Table of Contents

1 Introduction to Python Programming

Most of you might be already aware of the importance of learning a programming language. However, as a person with a science or biology background, you may feel intimidated by the concepts associated with programming and the technical jargons that seem to add another level of complexity to it. The programming world may seem like a completely new world, far distant from your own. You might have tried different ways to learn a programming language that can help in your work, either from your peers or even online. There is a high chance, you have been told to "Learn Python."

So, what is Python? Python is an object-oriented programming (OOP) language, which simply means, creating objects that can help you solve a given problem. If you would like to read a bit more about it, you can find the information in the Reference 1: .

However, if it feels too overwhelming, don't worry. This book has been designed to walk with you throughout this journey of learning Python, without getting lost.

Python is a free, open-source programming language that is simple, flexible, and renowned for its readability and user-friendliness. Guido van Rossum developed Python in the late 1980s, and it has since grown to be an effective tool for a wide range of applications, including scientific computing and web development.

It is easy to learn, with simple syntax similar to the English language. It is also free, open-source, and with a plethora of applications.

Key Features of Python:

- Readability: Python places a strong emphasis on clear, readable code, making it understandable to both novices and professionals.
- Versatility: Python's versatility includes support for procedural, object-oriented, and functional programming paradigms.
- Interpreted Nature: Python's interpreted nature makes it possible for quick development and simple debugging

2 Why Python for Biologists?

Owing to its ease of use and rich library of functions, Python is a popular choice in biological sciences. Also, because of its readability, it is a great option for researchers who may not have a strong programming background.

2.1 Accessibility:

- Python's code is clear and straightforward, allowing researchers to concentrate on solving biological problems rather than struggling with complex terms and jargon words.
- Python's syntax is simple, lowering the learning curve for biologists venturing into the realm of programming.

2.2 Extensive Libraries:

- NumPy, Pandas, and Biopython are just a few of the many libraries and frameworks that Python offers that are specifically designed for the analysis and manipulation of biological data.
- By making complicated tasks simpler, these libraries enable biologists to analyze data effectively.

2.3 Community Support:

- The Python community is active and supportive, making significant contributions to the language's advancement.
- Online forums, tutorials, and documentation makes it easier for biologists to ask for assistance and exchange their expertise.

2.4 Interdisciplinary Applications:

- Python is a great choice for interdisciplinary study because of its adaptability, which goes beyond biology.
- Integration with additional tools and languages facilitates collaboration with experts from diverse fields.

3 Setting up Python

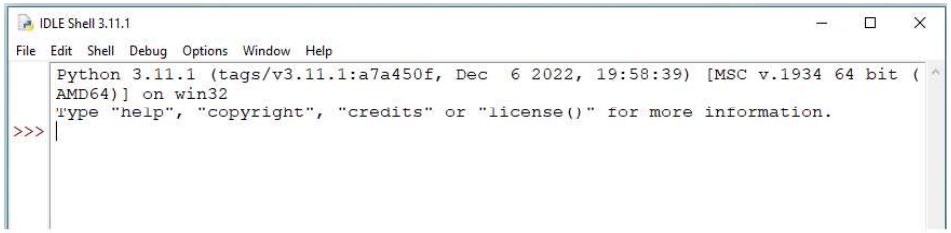
Python is thus a popular, easy-to-learn, and very powerful programming language, which is used in software and web development, data science, machine learning, and many other fields. So, let's move ahead and install it on your computer.

3.1 Installation:

Let me briefly give you an idea about such installations. The system requires a base distribution of software, which is good enough for your work. Then you need an editor that provides more flexibility and better user interface. So, when you install Python, you will see the installation of python and its editor (called as IDLE) in your system. Follow following steps for the installation:

- Go to the Python download page at https://www.python.org/downloads/ (Reference 2) and download the latest python version as per your system (Windows / Mac)
- When the download is completed, double-click the file and follow the instructions to install it.
- Once the installation is completed, You will see something like this in the start-up menu program list:

- Open the IDLE (Python ...) link. It should look like this:



- You can start writing codes here.

4 Basics

4.1 Interactive Mode Programming

Starting the IDLE, the interpreter, without using or opening any file, gives the following prompt (as also seen in the above image):

```
Type "help", "copyright", "credits" or "license" for more information.
\>>>
```

Type the following text at the Python prompt and press the Enter:

```
3+2
```

You can also type the following and press Enter:

```
print("Hello, Python!")
```

What you did just now is called interactive Mode Programming. The script is not saved and the moment you close the IDLE, everything is lost. So, it is preferred to use Script Mode programming.

4.2 *Script Mode Programming*

Here, we are saving the script as a .py file. Let us write a simple Python program in a script file.

- To create a file in IDLE, go to file>New window (Ctrl+N)
- Write the following text in the new window:

  print("Hello!, I am learning Python")

  Note: The 'p' is small in print
- Save (Ctrl+S) the file. Python files have default extension .py. You can name the file as: 'my_first_program.py'
- Go to the tab list at the top of this window and press run>run module (F5), and you can see the output in the IDLE window.

Here, we have invoked the interpreter/IDLE with a script/saved file; the IDLE begins the execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active. Every time, you need to write script to the file, save it and execute to see the output in the IDLE.

So, Congratulations, you've successfully run the first Python program

4.3 Writing clean script

```
1 print("Python Lecture")
2 print(2 + 2) # clean script
3 print(5+4-1)
4 print (2+2) # avoid these kind of spacing
```

Note: Please note that numbers written at the start of each statement is not the part of code, it's just showing the line number.

Consider the line of codes shown above: In the second line, numbers are well-spaced and easy to read. In the last two, spaces are not proper, you need to avoid such kind of spacing for clean code practice and make your codes easy to read for other users.

4.4 Basic Python Syntax:

There are set of guidelines that specify how Python code should be written and organized is known as Python syntax. Knowing the fundamentals of syntax is essential to developing readable, error-free, and understandable code.

4.4.1 Introduction to Input and Output:

In Python, Input and Output (I/O) operations allow interaction with users and the external environment. Understanding basic input and output operations is essential for creating interactive programs, handling user input, and saving/loading data from external sources. These skills lay the groundwork for more complex applications, especially in the context of biological data analysis and processing.

4.4.1.1 Printing Output

The `print()` function is used to display information to the console. Use the `print()` function to display the value of a variable in python. In fact, you can directly print a text string using quotes in print command. See the examples below

```
1 x = 1
2 print(x)
3 print("Hello, World!")
4 print("Sum =", x + 5)
```

```
Output
1
Hello, World!
Sum = 6
```

As shown above in line 4, the two different printing jobs can be concatenated using commas between arguments of print().

Multiple values can be printed using commas.

```
name = "Alice"
age = 25
print("Name:", name, "Age:", age)
```

```
Output:
Name: Alice Age: 25
```

4.5 Taking User Input with `input()`:

The `input()` function allows the program to accept input from the user. By default, `input()` returns the user's input as a string.

```
user_input = input("Enter your name: ")
```

When you run this prompt will ask for the input with the statement flashed "Enter your name: ". You need to enter the text in the IDLE and press enter. The text value entered is saved under the variable user_input. Let say, you type "Sam". Now, the next line can be:

```
print("Hello, " + user_input + "!")
```

```
Output:
Hello, Sam!
```

This will use the information fed by the user.

4.5.1 Indentation

Python uses **indentation** (usually 4 spaces) to define blocks of code.

```
1 if True:
2     print("Hello")
```

```
Output:
Hello
```

In other programming languages like R, curly braces {} - define a block of code, whereas in Python, blocks of code are denoted by line indentation, which is rigidly enforced. This will be clearer when we learn the codes for loops in later chapters.

4.5.2 Comments

You must have observed above that any statement or characters written after the symbol #, and up to the end of the physical line, is ignored by python interpreter. They are called a comment(s), denoted by a hash sign (#) at the start of the line. For example -

```
# First Comment
print("Hello Python!") # Second comment
```

```
Output:
Hello Python!
```

You can type a comment on the same line after a statement or expression –

```
1 name = "Sam" # This is again a comment
```

You can comment on multiple lines as follows –

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

The Python interpreter also ignores the following triple-quoted string (''' or """) and can be used as multiline comments:

```
'''
This is a multiline
comment.
'''

```

4.5.3 Multiline statements/Line continuation

If the script in one line is too long and to increase your readability, you want to take some parts to next line by pressing "Enter", it will throw an error. The right way to do it is by adding the slash ("\") symbol at the end of the line. This is more useful when you are working in interactive mode and want to write multiline code. For example:

```
a = 1
b = 2
c = 3
total = a + \
        b + \
        c

print(total)
```

However, statements that contain [], {}, or () brackets do not require line continuation characters. For example –

```
sum = (1 +
       2 +
       3)
print(sum)
```