LEARN PYTHON & R FOR BIOINFORMATICS

Prerequisite Terminologies:

In order to have a thorough understanding of our main topic, you should have the basic concept of the following terminologies:

1. Parameters/arguments.
2. Modularizing the code in Python.

Introduction:

A Function is a block/set of organized and reusable code that is utilized to perform a single, related function. Functions provide better modularity of your program and high level of code reusing. Basically there are two types of functions:

1. **Built-in functions-** The Python built-in functions are defined as the functions whose functionality is pre-defined in Python. E.g., **print().**
2. **User-defined functions-** A function that you define yourself in a program is known as a user-defined function. However, you cannot use the Python keywords as function names.

Advantages of using functions in your code:

Functions are the standard way of modularizing your code in Python. The biggest advantage of using functions in your code is its code reusability since you can call that particular function in your code whenever it is needed, rather than writing the entire code again and again.

Syntax:

```
def Function_Name ( Parameters ):
Statement_1
Statement_2
.
:
Statement_n
return [ expression ]
```

- ➔ **“def”** is the keyword used to define the function in Python.
- ➔ **“Function_Name”**, enter the name of your function, use any name that relates to that particular code/function.
- ➔ **“(Parameters)”** defines the particular arguments/parameters of your function.
- ➔ Function definition must be terminated by using **“:”**.
- ➔ **“return”** to return the value/output of that particular function into the program where that particular function has been called. It's very important to return something from a function otherwise it's not much useful to modularize your code using functions.

Steps:

- ➢ **Returning a single value from a particular function:**

[For better understanding of this concept take the function below **“**Function to calculate the length of a sequence” as an example.**]**

- Declare a function, keeping the syntax in your mind, as:

```
def figure_out_length(receivedSequence):
length = len(receivedSequence)
return length
```

  ★ *len()* is a built-in function of Python.

  [This function will receive the sequence from the main program and will calculate the length of the received sequence and return the length of the sequence to the main program.]

Note: Functions are always declared above the main program, in Python.

- In the main program, ask the user to enter the input sequence or you can input a hardcoded sequence and store the sequence in a variable.
- Call the function and use the variable_name as a parameter in the function you’ve declared above, and declare a variable to store the output of the function in the main function, as:

```
mySequence = input(“Enter Your Sequence:”)
mySeqLength = figure_out_length(mySequence)
```

Note: You must declare a variable in the main program to store the returned value of the function, otherwise you won’t be able to get the output.

- Print out the results on the output windows, as

```
print(mySequence, mySeqLength)
```

  [It will print your sequence and sequence length respectively.]

Note: Since, Python is considered as a dynamic language which means you cannot enter integer type data in the variable storing a string type data type. So, if you're not declaring a variable to be a string variable, it can be an integer variable as well.

➢ **Returning Multiple values from a particular function:**
[For better understanding of this concept take the function given below "Function to calculate the charge and sequence length of a protein" as an example.**]**

- Declare a function, keeping the syntax in mind, as:

```
def getCharge(proteinSequnce):
"""" Return the net charge of a protein sequence """"
myProt = proteinSequence.upper()
charge= -0.002
aminoCharge = {"Amino acid Charge library"}
for aminoAcid in myProt:
        Charge += aminoCharge.get(aminoAcid, 0)
return charge, len(proteinSequence)
```

[This function will calculate the charge of your Protein and will also calculate the length of the sequence.]

- Following table describes the function of each statement written above in the function:

def getCharge(proteinSequnce):	➔ Defining your function in Python.
"""" Return the net charge of a protein sequence """"	➔ Description of your function.
myProt = proteinSequence.upper()	➔ Initializing the variable *myProt* and converting the protein sequence in upper case.
charge= -0.002	➔ Calculation of charge on the protein. [for better understanding of this concept, watch the previous video on 'Protein

	Charge Calculation’.]
aminoCharge = {“Amino acid Charge library”}	➔ Adding amino acid charge library to calculate charge on the protein accordingly.
for aminoAcid in myProt: *Charge +=aminoCharge.get(aminoAcid, 0)*	➔ For loop to calculate charge on the entire sequence length of protein by calculating charge on individual amino acid residues.
return charge, len(proteinSequence)	➔ After calculation of charge and sequence length, the values would be returned to the main program, where the function has been called.

- In the main program, declare two variables to store the returned values from the function and call the function, as:

  myCharge, myLength = getCharge(“Enter your Protein Sequence”)

- To print your output, call the the print() function, as:

  print(myCharge, myLength)

- To print out the value of only one variable, use index method, as:

  myCharge2 = getCharge(“Enter your Protein Sequence”) [0]
  print(myCharge2)
- You can set a default value to a particular argument/parameter in the Function, in order to reduce the chances of error. It is an important step of code modularization. For example,

  def getCharge(proteinSequence = “AKMPYTAVTTKKMMP”):

[In this way, if the user did not enter the input sequence, the program will run automatically using the default sequence.]

Summary:

In this video we learned how to create functions in Python and how we can use these user-defined functions to analyze the biological data. We learned two different ways of returning values from the functions into the main program. We also learned how to set a default value to a parameter in the function in order to reduce the chances of errors in our code.