**Lab sheet1: Information retrieval and sequence analysis**

**Q1**

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

1. Access the entries for Human PTGS1 and PTGS2 in the “Gene” database at the NCBI (<https://www.ncbi.nlm.nih.gov/>) Website.
   1. 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.

* 1. 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

* 1. 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

* 1. 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.
  1. 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.

1. Considering the Homo sapiens PTGS2 gene entry in NCBI gene <https://www.ncbi.nlm.nih.gov/gene/> database,
   1. What is the gene name?

PTGS2

* 1. What is the GeneID number?

5743

* 1. Where in the human genome is this gene located?

PTGS2 gene is located on chromosome 1, at position 1q31.1

* 1. 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
  2. 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*

* 1. 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

1. Search for the UniProt entry for PTGS2 in Expasy <https://www.expasy.org/>website.
   1. What are the alternate names for this protein.

Prostaglandin-endoperoxide synthase 2, Cyclooxygenase-2

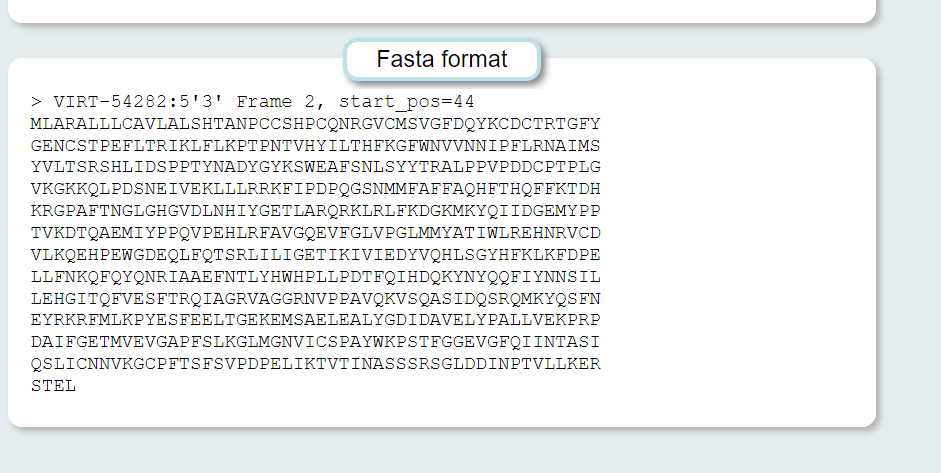
* 1. What types of drugs target this protein?

Nonsteroidal anti-inflammatory drugs

* 1. What amino acid is acetylated by aspirin (amino acid type)?

Serine acid

1. Translate the mRNA sequence of PTGS2 into Protein. Use “Translate “ tool in ExPASy. Explain the output.



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",

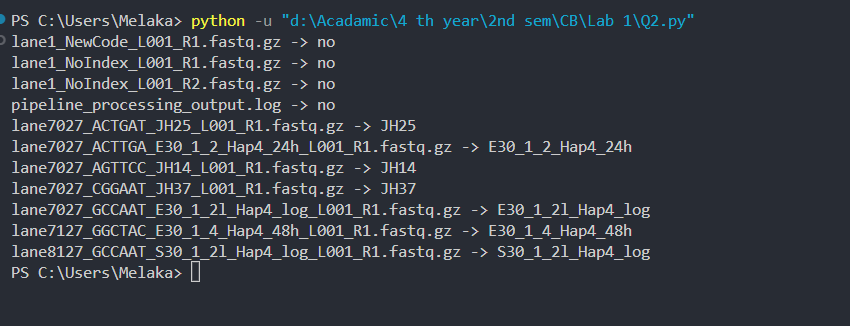
    "lane8127\_GCCAAT\_S30\_1\_2l\_Hap4\_log\_L001\_R1.fastq.gz"

]

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}")



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

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")

A chart with different colored text

Description automatically generated with medium confidence3. 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.

|  |  |
| --- | --- |
| **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 = """

    GACACCTCAGTACTAGGATGNNNNNNTATCAGCCTGAACTAGCAGGCCTGGTTCCAAATT

    TTTTTATCAACACTCGTAGGGGGATTATCCTAGAGGGGGTCTGGGATTTCTTTGACATCA

    GAGTATTTTTGCCTTGCTCCTTCACAATTTGGGAACAAATAATTTAGTGGTTATTAACCC

    TGGCTACGCACTGGAAACTTTAAAAATAATGCTGGTATGAAATTTACACAGAGTATCGTG

    AAAATTTTCACTGAGTACCATGTGGTTATACATTGGATAAGGCTCCAGGAAGCAGCTACT

    GGAAGACAGCCATGCCAAGAGTGGTTAGTGGTTGGAATTTTGGCAAGTCAGTTTTAGTCT

    GCCTTATCAAATACATGGGCATACAGATAAATCCTTAGATGGCTCTCCTACTTACTGAAA

    CATTTTCTATCTATCTATCTATCTATCTATCTATTTGGGAAGCTATCTATCTATCTATCA

    TTTATTTAAGGTAGTCTCTATCTGCCTCTGTCTCTGTCTGTCTCTGTGTCTCTGTGTCTG

    TCTGCTCTCTCTCTCTCTCTGTGGGAATCTCTCTCTGTGTGTGTGTGTGTATGTGTGTGT

    GTGTGTGTGTGGTGTGCATGAACATGAGTAAAATCCATAAGGAAACTTTCAGAGTTGGTC

    CCTCTCCTTATATCAAATGGATCCAGGAATTAAACTCAGGTTCAATTCTTGGTGCCTTTAC

    TAGTTGAGCCATCTCACTGGCTCTTCATCATCTTTAGAATAAACTCACTTTATTACACAC

    ACACACACACACACAACCTGGGAGTACACACACACACACAACCAAAGCCCCAACGGAAAA

    CTACAATATTATAATGAATACACAGGTTCTCAACATAGTCTCTGCCACGCTTGCAGACAA

    AGATGAGTAGAAGTAGAAAGAACCAGGGAAACGTGGAGCAAGTCAGAAGGAATAACAGTC

    AGAAGGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAGTAACAGTCAGAAGGAATAGC

    AGTCAGAAGGAATAACAGTCAGAAGACAGCACAGTCAGAAGGAATAACAGTCAGAAGGAA

    TAACAGTCAGAAGGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAATAGCAGTCAGAA

    GGAATAACAGTCAGAAGGAATAACAGTCAGAAGGAATAACAGTCAAAGAAATAGCAGTCA

    GAAGGAATAGCAGTCAGAAGGAATAACAGTCAAAGGAGCAGTCAGAAGGAGTAACAGTCA

    GAAGGAATAACAGTCAGAAGGAATAACAGTCAAAGGAATAGCAGTCAGAAGGAGTAACAG

    TCAGAGCAAACACAGAGATGACAAAGGCAATGGGGTCAGAGACTTCACCACTCTCCAAGA

    """

    tfbs\_dict = {

        "RUNX1": "BHTGTGGTYW",

        "TGIF1": "WGACAGB",

        "IKZF1": "BTGGGARD"

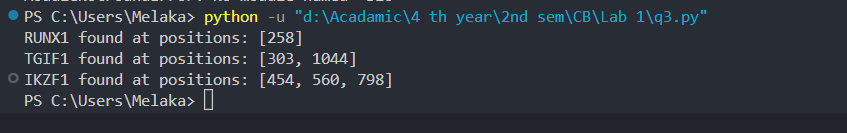
    }

    positions = search\_tfbs(search\_seq, tfbs\_dict)

    for tf, tf\_positions in positions.items():

        print(f"{tf} found at positions: {tf\_positions}")

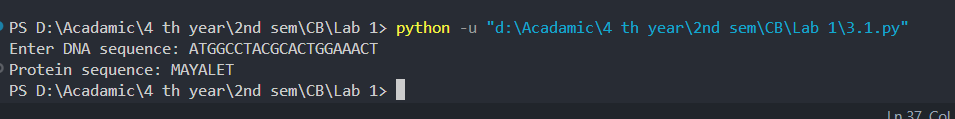
Output:-



**Q3 – Biopython**

**Biopython Tutorial and Cookbook** [**https://biopython.org/DIST/docs/tutorial/Tutorial.html#sec2**](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. codon\_table = {
3. "TTT": "F", "TTC": "F", "TTA": "L", "TTG": "L",
4. "CTT": "L", "CTC": "L", "CTA": "L", "CTG": "L",
5. "ATT": "I", "ATC": "I", "ATA": "I", "ATG": "M",
6. "GTT": "V", "GTC": "V", "GTA": "V", "GTG": "V",
7. "TAT": "Y", "TAC": "Y", "TAA": "\*", "TAG": "\*",
8. "CAT": "H", "CAC": "H", "CAA": "Q", "CAG": "Q",
9. "AAT": "N", "AAC": "N", "AAA": "K", "AAG": "K",
10. "GAT": "D", "GAC": "D", "GAA": "E", "GAG": "E",
11. "TCT": "S", "TCC": "S", "TCA": "S", "TCG": "S",
12. "CCT": "P", "CCC": "P", "CCA": "P", "CCG": "P",
13. "ACT": "T", "ACC": "T", "ACA": "T", "ACG": "T",
14. "GCT": "A", "GCC": "A", "GCA": "A", "GCG": "A",
15. "TGT": "C", "TGC": "C", "TGA": "\*", "TGG": "W",
16. "CGT": "R", "CGC": "R", "CGA": "R", "CGG": "R",
17. "AGT": "S", "AGC": "S", "AGA": "R", "AGG": "R",
18. "GGT": "G", "GGC": "G", "GGA": "G", "GGG": "G"
19. }
20. dna\_sequence = input("Enter DNA sequence: ")
21. if not all(c in "ACTG" for c in dna\_sequence):
22. print("Invalid DNA sequence. Please enter a valid sequence.")
23. exit()
24. if len(dna\_sequence) % 3 != 0:
25. print("DNA sequence length must be a multiple of 3. Please enter a valid sequence.")
26. exit()
27. protein\_sequence = ""
28. for i in range(0, len(dna\_sequence), 3):
29. codon = dna\_sequence[i:i+3]
30. protein\_sequence += codon\_table[codon]
31. print("Protein sequence:", protein\_sequence)



1. 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)

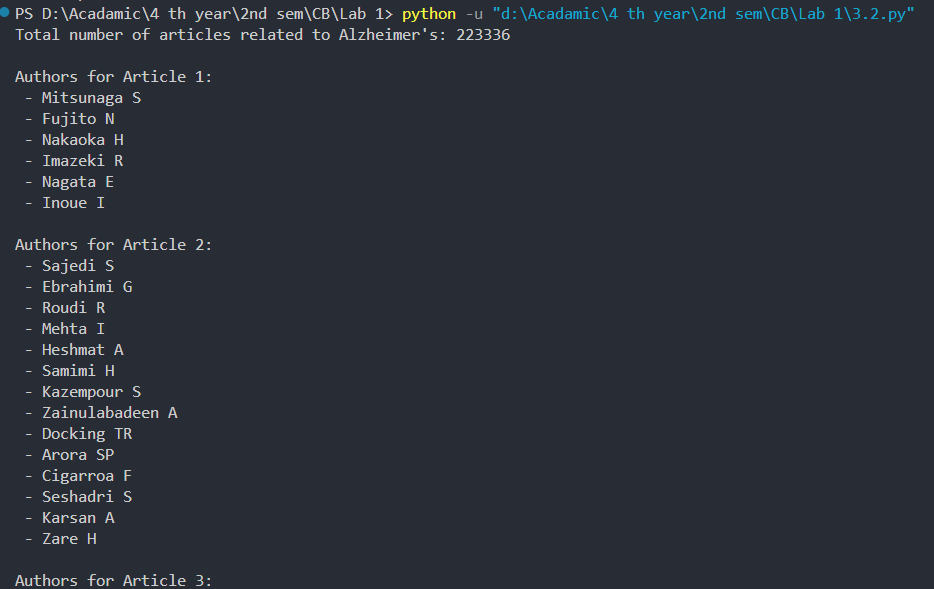
*# Print authors*

for i, authors in enumerate(authors\_list, 1):

    print(f"\nAuthors for Article {i}:")

    for author in authors:

        print(f" - {author}")



1. Write a Biopython-program that finds CpG-islands from a given DNA-sequence.
2. from Bio.Seq import Seq
3. from Bio.SeqUtils import nt\_search
4. dna\_seq = Seq(input("Enter the DNA sequence: "))
5. dna\_seq\_str = str(dna\_seq)
6. def find\_cpg\_islands(sequence) :
7. cpg\_positions = nt\_search (sequence, "CG") [1:]
8. islands = []
9. current\_island = []
10. for pos in cpg\_positions:
11. if not current\_island or pos == current\_island[-1] + 1:
12. current\_island.append (pos)
13. else:
14. islands.append (current\_island)
15. current\_island = [pos]
16. islands.append (current\_island)
17. return islands
18. cpg\_islands = find\_cpg\_islands(dna\_seq\_str)
19. print("CpG Islands found in:", cpg\_islands)

