki stop kodonuna bak

             # Eğer uygun stop kodonu bulunduysa, kodlaşdırıcı bölgeyi ekle
             if possible_stops:
                 stop_index = min(possible_stops)  # En erken uygun stop kodonunu seç
                 coding_regions.append(my_Seq[start_index:stop_index + 3])

         return coding_regions

     # Kodlaşdırıcı bölgeleri bul
     coding_regions = find_coding_regions(my_Seq)

     # Sonuçları yazdır
     print("Tapılan potensial kodlaşdırıcı bölgələr:")
     for region in coding_regions:
         print(region)
```

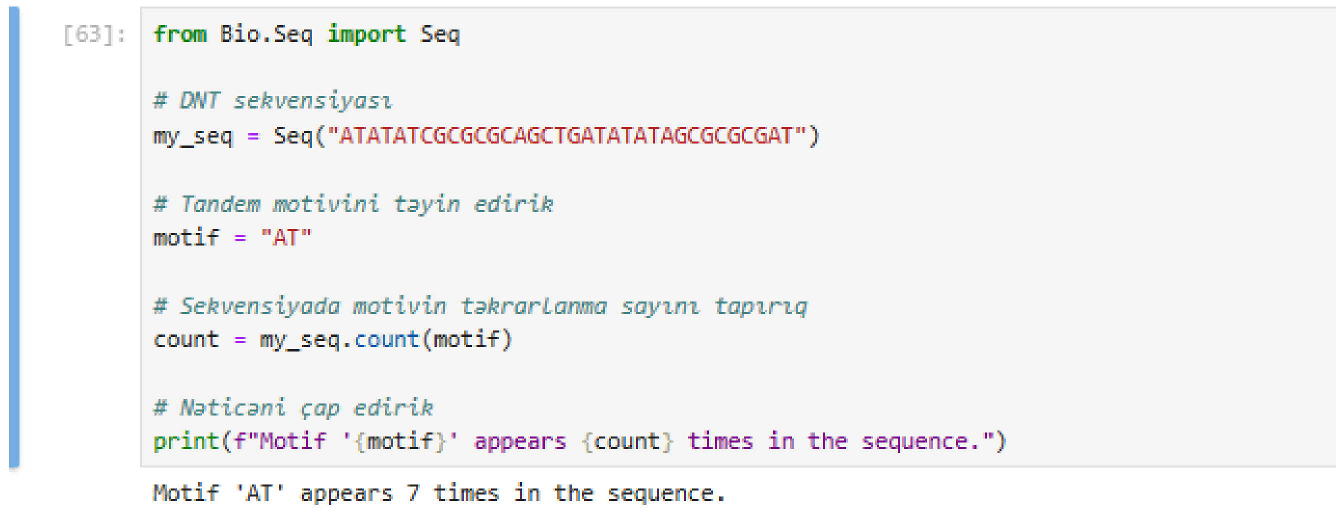
```
Tapılan potensial kodlaşdırıcı bölgələr:
ATGGCATCGGTGACCAGTATGCTACGATTAAATGCGTAA
ATGCTACGATTAAATGCGTAA
ATGGATCGATTCAAGTGA
```

✓ **Only multiples of 3 which regions alımır** (Generek gene structure more true simulate) .
✓ Every **ATG closest to suitable TAA , TAG or TGA finds** (More true analysis does).
✓ **Reading frame violation stop codons that do not evaluates**
✓ **The earliest the correct stop codon takes** , unnecessary scanning doesn't do it .

Task 12. Tandem repeats in DNA sequencing motives (i.e. consecutive as one how many times recurring short nucleotide find the sequences and their repetition number certain please .

- **Tandem repeat motifs:** Motifs that are repeated several times in a row in a DNA sequence.
- **Example motif:** For example, "AT" or longer sequences.

- **count() method :** This method is used to count a certain number of one of the motif (here "AT") in the sequence how many times finds it repeating .
- **Simple DNA sequence :** For example , the "AT" motif on ATATATCGCGCGCAGCTGATATATAGCGCGCGAT how many times repetition we find .



```
[63]:
from Bio.Seq import Seq

# DNT sekvensiyası
my_seq = Seq("ATATATCGCGCGCAGCTGATATATAGCGCGCGAT")

# Tandem motivini təyin edirik
motif = "AT"

# Sekvensiyada motivin təkrarlanma sayını tapırıq
count = my_seq.count(motif)

# Nəticəni çap edirik
print(f"Motif '{motif}' appears {count} times in the sequence.")
```

```
Motif 'AT' appears 7 times in the sequence.
```

Finding a Tandem Recurring Motif:
the count() method for longer motifs . However, this approach is ideal for finding the number and positions of motif repetitions.