Lab sheet1: Information retrieval and sequence analysis

Q1

COX-2 (prostaglandin H2 synthase-2 (PTGS2)) gene

i. Access the entries for Human PTGS1 and PTGS2 in the "Gene" database at the NCBI (https://www.ncbi.nlm.nih.gov/) Website.

a. PTGS1 and PTGS2 are isozymes. Isozymes catalyze the same reaction but are separate genes. What types of reactions do PTGS enzymes catalyze? Also, what pathway are these enzymes a part of?
PTGS1 and PTGS2 are isozymes that catalyze the conversion of arachidonic acid into prostaglandins. They are part of the prostaglandin synthesis pathway.

b. How is the expression of PTGS1 and PTGS2 different?
PTGS1 is constitutively expressed in many tissues, while PTGS2 is inducible.
There is a biased expression in bone marrow (RPKM 59.3), urinary bladder (RPKM 41.1) and 11 other tissues

c. Which isozyme (PTGS1 or PTGS2) is required to inhibit inflammation?
PTGS2 is thought to be the more important isozyme. - PTGS2 is inducible and is therefore produced in higher levels at sites of inflammation

d. The drug Celebrex selectively inhibits PTGS2 while aspirin and other NSAID's inhibit both PTGS1 and PTGS2 in the same way. Why do you think researchers wanted to discover a selective inhibitor to PTGS2?
- To reduce the side effects of NSAIDs.
- To develop a more effective treatment for inflammatory diseases.
- To gain a better understanding of the role of PTGS2 in inflammation.

e. Describe how studying 3-D structures of PTGS1 and PTGS2 could help researchers design a drug that binds to PTGS1, but not to PTGS2.
By identifying the active site of each enzyme. The active site is the part of the enzyme that binds to the substrate. By comparing the active sites of PTGS1 and PTGS2, researchers could identify differences that could be exploited to design a drug that binds specifically to one enzyme or the other.

ii. Considering the Homo sapiens PTGS2 gene entry in NCBI gene https://www.ncbi.nlm.nih.gov/gene/ database,

a. What is the gene name?
PTGS2

b. What is the GeneID number?
5743

c. Where in the human genome is this gene located?
PTGS2 gene is located on chromosome 1, at position 1q31.1

d. What is the RefSeq accession number for the mRNA sequence of H o m o sa pie n s prostaglandin-endoperoxide synthase 2? - NM_000963.3

e. Download the prostaglandin-endoperoxide synthase 2 Reference mRNA sequence in "FASTA" format.

gene_id: 5743
gene_symbol: PTGS2
description: prostaglandin-endoperoxide synthase 2
scientific_name: Homo sapiens
common_name: human
tax_id: 9606
genomic_range: NC_000001.11:186671791-186680423;NC_060925.1:186026616 186035248
orientation: -;-
location: chr 1
gene_type: PROTEIN_CODING
transcript_accession: NM_000963.4
transcript_name:
transcript_length: 4510
transcript_cds_coords: NM_000963.4:134-1948
protein_accession: NP_000954.1
isoform_name:
protein_length: 604
protein_name: prostaglandin G/H synthase 2 precursor

f. What is the RefSeq accession number for the H o m o sa pie n s PTGS2 protein sequence? Download the sequence in "FASTA" format.
NP_000954.1

iii. Search for the UniProt entry for PTGS2 in Expasy https://www.expasy.org/website.

a. What are the alternate names for this protein.
Prostaglandin-endoperoxide synthase 2, Cyclooxygenase-2

b. What types of drugs target this protein?
Nonsteroidal anti-inflammatory drugs

c. What amino acid is acetylated by aspirin (amino acid type)?
Serine acid

iv. Translate the mRNA sequence of PTGS2 into Protein. Use "Translate " tool in ExPASy. Explain the output.

Fasta format

```
> VIRT-54282:5'3' Frame 2, start_pos=44
MLARALLLCAVLALSHTANPCCSHPCQNRGVCMSVGFDQYKCDCTRTGFY
GENCSTPEFLTRIKLFLKPTPNTVHYILTHFKGFWNVVNNIPFLRNAIMS
YVLTSRSHLIDSPPTYNADYGYKSWEAFSNLSYYTRALPPVPDDCPTPLG
VKGKKQLPDSNEIVEKLLLRRKFIPDPQGSNMMFAFFAQHFTHQFFKTDH
KRGPAFTNGLGHGVDLNHIYGETLARQRKLRLFKDGKMKYQIIDGEMYPP
TVKDTQAEMIYPPQVPEHLRFAVGQEVFGLVPGLMMYATIWLREHNRVCD
VLKQEHPEWGDEQLFQTSRLILIGETIKIVIEDYVQHLSGYHFKLKFDPE
LLFNKQFQYQNRIAAEFNTLYHWHPLLPDTFQIHDQKYNYQQFIYNNSIL
LEHGITQFVESFTRQIAGRVAGGRNVPPAVQKVSQASIDQSRQMKYQSFN
EYRKRFMLKPYESFEELTGEKEMSAELEALYGDIDAVELYPALLVEKPRP
DAIFGETMVEVGAPFSLKGLMGNVICSPAYWKPSTFGGEVGFQIINTASI
QSLICNNVKGCPFTSFSVPDPELIKTVTINASSSRSGLDDINPTVLLKER
STEL
```

First extracted the DNA seq of the PTGS2 from the file and translated it from the translator using the Verbose: Met, Stop, spaces between residues output format. Then chosen the most suitable frame and got the protein sequence and compared it with the protein sequence downloaded from the website.

Q2. Python Exercises

1. Write a Python code to extract the sample name from these files ignoring any files which do not match the format given below.

The format is:

1. Written lane number
2. Barcode
3. Sample name
4. Numeric lane number (starting with L)
5. Read number (R1/2/3/4)
6. File extension

Eg. `Lane8127_GCCAAT_S30_1_2l_Hap4_log_L001_R1.fastq.gz` the sample name would be,

`S30_1_2l_Hap4_log`

```
import re

def extract_sample_name(file_name):
    pattern = r'^lane\d+_([A-Z\d]+)_([A-Za-
z\d_]+)_L\d+_([R]\d+)\.fastq\.gz$'
```

```
    match = re.match(pattern, file_name)

    if match:
        return match.group(2)
    else:
        return "no"

files = [
    "lane1_NewCode_L001_R1.fastq.gz",
    "lane1_NoIndex_L001_R1.fastq.gz",
    "lane1_NoIndex_L001_R2.fastq.gz",
    "pipeline_processing_output.log",
    "lane7027_ACTGAT_JH25_L001_R1.fastq.gz",
    "lane7027_ACTTGA_E30_1_2_Hap4_24h_L001_R1.fastq.gz",
    "lane7027_AGTTCC_JH14_L001_R1.fastq.gz",
    "lane7027_CGGAAT_JH37_L001_R1.fastq.gz",
    "lane7027_GCCAAT_E30_1_2l_Hap4_log_L001_R1.fastq.gz"
,
    "lane7127_GGCTAC_E30_1_4_Hap4_48h_L001_R1.fastq.gz",
    "lane8127_GCCAAT_S30_1_2l_Hap4_log_L001_R1.fastq.gz"
]

sample_names = [extract_sample_name(file_name) for
file_name in files]
for file_name, sample_name in zip(files, sample_names):
    print(f"{file_name} -> {sample_name}")
```

```
PS C:\Users\Melaka> python -u "d:\Acadamic\4 th year\2nd sem\CB\Lab 1\Q2.py"
lane1_NewCode_L001_R1.fastq.gz -> no
lane1_NoIndex_L001_R1.fastq.gz -> no
lane1_NoIndex_L001_R2.fastq.gz -> no
pipeline_processing_output.log -> no
lane7027_ACTGAT_JH25_L001_R1.fastq.gz -> JH25
lane7027_ACTTGA_E30_1_2_Hap4_24h_L001_R1.fastq.gz -> E30_1_2_Hap4_24h
lane7027_AGTTCC_JH14_L001_R1.fastq.gz -> JH14
lane7027_CGGAAT_JH37_L001_R1.fastq.gz -> JH37
lane7027_GCCAAT_E30_1_2l_Hap4_log_L001_R1.fastq.gz -> E30_1_2l_Hap4_log
lane7127_GGCTAC_E30_1_4_Hap4_48h_L001_R1.fastq.gz -> E30_1_4_Hap4_48h
lane8127_GCCAAT_S30_1_2l_Hap4_log_L001_R1.fastq.gz -> S30_1_2l_Hap4_log
PS C:\Users\Melaka>
```

2.Create a FASTA file by obtaining 10 Dengue 1- Envelop gene DNA sequences from NCBI. Write a Python-program that reads the FASTA file, cleans up the header line to have only Accession number & gene-name and print headers and sequences to standard output as multi-FASTA-file again.

Here the DNA sequence that I wrote in the code is not the original gene sequence of Dengue 1- Envelop gene DNA sequences. It is an example.

```
import re

from Bio import Entrez, SeqIO

def fetch_dengue_sequences():

    Entrez.email = "pasindupathiranagama@gmail.com"

    handle = Entrez.esearch(db="nucleotide", term="Dengue 1 Envelope", retmax=10)

    record = Entrez.read(handle)

    ids = record["IdList"]


    dengue_sequences = []

    for record_id in ids:

        handle = Entrez.efetch(db="nucleotide", id=record_id, rettype="gb",
retmode="text")

        seq_record = SeqIO.read(handle, "genbank")

        dengue_sequences.append(seq_record)


    return dengue_sequences
```

```python
def write_fasta_file(sequences, output_file="dengue_sequences.fasta"):
    with open(output_file, "w") as output_handle:
        SeqIO.write(sequences, output_handle, "fasta")

def clean_up_header(header):
    pattern = r'(\S+).*?\((\w+)\) gene'
    match = re.search(pattern, header)

    if match:
        accession_number = match.group(1)
        gene_name = match.group(2)
        cleaned_header = f">{accession_number}_{gene_name}"
        return cleaned_header
    else:
        return ">Unknown"


def print_multi_fasta(file_path):
    sequences = list(SeqIO.parse(file_path, "fasta"))

    for seq_record in sequences:
        cleaned_header = clean_up_header(seq_record.description)
        print(cleaned_header)
        print(seq_record.seq)


if __name__ == "__main__":
    dengue_sequences = fetch_dengue_sequences()
    write_fasta_file(dengue_sequences)
    print_multi_fasta("dengue_sequences.fasta")
```

3. Write a Python program to search the DNA Sequence for the presence of one of the following Transcription Factor Binding Sites(TFBS) with ambiguity codes. Search for all the positions in the sequence where TFBS is located.

Code	Represents
A	Adenine
G	Guanine
C	Cytosine
T	Thymine
Y	Pyrimidine (C or T)
R	Purine (A or G)
W	weak (A or T)
S	strong (G or C)
K	keto (T or G)
M	amino (C or A)
D	A, G, T (not C)
V	A, C, G (not T)
H	A, C, T (not G)
B	C, G, T (not A)

Transcription Factor	Consensus Sequence
RUNX1	BHTGTGGTYW
TGIF1	WGACAGB
IKZF1	BTGGGARD

The sequence is shown below.

```
>search_seq
GACACCTCAGTACTAGGATGNNNNNNTATCAGCCTGAACTAGCAGGCCTGGTTCCAAATT
TTTTTATCAACACTCGTAGGGGGATTATCCTAGAGGGGGTCTGGGATTTCTTTGACATCA
GAGTATTTTTGCCTTGCTCCTTCACAATTTGGGAACAAATAATTTAGTGGTTATTAACCC
TGGCTACGCACTGGAAACTTTAAAAATAATGCTGGTATGAAATTTACACAGAGTATCGTG
AAAATTTTCACTGAGTACCATGTGGTTATACATTGGATAAGGCTCCAGGAAGCAGCTACT
GGAAGACAGCCATGCCAAGAGTGGTTAGTGGTTGGAATTTTGGCAAGTCAGTTTTAGTCT
GCCTTATCAAATACATGGGCATACAGATAAATCCTTAGATGGCTCTCCTACTTACTGAAA
CATTTTCTATCTATCTATCTATCTATCTATCTATTTGGGAAGCTATCTATCTATCTATCA
TTTATTTAAGGTAGTCTCTATCTGCCTCTGTCTCTGTCTGTCTCTGTGTCTCTGTGTCTG
TCTGCTCTCTCTCTCTCTCTGTGGGAATCTCTCTCTGTGTGTGTGTGTGTATGTGTGTGT
GTGTGTGTGTGGTGTGCATGAACATGAGTAAAATCCATAAGGAAACTTTCAGAGTTGGTC
CTCTCCTTATATCAAATGGATCCAGGAATTAAACTCAGGTTCAATTCTTGGTGCCTTTAC
TAGTTGAGCCATCTCACTGGCTCTTCATCATCTTTAGAATAAACTCACTTTATTACACAC
ACACACACACACACAACCTGGGAGTACACACACACACACAACCAAAGCCCCAACGGAAAA
CTACAATATTATAATGAATACACAGGTTCTCAACATAGTCTCTGCCACGCTTGCAGACAA
AGATGAGTAGAAGTAGAAAGAACCAGGGAAACGTGGAGCAAGTCAGAAGGAATAACAGTC
AGAAGGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAGTAACAGTCAGAAGGAATAGC
AGTCAGAAGGAATAACAGTCAGAAGACAGCACAGTCAGAAGGAATAACAGTCAGAAGGAA
TAACAGTCAGAAGGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAATAGCAGTCAGAA
GGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAATAACAGTCAAAGAAATAGCAGTCA
```

GAAGGAATAGCAGTCAGAAGGAATAACAGTCAAAGGAGCAGTCAGAAGGAGTAACAGTCA

GAAGGAATAACAGTCAGAAGGAATAACAGTCAAAGGAATAGCAGTCAGAAGGAGTAACAG

TCAGAGCAAACACAGAGATGACAAAGGCAATGGGGTCAGAGACTTCACCACTCTCCAAGA

```
import re

def search_tfbs(sequence, tfbs_dict):
    # Remove newlines and spaces from the sequence
    sequence = "".join(sequence.split())

    positions = {}
    for tf, consensus_sequence in tfbs_dict.items():
        # Convert ambiguity codes to regular expressions
        consensus_sequence =
consensus_sequence.replace('B', '[CGT]')
        consensus_sequence =
consensus_sequence.replace('D', '[AGT]')
        consensus_sequence =
consensus_sequence.replace('H', '[ACT]')
        consensus_sequence =
consensus_sequence.replace('K', '[GT]')
        consensus_sequence =
consensus_sequence.replace('M', '[AC]')
        consensus_sequence =
consensus_sequence.replace('N', '[ACGT]')
        consensus_sequence =
consensus_sequence.replace('R', '[AG]')
        consensus_sequence =
consensus_sequence.replace('S', '[CG]')
        consensus_sequence =
consensus_sequence.replace('V', '[ACG]')
```

```
        consensus_sequence =
consensus_sequence.replace('W', '[AT]')
        consensus_sequence =
consensus_sequence.replace('Y', '[CT]')

        matches = [match.start() for match in
re.finditer(f'(?={consensus_sequence})', sequence)]
        if matches:
            positions[tf] = matches
    return positions

if __name__ == "__main__":
    search_seq = """
    GACACCTCAGTACTAGGATGNNNNNNTATCAGCCTGAACTAGCAGGCCTGGT
TCCAAATT
    TTTTTATCAACACTCGTAGGGGGATTATCCTAGAGGGGGTCTGGGATTTCTT
TGACATCA
    GAGTATTTTTGCCTTGCTCCTTCACAATTTGGGAACAAATAATTTAGTGGTT
ATTAACCC
    TGGCTACGCACTGGAAACTTTAAAAATAATGCTGGTATGAAATTTACACAGA
GTATCGTG
    AAAATTTTCACTGAGTACCATGTGGTTATACATTGGATAAGGCTCCAGGAAG
CAGCTACT
    GGAAGACAGCCATGCCAAGAGTGGTTAGTGGTTGGAATTTTGGCAAGTCAGT
TTTAGTCT
    GCCTTATCAAATACATGGGCATACAGATAAATCCTTAGATGGCTCTCCTACT
TACTGAAA
    CATTTTCTATCTATCTATCTATCTATCTATCTATTTGGGAAGCTATCTATCT
ATCTATCA
    TTTATTTAAGGTAGTCTCTATCTGCCTCTGTCTCTGTCTGTCTCTGTGTCTC
TGTGTCTG
```

```
    TCTGCTCTCTCTCTCTCTCTGTGGGAATCTCTCTCTGTGTGTGTGTGTGTAT
GTGTGTGT
    GTGTGTGTGTGGTGTGCATGAACATGAGTAAAATCCATAAGGAAACTTTCAG
AGTTGGTC
    CCTCTCCTTATATCAAATGGATCCAGGAATTAAACTCAGGTTCAATTCTTGG
TGCCTTTAC
    TAGTTGAGCCATCTCACTGGCTCTTCATCATCTTTAGAATAAACTCACTTTA
TTACACAC
    ACACACACACACACAACCTGGGAGTACACACACACACACAACCAAAGCCCCA
ACGGAAAA
    CTACAATATTATAATGAATACACAGGTTCTCAACATAGTCTCTGCCACGCTT
GCAGACAA
    AGATGAGTAGAAGTAGAAAGAACCAGGGAAACGTGGAGCAAGTCAGAAGGAA
TAACAGTC
    AGAAGGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAGTAACAGTCAGAA
GGAATAGC
    AGTCAGAAGGAATAACAGTCAGAAGACAGCACAGTCAGAAGGAATAACAGTC
AGAAGGAA
    TAACAGTCAGAAGGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAATAGC
AGTCAGAA
    GGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAATAACAGTCAAAGAAAT
AGCAGTCA
    GAAGGAATAGCAGTCAGAAGGAATAACAGTCAAAGGAGCAGTCAGAAGGAGT
AACAGTCA
    GAAGGAATAACAGTCAGAAGGAATAACAGTCAAAGGAATAGCAGTCAGAAGG
AGTAACAG
    TCAGAGCAAACACAGAGATGACAAAGGCAATGGGGTCAGAGACTTCACCACT
CTCCAAGA
    """

    tfbs_dict = {
        "RUNX1": "BHTGTGGTYW",
```

```
        "TGIF1": "WGACAGB",
        "IKZF1": "BTGGGARD"
    }

    positions = search_tfbs(search_seq, tfbs_dict)

    for tf, tf_positions in positions.items():
        print(f"{tf} found at positions:
{tf_positions}")
```

Output:-

```
PS C:\Users\Melaka> python -u "d:\Acadamic\4 th year\2nd sem\CB\Lab 1\q3.py"
RUNX1 found at positions: [258]
TGIF1 found at positions: [303, 1044]
IKZF1 found at positions: [454, 560, 798]
PS C:\Users\Melaka>
```

Q3 – Biopython

Biopython Tutorial and Cookbook https://biopython.org/DIST/docs/tutorial/Tutorial.html#sec2

1. Write a Biopython program that asks the user to input a DNA-sequence and then translates the sequence to protein sequence.

```
2.
3. codon_table = {
4.     "TTT": "F", "TTC": "F", "TTA": "L", "TTG": "L",
5.     "CTT": "L", "CTC": "L", "CTA": "L", "CTG": "L",
6.     "ATT": "I", "ATC": "I", "ATA": "I", "ATG": "M",
7.     "GTT": "V", "GTC": "V", "GTA": "V", "GTG": "V",
8.     "TAT": "Y", "TAC": "Y", "TAA": "*", "TAG": "*",
9.     "CAT": "H", "CAC": "H", "CAA": "Q", "CAG": "Q",
10.          "AAT": "N", "AAC": "N", "AAA": "K", "AAG": "K",
11.          "GAT": "D", "GAC": "D", "GAA": "E", "GAG": "E",
12.          "TCT": "S", "TCC": "S", "TCA": "S", "TCG": "S",
13.          "CCT": "P", "CCC": "P", "CCA": "P", "CCG": "P",
14.          "ACT": "T", "ACC": "T", "ACA": "T", "ACG": "T",
15.          "GCT": "A", "GCC": "A", "GCA": "A", "GCG": "A",
16.          "TGT": "C", "TGC": "C", "TGA": "*", "TGG": "W",
17.          "CGT": "R", "CGC": "R", "CGA": "R", "CGG": "R",
18.          "AGT": "S", "AGC": "S", "AGA": "R", "AGG": "R",
19.          "GGT": "G", "GGC": "G", "GGA": "G", "GGG": "G"
20.      }
21.
22.      dna_sequence = input("Enter DNA sequence: ")
23.
24.      if not all(c in "ACTG" for c in dna_sequence):
25.          print("Invalid DNA sequence. Please enter a valid sequence.")
26.          exit()
27.
28.      if len(dna_sequence) % 3 != 0:
```

```
29.          print("DNA sequence length must be a multiple of 3. Please
   enter a valid sequence.")
30.          exit()
31.
32.      protein_sequence = ""
33.      for i in range(0, len(dna_sequence), 3):
34.          codon = dna_sequence[i:i+3]
35.          protein_sequence += codon_table[codon]
36.
37.      print("Protein sequence:", protein_sequence)
38.
```

```
PS D:\Acadamic\4 th year\2nd sem\CB\Lab 1> python -u "d:\Acadamic\4 th year\2nd sem\CB\Lab 1\3.1.py"
Enter DNA sequence: ATGGCCTACGCACTGGAAACT
Protein sequence: MAYALET
PS D:\Acadamic\4 th year\2nd sem\CB\Lab 1>
```

39. Write a Biopython program that will find all articles related to Alzheimer's in PubMed. Print the total number of articles available and the authors.

```
from Bio import Entrez

def search_pubmed(query, max_results=10):
    Entrez.email = "pasindupathiranagama@gmail.com"
    handle = Entrez.esearch(db="pubmed", term=query,
retmax=max_results)
    record = Entrez.read(handle)
    handle.close()
    return record

def fetch_pubmed_details(id_list):
    ids = ",".join(id_list)
```

```
    handle = Entrez.efetch(db="pubmed", id=ids,
rettype="medline",
    retmode="text")
    records = handle.read()
    handle.close()
    return records

def parse_pubmed_records(records):
    authors_list = []
    for record in records.split("\n\nPMID")[1:]:
        authors = []
    for line in record.split('\n'):
        if line.startswith('AU  - '):
            authors.append(line[6:])
            authors_list.append(authors)
    return authors_list

if __name__ == "__main__":
    query = "Alzheimer's"

# Search PubMed
search_results = search_pubmed(query)
# Print the total articles
total_articles = int(search_results["Count"])
print(f"Total number of articles related to Alzheimer's:
{total_articles}")

id_list = search_results["IdList"][:10]
pubmed_records = fetch_pubmed_details(id_list)
# Parse and print authors
authors_list = parse_pubmed_records(pubmed_records)
# Print authors
```

```python
for i, authors in enumerate(authors_list, 1):
    print(f"\nAuthors for Article {i}:")
    for author in authors:
        print(f" - {author}")
```

```
PS D:\Acadamic\4 th year\2nd sem\CB\Lab 1> python -u "d:\Acadamic\4 th year\2nd sem\CB\Lab 1\3.2.py"
Total number of articles related to Alzheimer's: 223336

Authors for Article 1:
 - Mitsunaga S
 - Fujito N
 - Nakaoka H
 - Imazeki R
 - Nagata E
 - Inoue I

Authors for Article 2:
 - Sajedi S
 - Ebrahimi G
 - Roudi R
 - Mehta I
 - Heshmat A
 - Samimi H
 - Kazempour S
 - Zainulabadeen A
 - Docking TR
 - Arora SP
 - Cigarroa F
 - Seshadri S
 - Karsan A
 - Zare H

Authors for Article 3:
```

40. Write a Biopython-program that finds CpG-islands from a given DNA-sequence.

```python
from Bio.Seq import Seq
from Bio.SeqUtils import nt_search
dna_seq = Seq(input("Enter the DNA sequence: "))
dna_seq_str = str(dna_seq)

def find_cpg_islands(sequence) :
    cpg_positions = nt_search (sequence, "CG") [1:]
    islands = []
```

```
49.          current_island = []
50.          for pos in cpg_positions:
51.              if not current_island or pos == current_island[-1] + 1:
52.                  current_island.append (pos)
53.              else:
54.                  islands.append (current_island)
55.                  current_island = [pos]
56.          islands.append (current_island)
57.          return islands
58.      cpg_islands = find_cpg_islands(dna_seq_str)
59.      print("CpG Islands found in:", cpg_islands)
```

```
PS D:\Acadamic\4 th year\2nd sem\CB\Lab 1> python -u "d:\Acadamic\4 th year\2nd sem\CB\Lab 1\3.3.py"
Enter the DNA sequence: GTGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG
CpG Islands found in: [[18], [33]]
PS D:\Acadamic\4 th year\2nd sem\CB\Lab 1>
```