Biopython lesson 2

* **What? Does it tell ?**

ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG DNA sequence **protein encoder one gene coding** This DNA sequence **is part** of

transcription and translation process as a result **amino acid to the chain** turns into .

So one result is taken :

ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG (DNA) ↓ Transcription (mRNA synthesis)

AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG (mRNA) ↓ Translation (Amino acid sequence) MAIVMGR*KGAR*

Here *** sign STOP codon** nu (stop codon) indicates .

STOP Codons What is it ?

Protein synthesis **codons (triplets) nucleotide groups)** through management Genetic in the code **3 types of STOP codons** there is :

•**UAA** (**Ochre**)

•**UAG** (**Amber**)

•**UGA** (**Opal**)

these codons one ribosome by when read , **protein synthesis stands** .

STOP Codon How Is it happening ?

Your DNA sequence let's see :

ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG (DNA)

AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG (mRNA)

Here **The UAG codon is** a STOP codon .

UAG ("Amber Stop Codon ") is called and **protein synthesis stops** .

Summary

✓ *** STOP codon means** .

✓ **UAA, UAG, UGA protein synthesis stopping are codons** .

✓ Of these codons one As for **the ribosome** , **protein completes and him/her free leaves** .

That is , the *** symbol indicates protein synthesis. there that it's over show**

Genetic Code : Standard Code and Vertebrates Mitochondrial Code Between Differences

Genetic code , **DNA and or in RNA codons (triplets) nucleotides)** which one amen to acids suitable that he came certain who rules is a collection of .

Different organisms and organelles (e.g. , mitochondria) various genetic from codes use can can .

Genetics in **Biopython** code tables **transl_table** with is shown .

The most very use done two code :

- **Standard Genetic Code (The Standard Code, transl_table =1)**
- **Vertebrates Mitochondrial Code (The Vertebrate Mitochondrial Code, transl_table =2)**

Home Differences

Feature	Standard Code (Table 1)	Mitochondrial Code (Table 2)
Start Codon	AUG (M)	AUG, AUA, AUU (M)
Stop Codons	UAA, UAG, UGA	UAA, UAG (but UGA = W)
UGA codon	STOP	Tryptophan (W)
AUA codon	Isoleucine (I)	Methionine (M)

https:// www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi

1. Standard Genetic Code (transl_table =1)

All cell in the nucleus (nucleus genome) use done main code This is it .

Bacteria , archaea and nuclear to the genome possessor which **animal , plant and mushrooms** by use is being done .

📌 **Features :**

✓ **Start codon :** AUG (Methionine, M)

✓ **Stop codons :** UAA, UAG, UGA

✓ All other codons **standard amen acids encodes** .

2. Vertebrates Mitochondrial Code (transl_table =2)

In mitochondria different evolution way because according to **mitochondrial DNA (mtDNA) from nuclear DNA different genetic from the code use does .**

📌 **Features :**

✓ **Start codons :** AUA, AUG, AUU (all encode Methionine, M)

✓ **Stop codons is different :**

•**UGA stop codon not** , **Tryptophan (W)** coding !

•Stop codons **only UAA and UAG** .

Biopython with Usage

You Biopython with **various translation from the tables use can you can** .

Standard Code Translation (Table 1)

from Bio.Seq import Seq dna = Seq ("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG") protein = dna.translate (table=1) print(protein)

📌 **Exit :**

markdown

MAIVMGR*KGAR*

Here * STOP codon shows .

Conclusion

✓ **Standard code** cell in the core use ✓ Mitochondrial

code in mitochondria use is done and some differences there is :

•**UGA encodes W (Tryptophan), not STOP .**

•**encodes Methionine (M) , in nuclear DNA and Isoleucine (I).**

•**Stop codons It's just UAA and UAG .**

Mitochondrial Code Translation (Table 2)

protein_mito = dna.translate (table=2) print(protein_mito)

📌 Difference :

If the UGA codon If there is , **it will be W (Tryptophan) instead of STOP .**

Mitochondrial Code Translation Result : MAIVMGRWKGAR* What Does that mean ?

Mitochondrial genetic from the code (Vertebrate Mitochondrial Code) use by doing broadcast Have you done it ?

print(coding_dna.translate (table="Vertebrate Mitochondrial"))

Output **:**

MAIVMGRWKGAR*

```
#back-transcription method
messenger_rna=Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG")
print(messenger_rna)
print(messenger_rna.back_transcribe())
print(messenger_rna.translate())
#ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG burdan ede bilirsen...
```

```
AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG
ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG
MAIVMGR*KGAR*
```

1. Standard Code with Difference What is it ?

Previous standard genetic in the code output so was :

MAIVMGR*KGAR*

But **mitochondrial code** result :

MAIVMGRWKGAR*

Here **main difference W (Tryptophan) amine It is acidic .**

2. The UGA Codon Encodes

Standard UGA **STOP** codon **in the code** where for **protein early stands .**

But **vertebrates mitochondrial in the code** UGA **is not STOP** , instead **Tryptophan (W) encodes .**

DNA sequence :

ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG (DNA)

AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG (mRNA)

Standard code translation :

MAIVMGR*KGAR* (UGA STOP)

Mitochondrial code translation :

MAIVMGRWKGAR* (UGA → W)

Here **UGA is no longer a STOP codon No** , it **encodes Tryptophan (W**

```
#veya
coding_dna=Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG")
print(coding_dna)
print(coding_dna.translate())
```

```
ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG
MAIVMGR*KGAR*
```

```
print(coding_dna.translate(table="Vertebrate Mitochondrial"))
#veya
coding_dna.translate(table=2)
```

```
MAIVMGRWKGAR*
Seq('MAIVMGRWKGAR*')
```

3. Conclusion

✓ **Vertebrate mitochondrial code** UGA codon **Like STOP no , like Tryptophan (W) is reading .**

✓ For **this reason standard code received Instead of STOP in the result , W (Tryptophan) appears .**

✓ Other codons the same remains .

To him/her according to also the output becomes MAIVMGRWKGAR* .

Deletion and addition of stop codons

```
print(coding_dna.translate())
#Seq("MAIVMGR*KGAR*")
print(coding_dna.translate(to_stop=True))
```

```
MAIVMGR*KGAR*
MAIVMGR
```

```
print(coding_dna.translate(table=2))
```

```
MAIVMGRWKGAR*
```

```
print(coding_dna.translate(table=2, to_stop=True))
```

```
MAIVMGRWKGAR
```

This code **BioPython** from the library use by doing **DNA sequence (coding_dna) amen to acids translates** (translates) does).

print(coding_dna.translate ())

➞ **Standard genetic** DNA sequence **with code** amen to acids turns into .

Output :

MAIVMGR*KGAR*

Asterisk (*) indicates STOP codons shows .

STOP codon , protein synthesis where it stands It is a place .

print(coding_dna.translate (to_stop =True))

➞ using to_stop =True when done , from **the STOP codon then coming part is thrown away** .

Output :

MAIVMGR

This is where **the protein is completely stopped. to the ground until which part shows .**

Alternative Genetic Code (Table=2)

print(coding_dna.translate (table=2))

➞ **Genetic code table=2 is selected (Vertebrate Mitochondrial Code**

Output :

MAIVMGRWKGAR*

Here **UGA encodes Tryptophan (W) instead of STOP .**

print(coding_dna.translate (table=2, to_stop =True))

➞ Here **mitochondrial with code to STOP converts** .

Output :

MAIVMGRWKGAR

the STOP codon next part is thrown away .

```
print(coding_dna.translate())
#Seq("MAIVMGR*KGAR*")
print(coding_dna.translate(to_stop=True))
print(coding_dna.translate(table=2))
print(coding_dna.translate(table=2, to_stop=True))
```

```
MAIVMGR*KGAR*
MAIVMGR
MAIVMGRWKGAR*
MAIVMGRWKGAR
```

🔍 **STOP Codon (*) Why Is it visible ?**

DNA **triplet with codons** is read and some codons **protein synthesis that it's over (STOP codon).**

The main STOP codons are:

•TAA, TAG, TGA (Standard in the code)

•TGA → W (Tryptophan) (Mitochondrial in the code)

```
coding_dna.translate(table=2, stop_symbol="@")
```

```
Seq('MAIVMGRWKGAR@')
```

📌 **If using to_stop =True if not , STOP codons like * at the exit it seems .**

📌 If to_stop =True added if we do , from **the STOP codon next part is thrown away .**

Full encoder sequence (CDS – Complete Coding Sequence) and standard missing start codons

What is CDS (Coding Sequence) ?

CDS is a complete protein encoder genetic is a sequence :

✓ **Its length is divisible by 3** (each one one amen acid encoder threesome from codons consists of).

✓ **With start codon (ATG) begins** (protein synthesis start point).

✓ **With stop codon (TAA, TAG, TGA) ends** (protein synthesis stop point).

✓ **STOP codon inside does not exist** (if if so , protein unfinished remains).

This kind of sequence if any , translate() method with comfortable in the way to protein turns into .

Standard Missing Start Codon What is it ?

In bacteria some genes instead of ATG (Methionine) other with codons to begin can .

For example , **in E. coli K12 , the " yaaX " gene has a GTG (Valine) instead of ATG . begins .**

Standard Missing Start Codon Problem with

If so one translate() method in sequence use if k , **the previous codons other amen acids with start over .**

Standard ATG gives Methionine (M) , but some bacteria GTG and or TTG codons as start use does .

Bacteria for specific genetic from the code use when you do :

print(coding_dna.translate (table=11)) # Bacteria and Archaea for genetic code

At this time some non-standard start codons such as ATG will be read .

Conclusion

📌 A **complete one encoder sequence (CDS)** if any , **translate() method correct will work .**

📌 **In bacteria and some in organisms standard missing start codons use is being done .**

📌 **Bacteria genes correct to read for table=11 like suitable genetic codes should be selected .**

[70]:
```
from Bio.Seq import Seq
gene=Seq("GTGAAAAAGATGCAATCTATCGTACTCGCACTTTCCCTGGTTCTGGTCGCTCCCATGGCA"
    "GCACAGGCTGCGGAAATTACGTTAGTCCCGTCAGTAAAATTACAGATAGGCGATCGTGAT"
    "AATCGTGGCTATTACTGGGATGGAGGTCACTGGCGCGACCACGGCTGGTGGAAACAACAT"
    "TATGAATGGCGAGGCAATCGCTGGCACCTACACGGACCGCCGCCACCGCCGCGCCACCAT"
    "AAGAAAGCTCCTCATGATCATCACGGCGGTCATGGTCCAGGCAAACATCACCGCTAA")
print(gene)
```

```
GTGAAAAAGATGCAATCTATCGTACTCGCACTTTCCCTGGTTCTGGTCGCTCCCATGGCAGCACAGGCTGCGGAAATTACGTTAGTCCCGTCAGTAAAATTACAGATAGGCGATCGTGATAATCGTGGCTATTACTGGGATGGAGGTCACTG
GCGCGACCACGGCTGGTGGAAACAACATTATGAATGGCGAGGCAATCGCTGGCACCTACACGGACCGCCGCCACCGCCGCGCCACCATAAGAAAGCTCCTCATGATCATCACGGCGGTCATGGTCCAGGCAAACATCACCGCTAA
```

[72]:
```
print(gene.translate(table="Bacterial"))
```

```
VKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDHGWWKQHYEWRGNRWHLHGPPPPPRHHKKAPHDHHGGHGPGKHHR*
```

[74]:
```
gene.translate(table="Bacterial", to_stop=True)
```

[74]:
```
Seq('VKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDH...HHR')
```

[76]:
```
gene.translate(table="Bacterial", cds=True)
```

[76]:
```
Seq('MKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDH...HHR')
```

[84]:
```
print(gene)
print(gene.translate(table="Bacterial"))
print(gene.translate(table="Bacterial", to_stop=True))
print(gene.translate(table="Bacterial", cds=True))
```

```
GTGAAAAAGATGCAATCTATCGTACTCGCACTTTCCCTGGTTCTGGTCGCTCCCATGGCAGCACAGGCTGCGGAAATTACGTTAGTCCCGTCAGTAAAATTACAGATAGGCGATCGTGATAATCGTGGCTATTACTGGGATGGAGGTCACTG
GCGCGACCACGGCTGGTGGAAACAACATTATGAATGGCGAGGCAATCGCTGGCACCTACACGGACCGCCGCCACCGCCGCGCCACCATAAGAAAGCTCCTCATGATCATCACGGCGGTCATGGTCCAGGCAAACATCACCGCTAA
VKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDHGWWKQHYEWRGNRWHLHGPPPPPRHHKKAPHDHHGGHGPGKHHR*
VKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDHGWWKQHYEWRGNRWHLHGPPPPPRHHKKAPHDHHGGHGPGKHHR
MKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDHGWWKQHYEWRGNRWHLHGPPPPPRHHKKAPHDHHGGHGPGKHHR
```

[100]:
```
print (gene.translate(table="Bacterial"))
#Seq('VKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDH...HR*',ProteinAlpabet())
```

```
VKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDHGWWKQHYEWRGNRWHLHGPPPPPRHHKKAPHDHHGGHGPGKHHR*
```

[104]:
```
gene.translate(table="Bacterial", to_stop=True)
Seq('VKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDH...HHR')
#Bakterial genetik kodda GTG etibarlı başlanğıc kodonudur və
#normal olaraq Valini kodlasa da, başlanğıc kodonu kimi istifadə edilərsə,
#metionin kimi tərcümə edilməlidir.
```

[104]:
```
Seq('VKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDH...HHR')
```

[106]:
```
gene.translate(table="Bacterial", cds=True)
Seq('MKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDH...HHR')
```

[106]:
```
Seq('MKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDH...HHR')
```

[112]:
```
#Translation tables
#Standart translation cədvəli və  Vertebrate Mitoxondrial DNT üçün translation cədvəli.
from Bio.Data import CodonTable
standard_table = CodonTable.unambiguous_dna_by_name["Standard"]
mito_table = CodonTable.unambiguous_dna_by_name["Vertebrate Mitochondrial"]
print(standard_table)
print(mito_table)
```

```
Table 1 Standard, SGC0

  |  T      |  C      |  A      |  G      |
--+---------+---------+---------+---------+--
T | TTT F   | TCT S   | TAT Y   | TGT C   | T
T | TTC F   | TCC S   | TAC Y   | TGC C   | C
T | TTA L   | TCA S   | TAA Stop| TGA Stop| A
T | TTG L(s)| TCG S   | TAG Stop| TGG W   | G
--+---------+---------+---------+---------+--
C | CTT L   | CCT P   | CAT H   | CGT R   | T
C | CTC L   | CCC P   | CAC H   | CGC R   | C
C | CTA L   | CCA P   | CAA Q   | CGA R   | A
C | CTG L(s)| CCG P   | CAG Q   | CGG R   | G
--+---------+---------+---------+---------+--
A | ATT I   | ACT T   | AAT N   | AGT S   | T
A | ATC I   | ACC T   | AAC N   | AGC S   | C
A | ATA I   | ACA T   | AAA K   | AGA R   | A
A | ATG M(s)| ACG T   | AAG K   | AGG R   | G
--+---------+---------+---------+---------+--
G | GTT V   | GCT A   | GAT D   | GGT G   | T
G | GTC V   | GCC A   | GAC D   | GGC G   | C
G | GTA V   | GCA A   | GAA E   | GGA G   | A
G | GTG V   | GCG A   | GAG E   | GGG G   | G
--+---------+---------+---------+---------+--

Table 2 Vertebrate Mitochondrial, SGC1

  |  T      |  C      |  A      |  G      |
--+---------+---------+---------+---------+--
T | TTT F   | TCT S   | TAT Y   | TGT C   | T
T | TTC F   | TCC S   | TAC Y   | TGC C   | C
T | TTA L   | TCA S   | TAA Stop| TGA W   | A
T | TTG L   | TCG S   | TAG Stop| TGG W   | G
--+---------+---------+---------+---------+--
C | CTT L   | CCT P   | CAT H   | CGT R   | T
C | CTC L   | CCC P   | CAC H   | CGC R   | C
C | CTA L   | CCA P   | CAA Q   | CGA R   | A
C | CTG L   | CCG P   | CAG Q   | CGG R   | G
--+---------+---------+---------+---------+--
A | ATT I(s)| ACT T   | AAT N   | AGT S   | T
A | ATC I(s)| ACC T   | AAC N   | AGC S   | C
A | ATA M(s)| ACA T   | AAA K   | AGA Stop| A
A | ATG M(s)| ACG T   | AAG K   | AGG Stop| G
--+---------+---------+---------+---------+--
G | GTT V   | GCT A   | GAT D   | GGT G   | T
G | GTC V   | GCC A   | GAC D   | GGC G   | C
G | GTA V   | GCA A   | GAA E   | GGA G   | A
G | GTG V(s)| GCG A   | GAG E   | GGG G   | G
--+---------+---------+---------+---------+--
```

This table **genetic code is the schedule** and **mRNA codons which one amen to acids that it has become shows** .

Genetic code in the table every one amen acid **one literally representat being** :

Symbol	Amino acid	Full name
D	Aspartic acid	Aspartate (Asp)
E	Glutamic acid	Glutamate (Glu)
K	Lysine	Lysine (Lys)
Q	Glutamine	Glutamine (Gln)
H	Histidine	Histidine (His)
R	Arginine	Arginine (Arg)
W	Tryptophan	Tryptophan (Trp)
L	Leysin	Leucine (Leu)
F	Phenylalanine	Phenylalanine (Phe)
S	Cool	Serine (Ser)
A	Alanine	Alanine (Ala)
M	Methionine	Methionine (Met)
V	Valine	Valine (Val)

❖ **unambiguous_dna_by_name what does it ?**

Biopython in the library **CodonTable.unambiguous_dna_by_name** method **uncertain codons non- (i.e., not completely accurate) genetic code tables to call for use is being done .**

This means that that **only standard codons use is being and "ambiguous" (i.e. indefinite (bases) There is no** . For example :

•**A, T, G, C** The bases are unambiguous **.**

•**Ambiguous bases** while **N, Y, R , etc.** like certain not done nucleotides (for example , **N = A/T/G/C) can**).

```
from Bio.Data import CodonTable

standard_table = CodonTable.unambiguous_dna_by_name ["Standard"]
mito_table = CodonTable.unambiguous_dna_by_name ["Vertebrate Mitochondrial"]
```

✓ This code two different genetic code schedule calls :

Standard genetic code) – all in living things , mainly nuclear DNA for use is being done .
Vertebrate Mitochondrial code) – in mitochondria use is being and some different codon have meanings .

- **L(s) → Start codon be knowing leucine**
- **M(s) → Start codon be knowing methionine**

•Normally **AUG codon For Methionine (M) use is being and beginning codon (start codon) exit does .**
•However some in bacteria **L(s) and or M(s)** beginning codon be can .

•**TAA, TAG, TGA → are Stop codons (translation stops).**
•**ATG → Methionine (M) and is also a Start codon.**

One **codon** , **3 nucleotides consisting of one is a group** and one amen to acid suitable comes . For example :

•**AUG → Methionine (M)**
•**UUU → Phenylalanine (F)**
•**UGA → Stop (translate) stops)**

" Ambiguous" genetics codes What is it ?
Uncertain (ambiguous) codons which genetic codes are also used to be done can . This for ambiguous_dna_by_name There is a method .

For example :

python
Copy
Edit
from Bio.Data import CodonTable

ambiguous_table = CodonTable.ambiguous_dna_by_name ["Standard"]
print(ambiguous_table)
This version uncertain bases are also accepted (for example , N, Y, R) codes).

✓ "Unambiguous" means that only from bases A, T, G, C use is done .
✓ unambiguous_dna_by_name exactly and fully determined done genetic code tables to call for use is being done .

```
[ ]: #stop ve start codonlari...
mito_table.stop_codons
['TAA', 'TAG', 'AGA', 'AGG']
mito_table.start_codons
['ATT', 'ATC', 'ATA', 'ATG', 'GTG']
mito_table.forward_table["ACG"]
'T'
```

```
[114]: print(mito_table.stop_codons)
```

```
['TAA', 'TAG', 'AGA', 'AGG']
```

```
[116]: print(mito_table.start_codons)
```

```
['ATT', 'ATC', 'ATA', 'ATG', 'GTG']
```

```
[118]: print(mito_table.forward_table)
```

```
{'TTT': 'F', 'TTC': 'F', 'TTA': 'L', 'TTG': 'L', 'TCT': 'S', 'TCC': 'S', 'TCA': 'S', 'TCG': 'S', 'TAT': 'Y', 'TAC': 'Y', 'TGT': 'C', 'TGC': 'C', 'TGA':
'W', 'TGG': 'W', 'CTT': 'L', 'CTC': 'L', 'CTA': 'L', 'CTG': 'L', 'CCT': 'P', 'CCC': 'P', 'CCA': 'P', 'CCG': 'P', 'CAT': 'H', 'CAC': 'H', 'CAA': 'Q', 'CA
G': 'Q', 'CGT': 'R', 'CGC': 'R', 'CGA': 'R', 'CGG': 'R', 'ATT': 'I', 'ATC': 'I', 'ATA': 'M', 'ATG': 'M', 'ACT': 'T', 'ACC': 'T', 'ACA': 'T', 'ACG': 'T',
'AAT': 'N', 'AAC': 'N', 'AAA': 'K', 'AAG': 'K', 'AGT': 'S', 'AGC': 'S', 'GTT': 'V', 'GTC': 'V', 'GTA': 'V', 'GTG': 'V', 'GCT': 'A', 'GCC': 'A', 'GCA':
'A', 'GCG': 'A', 'GAT': 'D', 'GAC': 'D', 'GAA': 'E', 'GAG': 'E', 'GGT': 'G', 'GGC': 'G', 'GGA': 'G', 'GGG': 'G'}
```

```
[ ]:
```

Seq objects comparison making

[128]:
```
#Seq obyektlərinin müqayisə edilməsi
from Bio.Seq import Seq
seq1 = Seq("ACGT")
"ACGT" == seq1
print("ACGT"==seq1)
print(seq1=="ACGT")
```

```
True
True
```

Sequence unknown which sequences

Some bioinformatics in files (e.g. , **GenBank and EMBL**) sometimes **complete genomic DNA sequence is not given** , on the contrary , **only configuration (structure) information exists** . This means that that the **sequence length known , but specific as which one consisting of letters (nucleotides) where not shown** .

What is this ? for use is it ?

- **All to show the genome in full necessary not** – just length to know enough does .
- **Information big when to memory savings to do for use to be done can** .
- **Sequence yet known not and either investigation is in process . So , Seq (None, length=1000) is a sequence created it happens , but inside is empty and only 1000 nucleotides consisting of where It is known .**

```
from Bio.Seq import Seq
seq_unknown = Seq(None, length=1000)  # 1000 uzunluğunda, lakin içi boş olan bir Seq obyekti
print("seq_unknown")
print(len("seq_unknown"))  # 1000 çap edəcək
```

```
seq_unknown
11
```

MAF (Multiple Alignment Format)

- **MAF (Multiple Alignment Format) is a format used** in bioinformatics **to compare multiple genome sequences** .
- **the alignment of DNA sequences** of different organisms .
- It is used to find genetic evolution, mutations, and orthologous genes.

Structure Explanation

Every in line different one genomics of the organism There is information .

Here **human (hg38)** , **chimpanzee (panTro4)** , **macaque (rheMac3)** , **mouse (mm10)** , **rat (rn5)** , **dog (canFam3)** and **opossum (monDom5)** genomes given .

Every of the line format is like this :

```
#MAF (Multiple Alignment Format)
s hg38.chr7     117512683 36 + 159345973 TTGAAAACCTGAATGTGAGAGTCAGTCAAGGATAGT
s panTro4.chr7  119000876 36 + 161824586 TTGAAAACCTGAATGTGAGAGTCACTCAAGGATAGT
s rheMac3.chr3  156330991 36 + 198365852 CTGAAATCCTGAATGTGAGAGTCAATCAAGGATGGT
s mm10.chr6      18207101 36 + 149736546 CTGAAAACCTAAGTAGGAGAATCAACTAAGGATAAT
s rn5.chr4       42326848 36 + 248343840 CTGAAAACCTAAGTAGGAGAGACAGTTAAAGATAAT
s canFam3.chr14  56325207 36 +  60966679 TTGAAAAACTGATTATTAGAGTCAATTAAGGATAGT
s monDom5.chr8  173163865 36 + 312544902 TTAAGAAACTGGAAATGAGGGTTGAATGACAAACTT
```

Every of the line format is like this :

s < genome name > < chromosome > < start position > < length > < strand > < total length > <DNA sequence >

For example let's see :

s hg38.chr7 117512683 36 + 159345973 TTGAAAACCTGAATGTGAGAGTCAGTCAAGGATAGT

•s → sequence information (this sell sequence that shows)

•hg38.chr7 → Human genome , chromosome **7**

•117512683 → This part **chromosome on it beginning position**

•36 → **Sequence length** (36 nucleotides)

•+ → **Strand direction** (front strand "+" and either reverse strand "-")

•159345973 → This chromosome general length

•TTGAAAACCTGAATGTGAGAGTCAGTCAAGGATAGT → **DNA sequence**

This **is different in types of the same DNA region comparison** shows .

•**Human** and **chimpanzee** genomes very is similar .

•**Mouse and rat** in their genomes some There are differences .

•**Opossum** while more very differs .

All this information what for is it necessary ?

•**Genetic evolution to understand** (which species more is it close ?)

•**Evolution in the process which one mutations head gave ?**

•**Diseases genetic the reasons to investigate**

•**Which genes protected and functional that to understand**

• **This kind of many multiple alignment studies genetic in analyses , especially also evolution , diseases genetic reasons and gene functions to understand for use is being done .**

MutableSeq

MutableSeq , collection order to change , elements additional did to delete opportunity gives .

MutableSeq objects , usually in Python programming data structures with while working use done and elements to be changed opportunity giving function This type of functions , data on made of changes directly give birth in itself change to do opportunity recognizes .

MutableSeq in Python , in general collections.abc in the module compilation done one abstract This class is a collection class in Python . and lists representation to do for use done one main is a class . MutableSeq , a list representation does and elements on directly change to do opportunity gives (additional create , delete , update etc.).

[160]:
```
#MutableSeq objects
from Bio.Seq import Seq
my_seq = Seq("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA")
from Bio.Seq import MutableSeq
mutable_seq = MutableSeq(my_seq)
MutableSeq('GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA')
```

[160]:
```
MutableSeq('GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA')
```

[162]:
```
mutable_seq[5]="C"
```

[164]:
```
print(mutable_seq)
```

```
GCCATCGTAATGGGCCGCTGAAAGGGTGCCCGA
```

[166]:
```
#remove funkisyasi T nukleotidini cixaracaq.
mutable_seq.remove("T")
print(mutable_seq)
```

```
GCCACGTAATGGGCCGCTGAAAGGGTGCCCGA
```

[170]:
```
#reverse funksiyasi
mutable_seq.reverse()
print(mutable_seq)
```

```
AGCCCGTGGGAAAGTCGCCGGGTAATGCACCG
```

[180]:
```
#Stringlerle islemek...
from Bio.Seq import reverse_complement, transcribe, back_transcribe, translate
my_string = "GCTGTTATGGGTCGTTGGAAGGGTGGTCGTGCTGCTGGTTAG"
print(reverse_complement(my_string))
print(transcribe(my_string))
print(back_transcribe(my_string))
print(translate(my_string))
```

```
CTAACCAGCAGCACGACCACCCTTCCAACGACCCATAACAGC
GCUGUUAUGGGUCGUUGGAAGGGUGGUCGUGCUGCUGGUUAG
GCTGTTATGGGTCGTTGGAAGGGTGGTCGTGCTGCTGGTTAG
AVMGRWKGGRAAG*
```

[182]:
```
#Alt ardıcıllıqların tapılması
from Bio.Seq import Seq, MutableSeq
seq = Seq("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA")
print(seq.index("ATGGGCCGC"))
print(seq.index(b"ATGGGCCGC"))
print(seq.index(bytearray(b"ATGGGCCGC")))
print(seq.index(Seq("ATGGGCCGC")))
print(seq.index(MutableSeq("ATGGGCCGC")))
```

```
9
9
9
9
9
```

1.Seq ("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA") : This is Biopython library Seq from the class use by doing one genetic The sequence (GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA) represents does .
2.seq.index ("ATGGGCCGC") : This is Seq the sequence "ATGGGCCGC" in the object beginning finds the index of . The output will be 9 , because the sequence "ATGGGCCGC" seg from position 9 in the facility begins .
3.seq.index (b"ATGGGCCGC ") : Here the sequence "ATGGGCCGC" in bytes format given (i.e. b"ATGGGCCGC "). Biopython also accepts bytes format does and this is index 9 returns because this sequence also seq from position 9 in the facility begins .
4.seq.index (bytearray (b"ATGGGCCGC ")) : This is the index of the sequence "ATGGGCCGC" byte array in format bytearray , bytes can be changed knowing is version and this is index 9 returns .
5.seq.index (Seq ("ATGGGCCGC")) : Here sequence "ATGGGCCGC". Seq to the object turning seg at the facility This is again index 9 . gives , because Seq objects also adaptable knows .
6.seq.index (MutableSeq ("ATGGGCCGC")) : Here sequence "ATGGGCCGC". MutableSeq to the object turning seg at the facility This is index 9 . returns because MutableSeq also Seq to the object similar in the way works and adaptable knows .

Result :
These operations shows that , Biopython , various given compare formats (string, bytes, bytearray , Seq , MutableSeq) and their indexes to find for very is flexible . index() method given sequence representation who of the object of the type dependent regardless of the sequence located index the value returns . In this case , all this operations of the sequence "ATGGGCCGC" seg in the facility index as 9 returns .

[198]:
```
for index, sub in seq.search(["CC", "GGG", "CC"]):
    print(index, sub)
seq.find("ACTG")
seq.find("CC")
seq.rfind("CC")
#GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA
```

```
1 CC
11 GGG
14 CC
23 GGG
28 CC
29 CC
```

[198]:
```
29
```

This code fragment , **Biopython** library's search() method how that it works This method shows that the given certain sequences (sub- sequences) one genetic in sequence (Seq in the object) which in positions located to find for use is being done .

Code meaning Let's explain :

1.seq.search (["CC", "GGG", "CC"]) :

- The search() method returns the string within the sequence (seq). every a sub- sequence (this in the case of ["CC", "GGG", "CC"] list sequences) are searched .
- This method , given sequences inside where positions and every finds a sub- sequence .

2.for index, sub in seq.search (["CC", "GGG", "CC"]): :

- Here for loop , search() method turned the result every a sub- sequence and his/her found index taking works .
- index, sub- sequence started index position shows .
- sub is the found sub- sequence .

3.print(index, sub) :

- Every of the found sub- sequence index and sub- sequence himself to the screen print does .

Result :

This operation seg ["CC", "GGG", "CC"] in the sequence looking for and found every position and sub- sequence print does . In the output :

- CC sub- sequence at indices 1, 14 and 28 found .
- GGG sub- sequence at indices 11 and 23 found .