Lab 02 Introduction to Python

The goal of Lab 02 | Introduction to Python is learn how to conduct basic operations and write simple functions in Python. You will write functions to count the number of nucleotides in a sequence. The lab is divided into the following sections:

1. Getting started with Jupyter Lab
2. Prelab review
3. Nucleotide counts

Assignment

Follow the instructions in this document and answer the questions in the cell below each question. Start by copying Lab02 from the public directory to your home or preferred working directory.

Gradoscope

Submit your answers on Gradescope (www.gradescope.com, also accessible through a gradescope link in blackboard). You should be able to login using your institutional NETID on gradescope (Login > School credentials > Rochester NETID). If you have problems ask myself or the TAs for help.

When uploading the assignment to Gradoscope you will be asked to select the page(s) that contain the answer to each question. Click the question on the left and click the page(s) on the right for each question. When you are done hit 'Submit' on the bottom right.

Instructions for PDF upload

Within Jupyter export your notebook in HTML by selecting >File, >Export to ..., >HTML. Note that PDF export often does not work as it depends on the system and your browser. You can use PDF export, just check to make sure the cells and your answers are filled out.

Open the exported HTML (downloads) in your browser and print to PDF. Check to make sure all your cells have been run and the **results** displayed.

Reminder, provide comments for any code you write to ensure partial credit.

Question 1

What is your name?

ENTER YOUR NAME HERE

Double click this cell to edit. Shift-enter to render it again.

Rename this file replacing NETID with your network username.

If interested you can read about Markdown using Jupyter notebooks here:
https://jupyter.brynmawr.edu/services/public/dblank/Jupyter%20Notebook%20Users%20Manual.ipynb
(https://jupyter.brynmawr.edu/services/public/dblank/Jupyter%20Notebook%20Users%20Manual.ipynb)

(1 point)

Getting starting with Jupyter Lab

You are viewing a Jupyter notebook. It can be viewed using either *Jupyter Lab* or *Jupyter Notebook*.

The notebook consists of cells of different types. Markdown cells show text and images. Code cells can be executed using Python3. The current cell is a Markdown cell. The next one is a code cell.

In [1]:
```
print( "This is code cell and when run the output is printed below.")
```

```
This is code cell and when run the output is printed below.
```

You can insert or delete cells using the + and scissors icons from the menu. Change the type of cell (Markdown, code) using the dropdown menu.

Cells are meant to be run in order from top to bottom. You can rerun all the cells starting from the first one using *Run all* from the Notebook (Jupyter Lab) or Cell (Jupyter Notebook) menu.

Running cells out of order can cause problems, especially if you use the same variable name in different cells. Take the following three cells, where x is assigned 5, printed and then assigned 6.

In [2]:
```
x = 5
```

In [3]:
```
print(x)
```

```
5
```

In [4]:
```
x = 6
```

Rerun the prior three cells in order by clicking inside the cell and then shit-enter. Then run the cell with "print(x)". It should now print 6 because the last cell to be run assigned x the value 6.

The number in brackets to the left of the code indicates the order of code that was executed. To restart the notebook and remove any prior history, restart the kernel under the Kernel menu (Jupyter Notebook) or the Notebook menu (Jupyter Lab), and the run all cells.

Review of prelab

Strings and for loops

String assignment and print

In [5]:
```
seq = 'CGAT'
print( seq )
```

```
CGAT
```

Get characters and substrings from a string. Note that strings are zero-based arrays, zero is the first position.

In [6]:
```
print(seq[2])
print(seq[0:2])
```

```
A
CG
```

Iterate characters in a string. Conveniently, a string can be accessed like a list of characters!

In [7]:
```
for i in range( len(seq) ):
    print(seq[i], i)
```

```
C 0
G 1
A 2
T 3
```

Count characters in a string.

In [8]:
```
count = 0
for i in range( len(seq) ):
    count += 1
    # This is the same as count = count + 1
print( 'There are', count, 'characters')
```

```
There are 4 characters
```

Count the number of 'A' characters in a string.

In [9]:
```
countA = 0
for i in range( len(seq) ):
    if ( seq[i] == 'A'):
        countA += 1
print( 'Number of \'A\': ', countA)
# Notice that a backslash is used to escape the single quotes so that
they are literally interpreted
# rather than being interpreted as the end of the printed string
```

```
Number of 'A':  1
```

Using a dictionary to count nucleotides

In [10]:
```
nucleotideCounts = { 'A': 0, 'G': 0, 'C': 0, 'T': 0 }
seq = 'CGATCGATAGCAGGGGA'
# For loop to count - see next cell for explanation
for n in seq:
    nucleotideCounts[n] += 1
# To print we can iterate over all characters in the sequence.
for base in ['A','G','C','T']:
    print(nucleotideCounts[base], end = " ")
```

```
5 7 3 2
```

Because seq can be interpreted as a list, and range() returns a list, we can just use seq rather than the range() method. This is illustrated in the printing loop where we iterate over each element in the list of characters.

What is the *end = " "* and how would one know to use that to print characters on the same line? Besides using a google search to find this out, Python has extensive online documentation, including each built-in function.

Python documentation is here: https://docs.python.org/3/ (https://docs.python.org/3/)
And the part on *print* is here: https://docs.python.org/3.7/library/functions.html#print (https://docs.python.org/3.7/library/functions.html#print)

Functions

The following code implements a character count of any input string.

In [11]:
```
def charactercount(string):
    count = 0
    for i in string:
        count += 1
    return count

short = charactercount('UofR')
long = charactercount('UniversityofRochester')
print( short )
print( long )
```

```
4
21
```

Nucleotide counts

Question 2

Write two functions in Python. Use the code cells below to enter your answer. A template for both is provided.

The first function should take as input a DNA sequence (string), count the number of of A, C, G and T bases using a list, and return that list.

The second function should take as input a DNA sequence (string), count the number of bases using a dictionary, and return the dictionary.

(6 points)

In [12]:
```
# A function that uses a list
def listfun (string):
    bases = [0, 0, 0, 0] # To store counts for A, C, G, T in that ord
er
    # Enter you code here

    return bases
test = ('GTCTTCAATAGACTCCTTGTGCAAGCGCTGATAGTCCTGCAACCCGTCCAAGTGTGGAGA
ATAGTGGGTAGCAATT'+
'GCGGGCTGCAGTCTATCTGAGATGGGCCGTTGTGGCACGATCTTGACCGAGGTCAAATGTTCATACTC
ATGTTCCTTCTTCTGCTG'+
'CGCAGTGGAGGCAGACTGGGACATTTTTGCTTTCAACTTGTCAATTTCACTTGACTGTTCTTCTAGTT
TTGATGATTG')
listfun( test )
```

Out[12]:
```
[0, 0, 0, 0]
```

In [13]:
```
# A function that uses a dictionary
def dictfun (string):
    bases = { 'A': 0, 'G': 0, 'C': 0, 'T': 0 }
    # Enter you code here

    return bases
test = ('GTCTTCAATAGACTCCTTGTGCAAGCGCTGATAGTCCTGCAACCCGTCCAAGTGTGGAGA
ATAGTGGGTAGCAATT'+
'GCGGGCTGCAGTCTATCTGAGATGGGCCGTTGTGGCACGATCTTGACCGAGGTCAAATGTTCATACTC
ATGTTCCTTCTTCTGCTG'+
'CGCAGTGGAGGCAGACTGGGACATTTTTGCTTTCAACTTGTCAATTTCACTTGACTGTTCTTCTAGTT
TTGATGATTG')
dictfun( test )
```

Out[13]:
```
{'A': 0, 'G': 0, 'C': 0, 'T': 0}
```

Question 3

Suppose we wanted to count each character (a-z) in a book. What type of data structure would be easier to write the code for (list or dictionary) and why?

(4 points)

Double click here to enter your answer:

File IO

In many cases one would want to read sequences from a file. FASTA formatted files are a common convention in Computational Biology and Bioinformatics and is used for storing named sequences of nucleotides.

The hallmark of FASTA format is that it contains two types of lines; some sequence identifier line that will always begin with the '>' symbol, and actual nucleotide sequence. The sequence identifier line needs to only take up one line, and so is separated from nucleotide sequence by the newline character (\n).

The nucleotide sequence itself consists of the 4 nucleotides, Adenine, Thymine, Cytosine and Guanine, indicated by A, T, C and G.

FASTA files often store very long sequences such as entire chromosomes. For this reason, it is syntactically and aesthetically practical for FASTA sequences to be interrupted by line breaks. A common default is to have sequence break to the next line every 80 nucleotides.

FASTA files can also store more than one sequence, with the '>' symbol indicating when a new sequence entry is starting and its name.

Lets read a FASTA file. To do so, make sure that the file lab02.fasta is visible in your Jupyter Lab/Notebook file browser, i.e. its in the same directory as your python notebook. If you haven't already, copy Lab02 into your directory.

In [14]:

```
filename = 'lab02.fasta'
fasta = open( filename, 'r' )
head = '' # empty header defined
seq = '' # empty sequence
for line in fasta.readlines(): # Using the readlines() method on the
 File object returns a list of lines
    line = line.strip()
    # To remove the newline '\n' and any leading or trailing white sp
ace we use the strip() function.
    # The strip function returns a string after removal.
    if ( line[0] == '>' ):
        header = line[1:]
    else:
        seq = seq + line

fasta.close() # Close the filehandle using the close() method on the
 File object.
print(header)
print(seq)
```

lab02 nucleotide sequence

GCATGCAATACAGTGACATATATATATACCCTAACACTACCCTAACCCTACCCTATTTCAACCCTTCCA
ACCTGTCTCTCAACTTACCCTCACATTACCCTACCTCTCCACTTGTTACCCTGTCCCATTCAACCATAC
CACTCCCAACCACCATCCATCCCTCTACTTACTACCACCAATCAACCGTCCACCATAACCGTTACCCTC
CAATTAGCCATATTCAACTTCACTACCACTTACCCTGCCATTACTCTACCATCCACCATCTGCTACTCA
CCATACTGTTGTTCTACCCTCCATATTGAAACGTTAACAAATGATCGTAAATAATACACATATACTTAC
CCTACCACTTTATACCACCACACATCACATGCCATACTCACCTTCACTTGTATACTGATATGCCATACG
CACACGGATGCTACAGTATATACCACTCTCAAACTTACCCTACTCTCACATTCTACTCCACTCCATGAC
CCATCTCTCACTAAATCAGTACTAAATGCACCCACATCATTATGCACGGCACTTGCCTCAGCGGTCTAT
ACCCTGAGCCATTTACCCATAACTCCCACGATTATCCACATTTTAATATCTATATCTCATTCGGCGGGC
CCAAATATTGTATAACTGCTCTTAATACATACGTTATACCACTTTTGCACCATATACTAACCACTCAAA
TTATATACACTTATGCCAATATAACCAAAAAATTACCACTAAAATCACCTAAACATAAAAATTATTTAT
CTTTCAACTTTACGAAATAAACACACTCAATTGCGTATCTATACCACCATGACGTCATTAACGTAAAAG
TTCCTTAATATTACCATTTGCTTGAACGGATACCATTTCAGAATATTTCTAACTTTCACAGACCATACA
TTAGAATAATATGCCACCTCACTGTCGTAACACTCTATATTCACCGAGAAACAATACGGTAGTGGCTCA
AACTCATGCCGGTGCTATGATACAATTGTATCCTATTTCCATTCTCATATGCTATCCGCAATATCCTAA
AAGCATAACTGATGCATCTTTAATCTTGTATGTGACACTACTCATACGAAGAGACTATATCTAAAGAAG
ACGATACAGTGATATGTACGTTGTTTTTGTAGAATTATAATGAAACGTCAAATAACCCTACTATATTAT
AACTTATCAGCGCCGTATACTAAAACGGACGTTACGATATTGTCTCACTTCATCTTACCACCCTCTATC
TTATTGTTGATAAAACACTAACCCCTCAGCTTTATTTCTAGTTACAGTTACAACAAACTATCCCAAACC
ATAAATCTTAATATTTTAGGTGTCAAAAAATGAGGATCTCCAAATGAGAGTTTGGTACCATGACTTGTA
ACTCCACTACCCTGATCTGCAATCTTGTTCTTAGAAGTGACGCATACTCTATATGGCCCGACGCGACGC
GCCAAAAAAATGAAAAAAGAAGCAGCGACTCATTTTTATGGAAGGACAAAGTGCTGCGAAGTCATACGCT
TCCAATTTCATTATTGTTTATTGGACATACTCTGTTAGCTTTATTACCGTCCACGCTTTTTCTACAATA
GTGTAAAAGTTTCTTTCTTATGTTCATCGTATTCATAAAATGCTTCACGAACACCGTCATTGATCAAAT
AGGTTTATAATATTAATATACATTTATATAATCGGCGGTATTTATATCATCAAAAAAAGTAGTTTTTTA
TTTTATTTTTTCATTACTTTTCACTGTCTATGGATTTTCATTCGTAAAGGCATCACTCCCTAGTTTGCG
ATAGTGTAGATACCGTCCTTGGATAGAGCACTGGAGATGGCTGGCTTTAATCTGCTGGAGTACCATGGA
ACACCGGTGATCATTCTGGTCACTTGGTCTGGGGCAATACCAGTCAACATGGTGGTGAAGTCACCGTAG
TTGAAAACGGCTTCAGCAACTTCGACTGGGTAGGTTTCAGTTGGGTGGGCGGCTTGGAACATGTAGTAT
TGGGCCAAGTGAGCTCTGATATCAGAGACGTAGACACCCAATTCCACCAAGTTGACTCTTTCGTCAGAT
TGAGCTAGAGTGGTGGTTGCAGAAAGCAGTAGCAGCGATGGCAGCGACACCAGCAGCGATTGAAGTTAA
TTTGACCATTGTATTTGTTTTGTTTGTTAGTGCTGATGTAATCTTAACAAGAAATAGTGAAATGAAAGC
GCATACCTCAAAGGCATATAGTTGAAGCAGCTCTATTTATACCCGTTCCTCCATCTGTCATCACTACTT
AAACGATTCGTTAACAGACGCTCATTTAGCACCTCACATATTCTCCATATCTCATCTTTCACACAATCT
CATTATCTCTATGGAGATGCTCTTGTTTCTGAACGAATCATACATCTTTCATAGGTTTCGTATGTGGAG
TATTGTTTTATGGCACTCATGTGTATTCGTATGCGCAGAATGTGGGAATGCCTATTATAGGGGTGCCGG
GGGGTGCCTTGAAAAACCCTTTTGCGCGCCTGTTAAGTTTCCGTTTTCAGTCAAAAAGAATATCCGAAT
TTTAGATTTGGACCCTCGTTCAGAAGCTTACTGTCTAAGCCATCATTTGGTGTGTCCTAAACGGTTTCC
ATGCAAAGCAACATCTTTGTTACTCATTCCTGAAGGTTGAATAAAAAATAGCAACCATGTGCAGGGAGT
CGTATATTGTTAGATTCTAGAGACTTGTACGCATCATCAAAGCTGTAAATAGAATAAACATACGCAAGG
CGTCAAAAGTGCATAGTTAAGAAAATTCCTGACATGTGAAAATATGTGTTTATGAAATGTGTCAAGGCC
CGTCTATAGCGTAGTTAACCCCTCTGCAGGAGTAAGTGACTTTTTTTACGCTCAAAAGGCAACGAGGGC
ACATACTTAAAAGTCATTTTCAAACACATCTGCAGTTTGCAACGACAGATAACAATATTATGATAGGAT
GGTATGATGTTATTAGCTGCCACATATTTTTTAACATAGCATTAGTCACGTCTCTTCAATTGTTGGGAT
GAAACTCTAAAATATCATTCCTTTAGTAGTATTCCAGTTACCAGTATATTATCACATGCCGAAAAAGAA
GATGACATAAAGATCGACAAACAGTCTTCAAATATAATGGAAGCTGGAATGCAAGGATTGATAATGTAA
CAGGATACTGAATGACAAAGTATAAATGAAAAAAAAAAAAAAAAAAAAGTAGTAATACTATTATGTGGA
AATACCGATTCCATTTTGAGGATTCCAATTGTTGGAATAAAAATCAACTATCATCTACTAACTAGTATT
TACGTTACTAGTATATTATCATATACGGTGTTAGAAGATGACGCAAATGATGAGAAATAGTCATCTAAA
TTAGTGGAAGCTGAAACGCAAGGATTGATAATGTAATAGGATCAATGAATATTAACATATAAAACGATG
ATAATAATATTTATAGAATTGTGTAGAATTGCAGATTCCCTTTTATGGATTCCTAAATCCTGGAGGAGA
ACTTCTAGTATATCTACATACCTAATATTATAGCCTTAATCACAATGGAATCCCAACAATTACATCAAA
ATCCACATTCTCTACACCAATACCATCGACGAGAGCTTCTAGTAAATTGTATACATAACAGTATAACCC
TTACCAACAATGGAATCTCAAAGATTATTAAATTATTCACAGACTCTGAGGATTCGGGTAAAATAGGGT
ATTTAACTGGTTACCGGAAAGGTTTAGAAAATTCGTGGAGGGTTGGCCGAGTGGTCTAAGGCGGCAGAC
TTAAGATCTGTTGGACGGTTGTCCGCGCGAGTTCGAACCTCGCATTCATTTTCCGGATCCACTTTATAG

```
TGGGTCACTTCATCTCTTGTAGGGTTCACTGAAAGATAATCAATCAAATTATCATATGACCAATGGGTC
ACTTTTTTGATGAGACCCATGTTTATTCTTCTATACGTTCGTGATACTGCACTTACGACTACCAGTAAC
ATCAGAATAACCATAGAGCTTATGCTATCTTGAAATATATCATCGCCGTAAAGTATGATGTGTAGCGCA
TTAGTCTTCTGCATTCATATTATCAGTGGTATACCGTTTGGTATATGCTTTCCAATCATGAAGTGTTGG
CTCTACTCTATATAAATTCGACATCCATTGATTTTCTCAATAAATGAGTTACCCAAGTAAGCTTTCCAT
TGATACAAGTGATCTACATTCTTGCGACGCCAAATTATAAAGCACTAAAAATCATATCATACCCAATTG
CGGGCAGCACATGTATATATTATACACTACTTATAACTAACCTTCTTCATATATAACAATATGCCTGAT
AATACTAATGCATTTAAATCATATCATGAAAGAAAATTACATGGGTTTTATTGACATAATTGCATTTAG
AATACATAAAATTCTAAAGAATTAATATATCCAAAGTATTAGACATAACCAAGAATAATAGTGAATAAT
TTTAGATTTTGTTACATATAATTCTGCTTGCCTATCTCTTCCACTCTTTTCAAAACGTTGCATGTAAGC
GTTACTAATATTCCGCTTTATTTTGTTGCAATTCCTAATTTTTTCATTACATTATCTTGCGAGTACGGA
AGCGATTAACGTTCTCCCAATAGAAGGAACAAACATAGATATTGAAGTTTTACTGCTTTTGCTTACCTG
ACCTTTTTCAAATTTAATTTTTTCCCGCTAATAAGACCATAAACTACCCCGAACCAAATTCTAAAAGAT
AGTCAGCTGGATTAGAGTTGTCATCTCCAAACATTAATTTTGCATTATCTTCGGCTTCAATCAAATCGC
CTGATAAGAACTCCTTTAATTCTTCATGAATGTTTGTATGTGGATGACTCTCCATAGTGCCAGCATGAT
TGTGGTTACCGACCGAATCATATAACGGTGGCTCCCAATTGTGCAATTCAGCCTTACTATTTTGTTCAT
TCATCAAAGCATTTGGGACAGATCTAATATCTATAATTCTTTCCTCACTATTCTCGCTATTATTTTGCC
CCGAACTGGCATGGTGGTTATTGGTAAAAGGAGATAATGCTGCGACAGAACTTTTCTTCTCTTCCAATT
CTTTTATCTTCGCCAATAACGCCTTCTTTTTTCGTGCTTGTATTTCTAAAATTTCGGCCAGGTATTGTA
AATATTCGACCGTTCTATCCAAAATGATACCCTTATTTGGTTTGATTTGTTTACCTAGGTCATCGTAAT
TCAATAAAGATGGTGGAACCAACTGGCCGAGTTCTTTTATCTTTTGCTTTATTAATTCTCTTCTTCTCC
TTTCGACGGCATTATGAAACTCTCTTTTGCGCCTCAGTTTCTCATCAGAAGTTAACCCGCCTAAAATCT
TTGGAACACTTCCAGGTCCTATATTTTCAGTCATATTGCTACTTATTGAAGTGTGTCTTGTTCTGGGTG
TGTTTATGCTACCATGCCTAAAAGAAGATGAAAGGAAACTTCCTGCACGAAATGACGATGATGGTGATC
TTACCTTTGGGGAATATGTGGAAGATACGGATGCTGGGCCCAAGCTTTGTGGATTATAAGAAAATGATG
ATGAATATGTGTTTGGTGTCATCATATCAGAATTGATGCTAGAAGATAAAGAGGAGCTCAAATCATCAG
TTAAGTTATACATTGTGTCATCGGTGCCATGCTGAAATAAAAAGTCATGTGGTAACTCAGCTTTTAAGC
CTTCTTGTTGCTGTAATGGAGTATTCATTGCTC
```

Question 4

Run the code in the above cell on you computer, making sure the file is read and sequence is output. If not, modify the filename location, e.g. filename = '/home/netid/labs/lab02.fasta' so that its the correct location.

What is the 587th - 599th base in the sequence. Use a command in the cell below to find your answer. And remember the first position is stored at 0 in a list.

(2 points)

In [15]:
```
# Using the seq string write a command that will output the result
```

Pseudocode

Pseudocode is a helpful tool for creating a program. Having the general idea and flow of the algorithm/program written out in plain language is a great way to get started and detect the semantics that the program may need to account for. It also helps avoid errors in logic before spending the time to write code that can't be used.

Here is an example of pseudocode for a program that will read a FASTA file, count nucleotides and output the total counts. Note that the total counts include all sequences in the FASTA file.

In [16]:
```
# Triple quotes are like multiline comments, whereas # is a single li
ne comment
# You can ignore any output.
"""
program nucleotideCounter
    set fastaFile to file given at comamnd line
    open fastaFile
    define countNucleotidesList function with sequence as input
        initialize list of nucleotide counts to zero
        for each nucleotide in sequence
            if nucleotide is A
                increase nucleotideCounts of A by 1
            if nucleotide is G
                increase nucleotideCounts of G by 1
            if nucleotide is C
                increase nucleotideCounts of C by 1
            if nucleotide is T
                increase nucleotideCounts of T by 1
        return nucleotideCounts
    initialize a sequence string
    for each line of fastaFile
        if line starts with '>'
            continue
        else
            add line to the end of sequence string
    get results of calling countNucleotidesList (sequence string)
    print results
    close fastaFile
"""
```

Out[16]:
```
'\nprogram nucleotideCounter\n    set fastaFile to file given at coma
mnd line\n    open fastaFile\n    define countNucleotidesList functio
n with sequence as input\n        initialize list of nucleotide count
s to zero\n        for each nucleotide in sequence\n            if nu
cleotide is A\n                increase nucleotideCounts of A by 1\n
if nucleotide is G\n                increase nucleotideCounts of G by
1 \n            if nucleotide is C\n                increase nucleoti
deCounts of C by 1 \n            if nucleotide is T\n
increase nucleotideCounts of T by 1 \n        return nucleotideCounts
\n    initialize a sequence string\n    for each line of fastaFile\n
if line starts with '>'\n            continue\n        else\n
add line to the end of sequence string\n    get results of calling co
untNucleotidesList (sequence string)\n    print results\n    close fa
staFile\n'
```

Question 5

Write pseudocode for a function that will return the base present given a sequence and position as input (1-based such that the sequence AGG has position A=1, G=2, G=3). If the position is not in the sequence, the program should not throw an error but should print "There is no base at that position".

(3 points)

In [17]:
```
"""
define findabase function with input sequence and position
    enter pseudo code here

"""
```

Out[17]:
```
'\ndefine findabase function with input sequence and position\n    en
ter pseudo code here\n    \n'
```

Question 6

Write a function that will return the base present given a sequence and position (1-based such that the sequence AGG has position A=1, G=2, G=3). If the position is not in the sequence, the program should not throw an error but should print "There is no base at that position". Use your function to query the sequence for positions 0, 1, 55, 105, 127, 205, 705

(4 points)

In [18]:
```
seq = ('TAACCCCTCAGCTTTATTTCTAGTTACAGTTACAACAAACTATCCCAAACCATAAATCTT'
+
'AATATTTTAGGTGTCAAAAAATGAGGATCTCCAAATGAGAGTTTGGTACCATGACTTGTA'+
'ACTCCACTACCCTGATCTGCAATCTTGTTCTTAGAAGTGACGCATACTCTATATGGCCCG'+
'ACGCGACGCGCCAAAAAATGAAAAAAGAAGCAGCGACTCATTTTTATGGAAGGACAAAGT'+
'GCTGCGAAGTCATACGCTTCCAATTTCATTATTGTTTATTGGACATACTCTGTTAGCTTT'+
'ATTACCGTCCACGCTTTTTCTACAATAGTGTAAAAGTTTCTTTCTTATGTTCATCGTATT'+
'CATAAAATGCTTCACGAACACCGTCATTGATCAAATAGGTTTATAATATTAATATACATT')
# i is the position you want, sequence is the input sequence.

def findpos( i, sequence ):
    # replace code below
    print("No answer")
findpos( 0, seq)
findpos( 1, seq)
findpos( 55, seq)
findpos( 105, seq)
findpos( 127, seq)
findpos( 205, seq)
findpos( 705, seq)
```

```
No answer
No answer
No answer
No answer
No answer
No answer
No answer
```

In [ ]: