INNSPUB

International Journal of Biosciences | IJB |
ISSN: 2220-6655 (Print), 2222-5234 (Online)
http://www.innspub.net
Vol. 11, No. 3, p. 116-125, 2017

RESEARCH PAPER OPEN ACCESS

Genomic data mining through python language

Rashid Saif*[1], Kinza Qazi[2], Talha Tamseel[2], Saeeda Zia[3]

[1]*Institute of Biotechnology, Gulab Devi Educational Complex, Lahore, Pakistan*

[2]*Department of Biotechnology, Virtual University of Pakistan, Lahore, Pakistan*

[3]*Department of Mathematics, National University of Computer and Emerging Sciences Lahore, Pakistan*

Key words: Python, Sequence analysis, Python syntax, Programing data structure

http://dx.doi.org/10.12692/ijb/11.3.116-125 Article published on September 27, 2017

Abstract

Pythonis a rigorous programming language, which may be used for many purposes including genomic data mining. This language was designed to emphasize on code readability and syntax, which allows programmer to express code in lesser space with comprehensive and exhaustive manner. Different analysis through Python can be conducted during dry labs sessions, which infer concrete and generalizable results from the wet lab genomic experiments, such as gene expression analysis, phylogenetic, GC percentage and gene sequencing. In this article, built-in Python functions like variables, stings, operators and formatting styles are introduced, and short programs are structured, implemented and executed. Basic operators are used to perform calculations through this language, gene sequences are analyzed and small built-in functions e.g. "length, print, integers and types" of Python are also conversed in this communication. Case sensitive commands are elaborated to avoid errors during the process of computing. This endeavor also shed light on the topic that how different Python methods and functions may be used to compute data structures, dictionaries, sets, lists, tuples, loops and statements on the genomic sequences. Finally, different programs are constructed to count undefined bases in a given sequence with the help of statement, condition functions based on Boolean expressions, loops function are also used to analyze undefined amino acids present in protein sequences with the help of "for" and "while" loops.

* **Corresponding Author:** Rashid Saif ✉ rashid.saif37@gmail.pk

Introduction

Python is a high level programming language which is designed to emphasize on code readability and syntax, which allows users to express code in less space and user friendly manner. As compare to C++ or C#, it consumes less space, execute in lesser time and have short syntax. Python language is use to create different programs for web and software development, but it is also applicable in the field of computational biology (List *et al.*, 2017). In bioinformatics, Python is used to create and code different programs, like phylogenetic tree, count of GC percentage in DNA sequence, transcription and translation of DNA sequence, gene expression data analysis, clustering techniques(Oliphant, 2007). It is an object oriented language, which can keep large programs well organized and occupy less space. Object oriented language is based on object, rather than procedures. Every model or class is considered as an object with some attributes, one most important advantage of scripting language is that, it does not rerun the entire program like C language which runs the program after compilation, as C is a procedural language everything run in a sequenced manner. In Python everything is object based, so when demanded object class is called compiler compile it and show on interpreter without measuring any sequences. Python interpreter and compiler is available in two versions, Python 2.x and 3.6, later one is the most appropriate and updated version with new set commands. Python 3.6 comes with two types of prompts, one is python IDEL and second one is python 3.6 GUI. Python have large data storage in build-in libraries. Data structures, loops, variables, strings, tuples and many other built-in function and methods are used to analyze data(Perkins, 2010).

Python methods

Python as a Calculator

Python have several commands that are similar to C and C++.First, open a python interpreter by typing Python in window's search bar. There are some basic immutable data types which is used to perform different minor operations in Python, such as int, float, str etc. (Mann, 2010).

These data types are used to work Python as a calculator. Open Python IDEL and type 5+5 after the prompt execution e.g. after pressing enter you will see that it executes the answer in next line as 10.

```
>>> 5+5
10
```

Let's try another calculation like 10.5-2*3 the answer is 4.5 (in Python 2 the answer is 4 because it doesn't able to deal with decimals).

```
>>> 10.5-2*3
4.5
```

To write an exponential function in Python, write it as *(asterisk). These assignment operators are used in Python to calculate different values.

```
>>> 10**2
```

Division in Pythons is done by using / operator by writing 12/5 in interpreter, the answer is 2 but in Python 3 the answer is 2.4. This happen in Python 3, because it has a build-in function of float, which is not in Python 2. In Python 2 to get accurate answers add “.0” after 12 and then division operator, so the answer will be 2.4

```
>>> 12/5
2 (in Python 2)
2.4 (in Python 3).
```

To compute remainder in division, use % operator for example 17%2, remainder is 1. In Python, the order of operations is same like in math multiplication, takes precedence over addition or subtraction. So, the answer to this question, 5*3+2 will be 17. Numbers have different types.To find out the type of a number, type (5) and the answer is type int, 3.5 is type float.

```
>>> (5)
Type 'int'
>>>3.5
Type 'float'.
```

Sequence Analysis Commands and Syntax

In biological sequence analysis, Python uses very important string data type. String is a series of letters which is written within quotes. Type anything within single or double quotes, which is declared as a string (Kinser, 2010). For small strings, single quotes will be used and for paragraphs double quotes, such as;

```
>>> ' actg'
>>> ' this is a stop codon, isn't it?'
Error
```

Errors occurred, because Python interpreter reads quotes and execute the phrase. In this case, it reads (' this is a stop codon, isn') this statement only and give error for afterwards statement. To avoid this,use double quotes and type long phrases

```
>>> "this is a stop codon, isn't it?"
```

For writing multiple statements within a same string another method is used by adding triple quotes before and after the strings.

```
>>> """
dna 1 acttttttttttttttttttttttttttttaccacttactac
dna 2 aattctftgatacactgctttc
dna 3 tgtgaacctttgactctctac
"""\ndna 1 acttttttttttttttttttttttttttttaccacttactac\ndna 2aattctftgatacactgctttc\ndna 3 tgtgaacctttgactctctac\n'
```

This method is appropriate, when more than one sequence will be added to the same string.

But this method after execution comes out with line breaks (end of the line)"\n" which makes executed statement scribbled, to elude such type of syntax, escape characters are used.

Escape Characters Feature

\n new line

\\ Backslash

\t new tab

\" double quotes

To appraise such type of inconvenience backslash after triple quotes at the starting of code is inserted. By this command no more line breaks comes out after execution. Now, the string looks really nice, no more line breaks (\n).

```
Print ("""\dna 1 acttttttttttttttttttttttttttttaccacttactac
dna 2 aattctftgatacactgctttc
dna 3 tgtgaacctttgactctctac""")
out put
dna 1 acttttttttttttttttttttttttttttaccacttactac
dna 2 aattctftgatacactgctttc
dna 3 tgtgaacctttgactctctac
```

Print is a built-in function of Python use to print string statement in Python.

```
>>> print ('acttctactaactgttcgtcatc')
Acttctactaactgttcgtcatc.
```

Basic String Operators

There are some basic string operators used in Python which performs different function.

(+) operator is used to concatenate two strings.

```
>>> 'atgactac' + 'actgcgc'
'atgactacactgcgc'
```

(in:) operator is used to check, member, union and intersection behavior present between two strings.

```
'atgactacactgcgc'
>>> 'actg' in 'acatgctgttac'
False
```

Variables in Python

Variables are storage containers for numbers and strings, without describing variable, string cannot be executed in program(Oliphant, 2007). Instead of manipulating strings and numbers, variables are important to store direct data in it and assign some name to it. To assign a variable, assignment operator (=)is used after variable name and then write its description.

```
>>> dna_sequence="acctcactgtgtgactc"
```

After entering, this variable is declared with a name of dna_sequence, to check the defined variable, type its name and it is implemented.

```
>>> dna_sequence
'acctcactgtgtgactc'
```

If variable is not defined, it will give an error like this

```
>>>dna
Traceback (most recent call last):
File "<pyshell#12>", line 1, in <module>
dna
Name Error: name 'dna' is not defined
```

Change the value of variables associated with names, suppose value is assigned to variable.

Considered, ifa=4 is assigned to a same value b=4, this cannot change the value of a, but assign a same value to b.

```
>>> a=4
>>> b=4
>>>b
```

```
4
>>>a
4
```

Increase the values of variable by doing this method.

```
>>> b=b+3
>>>b
7
```

Variables are case sensitive, means that variable are different and can only start with alphabets and after that any character is placed. If any other character except alphabet added as initial of name Python gives error.

Built-in Functions of Python

Python have a lot of built-in function such as, print(), return(), tuple(), sum(), Boolean(), open(), dictionary(),etc. all these functions are invaluable functions present in python tool kit, and used by passing several string and other datatype arguments with in the brackets.

"Print" is a built-in function of Python which is usedto print string statement in Python.

```
>>> print ('acttctactaactgttcgtcatc')
Acttctactaactgttcgtcatc
```

Input codes:

```
dna=input("enter a dna sequences:")
```

output:

```
enter a dnasequences: actgtcatctctctactacgcgtgtc
```

After enteringDNA sequence in output, variable named as "dna" assigned in Python library.

Integer is another built-in function use to covert string into an integer.

Input:

```
>>> actual_number=int(my_number)
```

Input:

```
>>>type (actual_number)
```

Output:

```
<class 'int'>
>>>type (my_number)
```

Output:

```
<class 'str'>
```

Length function:

```
>>>len (dna) 26.
```

Information Retrieval from Genomic Data

As discussed earlier that how Python can be used for different mathematical and computational purposes, Python is also useful in data sciences especially in bioinformatics, different programs of biological data will structured and analyze through Python(Lesk, 2013). Some imperative programs in Python is to count percentage of base contents, and find undefined bases etc.

GC percentage count using Python

DNA sequences consists four nitrogenous bases, each base pair of DNA have specific amount of percentage present in sequences. To find GC percentage in DNA sequences, through Python different data structures are designed to count GC percentage in a sequence.

Before proceeding towards percentage coding, there are few terminologies which are necessary to understand.

Get DNA Sequence from User

```
DNA='actgacgcatgcacgtcttgctgactctgcgac'
```

After getting sequence from user it is required to count presence of G's and C's in sequences. After GC percentage is counted.

Dna. count is a function use to count nucleotides amount in entered sequence.

Dna. find is also a method used to find individual nucleotide present on different position.

To count G's and C's following code is compose in Python interpreter.

Input

```
>>> no_c=DNA. Count ('c')
>>>no_c
```

Output

```
11
```

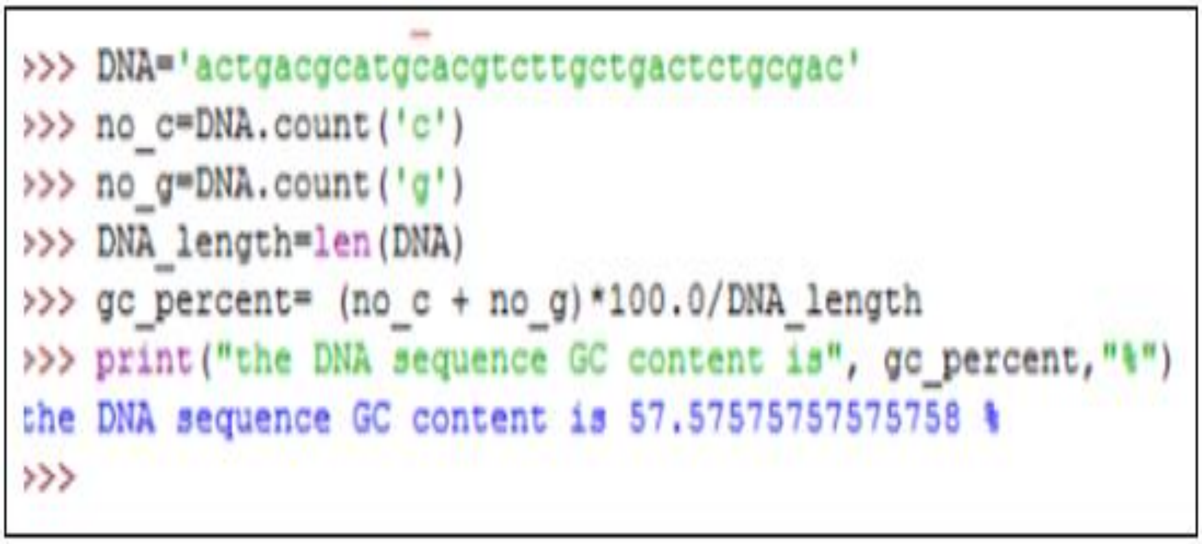
Input

```
>>>no_g=DNA.count('g')
>>>no_g
8
```

Here is a coding to compute GC %

```
>>>gc_percent= (no_c + no_g)*100.0/DNA_length
>>> gc_percent
57.57575757575758
```

It is not necessary to compose all this in Python interpreter, to make program more interactive write this entire code in simple text file and save it with extension (.Py), then open this file in Python shell and press F5 to execute this program. Figure 1 shows the print statement for GC percentage, write code in resource file and print statement is executed in interpreter.



```
>>> DNA='actgacgcatgcacgtcttgctgactctgcgac'
>>> no_c=DNA.count('c')
>>> no_g=DNA.count('g')
>>> DNA_length=len(DNA)
>>> gc_percent= (no_c + no_g)*100.0/DNA_length
>>> print("the DNA sequence GC content is", gc_percent,"%")
the DNA sequence GC content is 57.57575757575758 %
>>>
```

Fig. 1. Pseudocode for count GC percentage of a given sequence.

```
Print ("the DNA sequence GC content is", gc_percent,"%")
The DNA sequence GC content is 57.57575757575758 %
```

Above given percentage is ambiguous and large, to make this percentage count more clear and precise, formatting command is used which is written right after the print statement as %5.3f

Here % indicates the following format, 5 is the total number of digits, 3 indicates the digits followed by decimal and f is the formatting style.

Formatting Commands

%d is use to transformed into integers

```
>>> print ("%d" % 10.6)
10
```

%3d is used to give space

```
Print("%3d" % 10 )
 10
```

%e is used to convert power of scientific notation

```
>>> print ("%e" % 10.6 )
1.060000e+01.
```

Data Structures

List is one of the most important data structures. To create a list, type a sequence of values enclosed in square brackets. These values don't have to be the same type(Hamelryck and Manderick, 2003).To create a list it is necessary that value which is entered in list should be declared already and stored in computer memory, so when list is created computer automatically pick the value and assigned it in the list. Suppose here list of gene expression is created which contain gene string, and three float values which are the p values of gene expression.

gene_expression is a variable which hold this list.

```
gene_expression = ['gene', 5.19e-08, 0.0012817, 6.23e-06]
>>> gene_expression
['gene', 5.19e-08, 0.0012817, 6.23e-06]
```

Same like string, list variable is also starts with 0 index, from these indexing it’s easy to access individual values by typing a command and index position. Since index is starts from zero so the third element will be at index two.

To find the value in list type the index number and it tells what value is at this position.

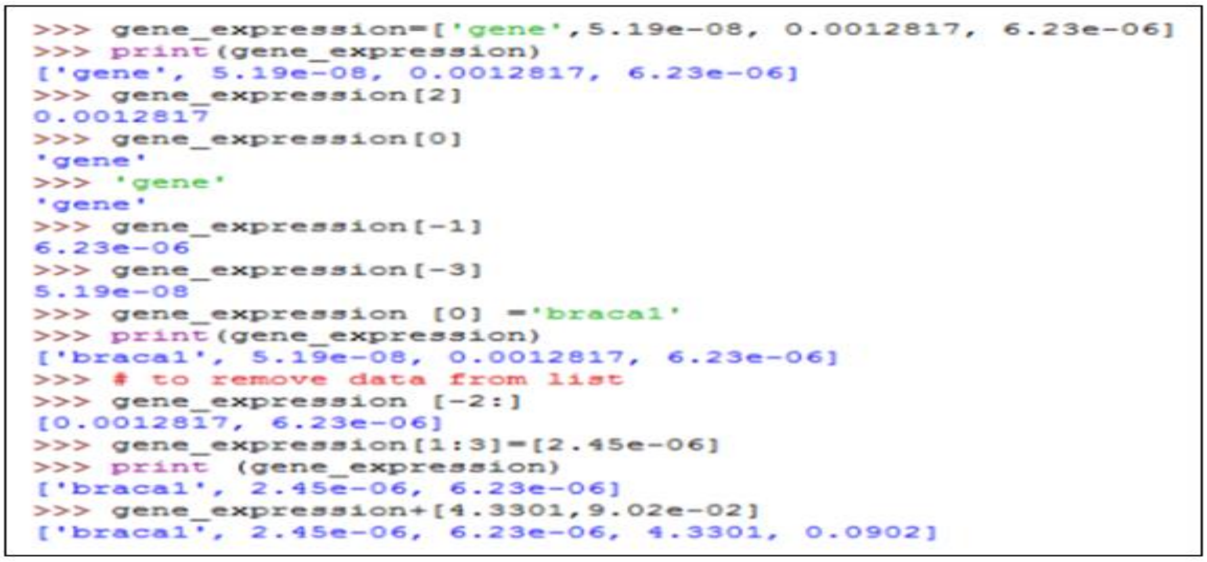
```
>>> gene_expression[2]
0.0012817
>>> gene_expression [0]
'gene'
```

List can also be modified to change the values if previous value is changed or another new gene is introduced categorized in same list rather than making new list, it is easy to modify same list. But Python is also case sensitive so it does not allow to add another value in list which have no index position(Kinser, 2010).Here index of 3 is used so it is not able to add any other value on index 4. By this it means, list can only be replaced old objects with new one and do not add new objects by enlarging its space.

```
>>> gene_expression [0] ='braca1'
>>> print (gene_expression)
['brca1', 5.19e-08, 0.0012817, 6.23e-06]
```

another important point on case sensitive issue is noticed, suppose braca1 is another string which is replaced earlier have some changed values like sequence of As, Gs, Cs and Ts are unknown before and written as Ns but know these bases are known and replaced with instring here Python gives an error. So we can't change string value because it is immutable data type while variable is mutable data type.

There is another command which is very important to slice a list, suppose there is some values which were no more in use so this slice command is used to eliminate that value and a new list is created. Figure 2 shows how to return on previous list, other than this new p value is also added to list and remove previous one which changes the list.



```
>>> gene_expression=['gene',5.19e-08, 0.0012817, 6.23e-06]
>>> print(gene_expression)
['gene', 5.19e-08, 0.0012817, 6.23e-06]
>>> gene_expression[2]
0.0012817
>>> gene_expression[0]
'gene'
>>> 'gene'
'gene'
>>> gene_expression[-1]
6.23e-06
>>> gene_expression[-3]
5.19e-08
>>> gene_expression [0] ='braca1'
>>> print(gene_expression)
['braca1', 5.19e-08, 0.0012817, 6.23e-06]
>>> # to remove data from list
>>> gene_expression [-2:]
[0.0012817, 6.23e-06]
>>> gene_expression[1:3]=[2.45e-06]
>>> print (gene_expression)
['braca1', 2.45e-06, 6.23e-06]
>>> gene_expression+[4.3301,9.02e-02]
['braca1', 2.45e-06, 6.23e-06, 4.3301, 0.0902]
```

Fig. 2. Pseudocode to compute data structures of lists and different methods used in list.

```
>>> gene_expression [-2:]
[0.0012817, 6.23e-06]
>>> gene_expression[1:3]=[2.45e-06]
>>> print (gene_expression)
['brca1', 2.45e-06, 6.23e-06]
```

Like string, concatenation operator is also used in list to add new data in it.

```
>>> gene_expression+[4.3301,9.02e-02]
['brca1', 5.19e-08, 0.0012817, 6.23e-06, 4.3301, 0.0902]
```

Some functions used in strings are also applicable in list such as deland Len functions.

```
>>>len(gene_expression)
4
>>>del gene_expression[3]
>>> gene_expression
['braca1', 5.19e-08, 0.0012817]
```

Extend method is used in which all items appended in a list.

```
>>>gene_expression. Extend ([6.23e-06, 4.3301, 0.0902])
```

```
>>> gene_expression
['braca1', 5.19e-08, 0.0012817, 6.23e-06, 4.3301,
0.0902]
```

Sorting of list is another important method use in Python programming. This method is often used. Suppose a variable name my list is created with list of numbers, sort function will sort the elements of list numerically. Here there are two different sort methods are used to sort same list.

```
>>>mylist= [23,3,9,12,13,5,.5]
>>>sorted (mylist)
[0.5, 3, 5, 9, 12, 13, 23]
>>>mylist.sort ()
>>>mylist
[0.5, 3, 5, 9, 12, 13, 23]
```

Tuples

List and string tuples are also important data structure which consist a number of values separated by commas. Unlike list tuples are immutable, and usually contain heterogeneous sequences elements. Tuples does not contain any string type. Tuples also have index.

```
>>> t= 1, 2,3,4,5
>>>t
(1, 2, 3, 4, 5)
```

Sets

Another data structure is the set. A set is an ordered collection with no duplicate elements. In other words, sets are lists with no duplicate entries, and since they are in order, they don't have an index. Same like a mathematical terms of sets having union and intersection properties (Perkins, 2010).

Let's see an example of set with respect to gene data. The gene ontology annotations associated with the gene brca1. Mutations in this gene are responsible for approximately 40% of inherited breast cancers, and more than 80% of inherited ovarian cancers.

To create a set of terms for brca1, we introduce by mistake, the term DNA repair twice. If we check the brca1 variable now, we notice that Python removed the duplicated element. So, DNA repair element only appears once(Cock *et al.*, 2009).

```
>>> brca1= {'DNA repair','zinc ion binding','DNAbinding','proteinubiquitination','DNA repair'}
>>> brca1
{'DNA repair', 'DNA binding', 'zinc ion binding', 'protein ubiquitination'}
```

So, in this way many sets can be created, as mentioned earlier that sets can perform arithmetic operation so some special keys can be used in that term.

To concatenate or union(|)this sign is used between two sets name.

```
>>> brca1 | brca2
{'double stranded break repair', 'protein ubiquitination', 'zinc ion binding', 'heat shock protein', 'DNA repair', 'h4 histone acetyltransferase activity', 'DNA binding'}
```

(&) operator is used for intersection and (–) is used for difference between two sets.

Dictionaries

Dictionaries are those data structures in which data is saved for further Python programming. Dictionaries can stored multiple values in it, and make a reference key to store multiple values relates with key(Goodstadt, 2010). DNA sequences are of different types, transcription factor motifs are one of type which might have different motifs which have different reference DNA sequences. Keys which is used as motifs are immutable, it can be numbers or even strings(Przulj, 2013).

```
TF_motif = {'sp1': 'gggcgg', 'c/EBP':'attgcgcaat','oct-1':'cacagtgt'}
```

To obtain any value from dictionary just type its key within square brackets.

```
>>>print ("the recognition sequence for sp1 transcription is %s." % TF_motif ['sp1'])
The recognition sequence for sp1 transcription is gggcgg.
```

To add new key in dictionary following statement is used,

>>> TF_motif ['AP-1'] ='tgagtca'

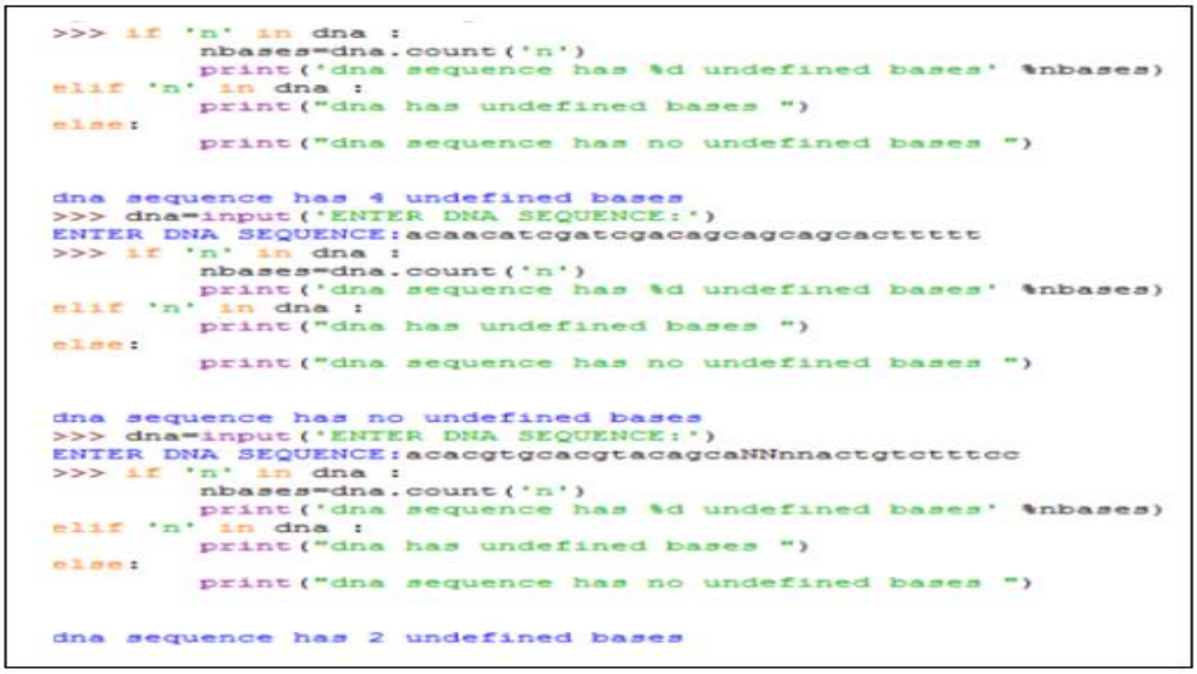
>>> TF_motif

{'sp1': 'gggcgg', 'c/EBP': 'attgcgcaat', 'oct-1': 'cacagtgt', 'AP-1': 'tgagtca'}

To get the length of dictionary Len function is used.

Statements and loops

Statements and loops are used in decision making in Python programming, if one or more condition can be applied to same program then run time of a program should be considered(Pearson and Lipman, 1988). To make program precise with least run time, statements will be used. If, else andelif (used to test several conditions in one structure) are some basic statements used in Python. Let's generate a block of code to see how statements can executes in if condition. Suppose there are some undefined bases in DNA sequence to find those bases following condition is used. Figure 3 shows the "If" statements execution when below given condition is true.



```
>>> if 'n' in dna :
        nbases=dna.count('n')
        print('dna sequence has %d undefined bases' %nbases)
elif 'n' in dna :
        print("dna has undefined bases ")
else:
        print("dna sequence has no undefined bases ")

dna sequence has 4 undefined bases
>>> dna=input('ENTER DNA SEQUENCE:')
ENTER DNA SEQUENCE:acaacatcgatcgacagcagcagcacttttt
>>> if 'n' in dna :
        nbases=dna.count('n')
        print('dna sequence has %d undefined bases' %nbases)
elif 'n' in dna :
        print("dna has undefined bases ")
else:
        print("dna sequence has no undefined bases ")

dna sequence has no undefined bases
>>> dna=input('ENTER DNA SEQUENCE:')
ENTER DNA SEQUENCE:acacgtgcacgtacagcaNNnnactgtctttcc
>>> if 'n' in dna :
        nbases=dna.count('n')
        print('dna sequence has %d undefined bases' %nbases)
elif 'n' in dna :
        print("dna has undefined bases ")
else:
        print("dna sequence has no undefined bases ")

dna sequence has 2 undefined bases
```

Fig. 3. Pseudocodes to compute statements and conditions through if's and else's statement.

If, elif and else statements

>>>dna=input ('ENTER DNA SEQUENCE:')

ENTER DNA SEQUENCE: aaaaacgactgtgacnnnnaccgtactac

>>>if 'n' in dna :

nbases=dna. Count ('n')

print ('dna sequence has %d undefined bases' %nbases)

DNA sequence has 4 undefined bases.

if 'n' in dna :

nbases=dna. count ('n')

print ('dna sequence has %d undefined bases' % nbases)

elif 'n' in dna :

print ("dna has undefined bases ")

else:

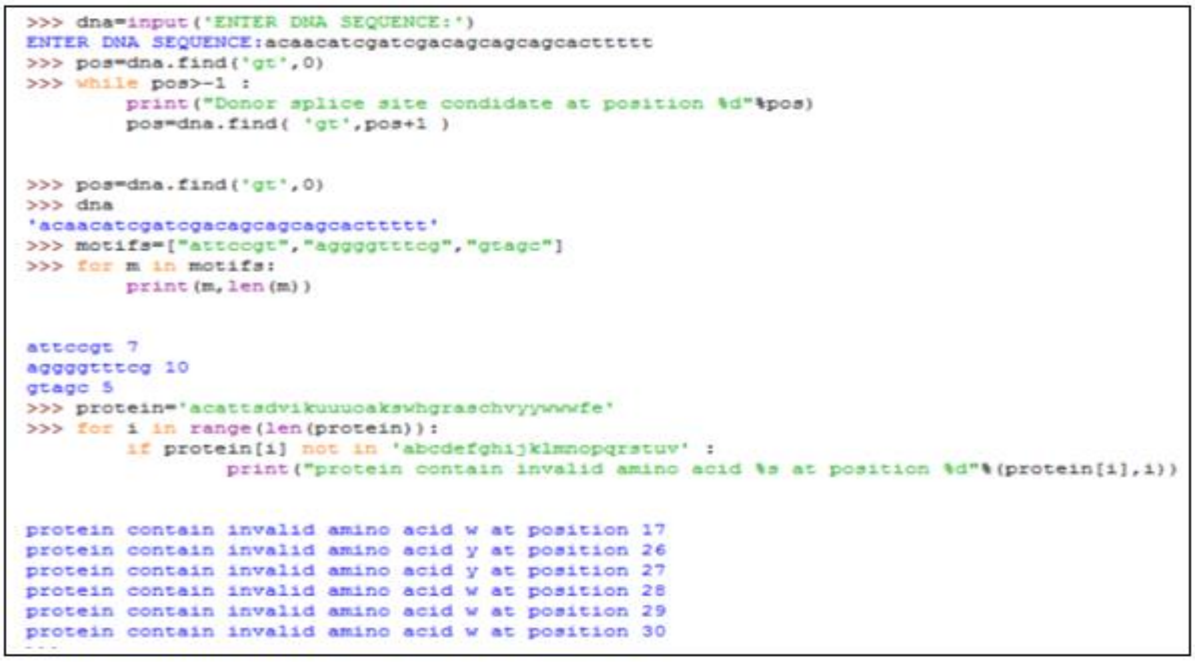
print ("dna sequence has no undefined bases ")

dna sequence has no undefined bases.

The condition in statements is called as Boolean expressions, which is either true or false(Anders *et al.*, 2014). Boolean expression are formed with the help of comparison, identity and membership operators.

Loops

Loops are another important and core operations in any programming language, major aspect of loops is less run time and precise code(Pearson and Lipman, 1988). Rather than computing a large code in loops, add a small block of code and execute it over and over again. So loops are those operations in data structures which is used to execute different statement of "if" and "else". Loops work on different conditions if conditions are true, loop is executed.



```
>>> dna=input('ENTER DNA SEQUENCE:')
ENTER DNA SEQUENCE:acaacatcgatcgacagcagcagcacttttt
>>> pos=dna.find('gt',0)
>>> while pos>-1 :
        print("Donor splice site condidate at position %d"%pos)
        pos=dna.find( 'gt',pos+1 )


>>> pos=dna.find('gt',0)
>>> dna
'acaacatcgatcgacagcagcagcacttttt'
>>> motifs=["attccgt","aggggtttcg","gtagc"]
>>> for m in motifs:
        print(m,len(m))


attccgt 7
aggggtttcg 10
gtagc 5
>>> protein='acattsdvikuuuoakswhgraschvyywwwfe'
>>> for i in range(len(protein)):
        if protein[i] not in 'abcdefghijklmnopqrstuv' :
                print("protein contain invalid amino acid %s at position %d"%(protein[i],i))


protein contain invalid amino acid w at position 17
protein contain invalid amino acid y at position 26
protein contain invalid amino acid y at position 27
protein contain invalid amino acid w at position 28
protein contain invalid amino acid w at position 29
protein contain invalid amino acid w at position 30
```

Fig. 4. Pseudocode of while and for loops to compute all positions of donor splice site candidates in the sequences, with the help of while loop and for loop, and also unknown protein sequences were identified.

Now let's discuss a program compiled in Python with the help of loops. Figure 4 shows two types of loops for loop and while loop.

```
ENTER DNA SEQUENCE: acaacatcgatcgacagcagcagcacttttt
>>>pos=dna.find ('gt',0)
>>> while pos>-1 :condition statement
print("Donor splice site condidate at position %d"%pos)
pos=dna.find( 'gt',pos+1 ) block of code to execute while loop if true.
>>>pos=dna.find('gt',0)
>>>dna
'acaacatcgatcgacagcagcagcacttttt'
>>>motifs= ["attccgt","aggggtttcg","gtagc"]
>>>for m in motifs:
print(m,len(m))
attccgt 7
aggggtttcg 10
gtagc 5
>>>protein='acattsdvikuuuoakswhgraschvyywwwfe'
>>>for i in range (len (protein)):
If protein[i] not in
abcdefghijklmnopqrstuv':
Print ("protein contain invalid amino acid %s at position %d"% (protein[i],i))
Protein contain invalid amino acid w at position 17
Protein contain invalid amino acid y at position 26
Protein contain invalid amino acid y at position 27
Protein contain invalid amino acid w at position 28
Protein contain invalid amino acid w at position 29
Protein contain invalid amino acid w at position 30
```

References

Anders S, Pyl PT, Huber W. 2014. HTSeq–a Python framework to work with high-throughput sequencing data. Bioinformatics **32(2),** 166-169. http://dx.doi.org/10.1093/bioinformatics/btu638

Cock PJ, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, Friedberg I, Hamelryck T, Kauff F, Wilczynski B. 2009. Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics **25(11),** 1422-1423. http://dx.doi.org/10.1093/bioinformatics/btp163

Goodstadt L. 2010. Ruffus: a lightweight Python library for computational pipelines. Bioinformatics **26(21),** 2778-2779. www.10.1093/bioinformatics/btq524

Hamelryck T, Manderick B. 2003. PDB file parser and structure class implemented in Python. Bioinformatics **19(17)**, 2308-2310.
http://dx.doi.org/10.1093/bioinformatics/btg299

Mann C. 2010. Python for bioinformatics. Kybernetes **39(8)**
http://dx.doi.org/10.1108/k.2010.06739hae.004

Lesk A. 2013. Introduction to bioinformatics. Oxford University Press.

List M, Ebert P, Albrecht F. 2017. Ten Simple Rules for Developing Usable Software in Computational Biology. PLOS Computational Biology **13(1),** e1005265.
http://dx.doi.org/10.1371/journal.pcbi.1005265

Oliphant TE. 2007. Python for scientific computing. Computing in Science & Engineering **9(3)**, 10-20.
http://dx.doi.org/10.1109/mcse.2007.58

Pearson WR, Lipman DJ. 1988. Improved tools for biological sequence comparison. Proceedings of the National Academy of Sciences **85(8)**, 2444-2448.
http://dx.doi.org/10.1073/pnas.85.8.2444

Perkins J. 2010. Python text processing with NLTK 2.0 cookbook. Packt Publ.

Przulj N. 2013. Introduction to the special issue on biological networks. Internet Mathematics **7(4),** 207-208.
http://dx.doi.org/10.1080/15427951.2011.621769