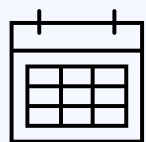




Two-Days Online Workshop on BIOPYTHON FOR BIOLOGISTS



**21-22 February, 2025
Saturday & Sunday**



Organised by
NextGen Life Sciences Pvt. Ltd.
New Delhi, INDIA

About NextGen Life Sciences

- ISO 9001:2015 certified company.
- Established in 2014, aiming to be an one-stop solution for Molecular Biology, Cell Biology, Immunology, Healthcare, Drug Development research products.
- Introduced own brand, **SciPhi™**, adhering to international standards under the “Make in India” initiative.
- Actively expanding product portfolio to meet evolving customer needs.
- Offering training and short-term internship programs for young minds in the Life Science and Biotechnology Industry.
- Prioritizing Industry-Academia collaboration for the benefit of the young generation.



Dr. Shivani Kumar
Application Scientist
mumbai.team@nextgenlife.com



Ms. Vasudha Pai
Inside Sales Representative
techteam@nextgenlife.com



Mr. Adil Parvez
R & D Assistant
techteam@nextgenlife.com



Dr. Kamal Shrivastava
R & D Manager
kamal@nextgenlife.com



Dr. Largee Biswas
Product Manager
techteam@nextgenlife.com



Dr. Nagma Abbasi
CEO and Founder
nagma@nextgenlife.com

Topics to be covered

- Introduction to Python and Biopython
- Python concepts (Lists, Loops, Functions)
- Setting and Installation of Biopython in Google Colab
- Seq and SeqRecord objects
- Sequence Manipulations
- Parsing Metadata From Sequence
- Multiple Sequence Alignment in Google Colab
- Phylogenetic Tree with Biopython
- Interactive Hands on Session
- Q&A Session



What is Python?

Python is a programming language that can be used for many tasks, including web development, data analysis, and software testing. It's easy to learn and use, and it's open source.

Features of Python

1. **Easy to learn:** Python is designed to be easy and fun to use.
2. **Open source:** Anyone can contribute to the development of Python.
3. **Versatile:** Python can be used for many tasks, including web development, data analysis, and software testing.
4. **Efficient:** Python is efficient and can run on many different platforms.
5. **Integrates well:** Python integrates well with all types of systems.
6. **Increases development speed:** Python can help increase development speed.



History of Python

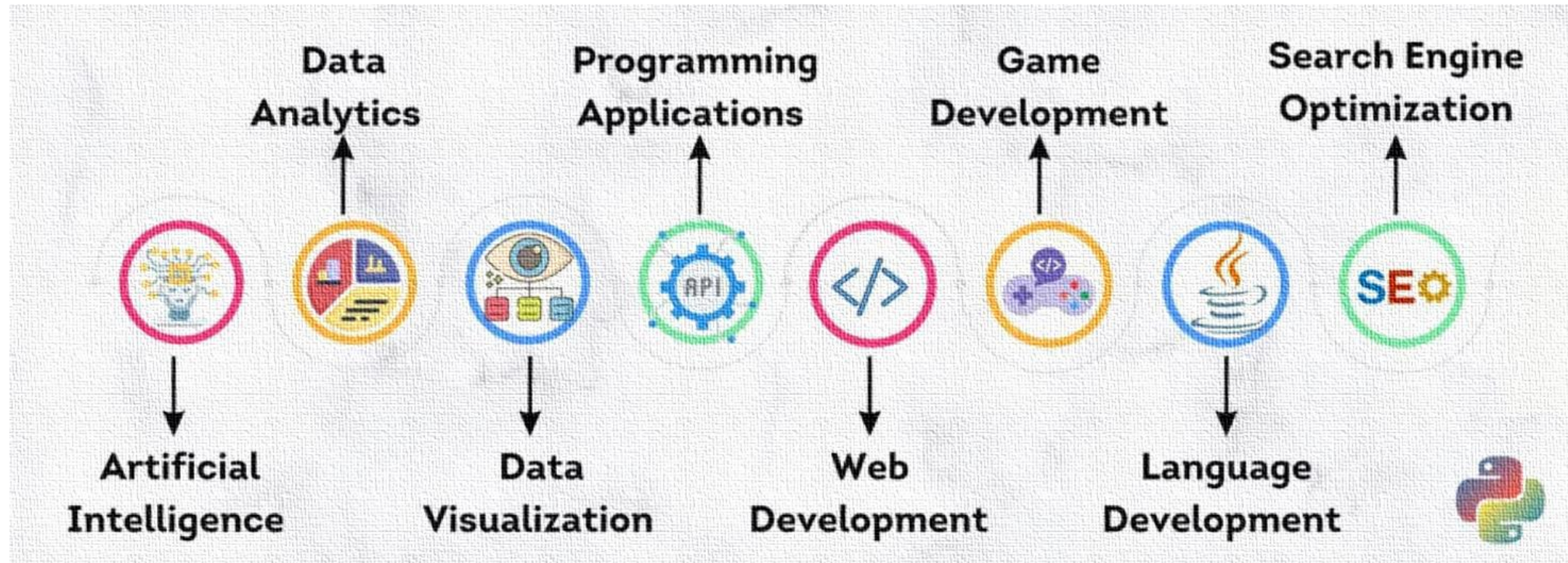
Python is a widely used general-purpose, high-level programming language. It was initially designed by **Guido van Rossum** in **1991** and developed by Python Software Foundation. It was mainly developed to emphasize code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Why Python called Python?

The inspiration for the name came from the BBC's TV Show – '**Monty Python's Flying Circus**', as he was a big fan of the TV show and also he wanted a short, unique and slightly mysterious name for his invention and hence he named it Python!



Applications of Python



Python in Biology

Python is used in biology for a wide range of applications, making complex bioinformatics tasks easier and more efficient. Python simplifies bioinformatics by enabling:

Sequence Analysis – DNA, RNA, and protein manipulation

Data Parsing – Handling FASTA, GenBank files

Multiple Sequence Alignment – Evolutionary comparisons

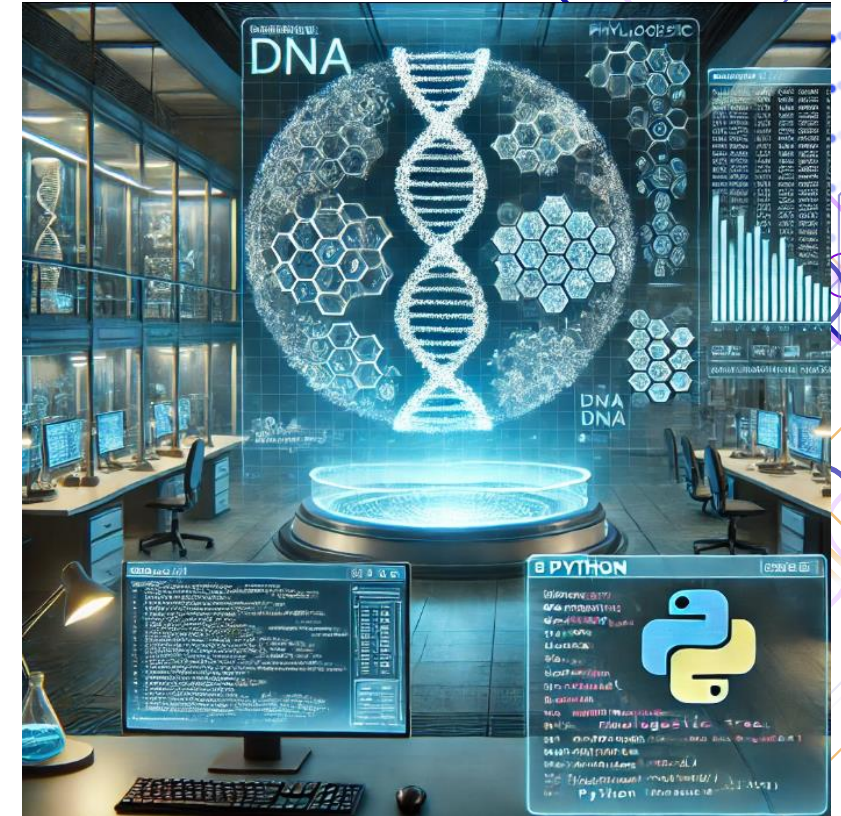
Structural Biology – Protein modeling and visualization

Phylogenetics – Constructing evolutionary trees

Genomics & Transcriptomics – RNA-Seq, variant analysis

Machine Learning – Predicting protein functions & drug discovery

Automation – Running bioinformatics workflows efficiently



Basic concepts of Python

1. Variable

- A **variable** is a name that stores a value in memory.
- It acts as a container for data that can be changed or used in calculations.
- Python does not require declaring variable types explicitly; it detects the type automatically.

python

```
name = "Biopython" # String variable
age = 25 # Integer variable
pi = 3.1416 # Float variable
is_biologist = True # Boolean variable

print(name, age, pi, is_biologist)
```

Types of Variables in Python

Data Type	Description	Example
String (<code>str</code>)	Stores text values	"Hello, Biopython"
Integer (<code>int</code>)	Stores whole numbers	42
Float (<code>float</code>)	Stores decimal numbers	3.14
Boolean (<code>bool</code>)	Represents <code>True</code> or <code>False</code>	<code>True</code>
List (<code>list</code>)	Stores multiple values	["A", "T", "G", "C"]
Tuple (<code>tuple</code>)	Like a list, but immutable	("DNA", "RNA", "Protein")
Dictionary (<code>dict</code>)	Stores key-value pairs	{"A": "Adenine", "T": "Thymine"}

2. Lists

What are Lists?

- A **list** is a collection of elements stored in a single variable.
- Lists are **ordered**, **mutable** (changeable), and allow duplicate values.
- Lists are defined using square brackets [].

```
nucleotides = ["A", "T", "G", "C"]  
print(nucleotides)  
# Output: ['A', 'T', 'G', 'C']
```

Operation	Example	Output
Access an element	<code>nucleotides[0]</code>	<code>'A'</code>
Change an element	<code>nucleotides[1] = "U"</code>	<code>['A', 'U', 'G', 'C']</code>
Add an element	<code>nucleotides.append("N")</code>	<code>['A', 'T', 'G', 'C', 'N']</code>
Remove an element	<code>nucleotides.remove("T")</code>	<code>['A', 'G', 'C']</code>
Loop through a list	<code>for n in nucleotides: print(n)</code>	Prints each element

3. Loops

A loop in Python is a control flow statement that allows a block of code to be executed repeatedly. Loops are essential for automating repetitive tasks and iterating over data structures.

There are two main types of loops in Python:

- **For loop:** Used to iterate over a sequence (like a list, tuple, or string) or other iterable objects.
- **While loop:** Used to repeatedly execute a block of code as long as a condition is true.

Loops are important in Python because they:

- **Provide code reusability:** Avoid writing the same code multiple times.
- **Automate repetitive tasks:** Process large datasets or perform actions multiple times without manual intervention.
- **Iterate over data structures:** Access and manipulate elements within lists, tuples, dictionaries, and other collections.
- **Control program flow:** Execute code blocks conditionally based on specific criteria.

For Loop

Python

```
fruits = ["apple", "banana", "cherry"]  
for fruit in fruits:  
    print(fruit)
```

This code will print each fruit in the list:

Code

```
apple  
banana  
cherry
```

While Loop

Python

```
count = 0  
while count < 5:  
    print("Count:", count)  
    count += 1
```

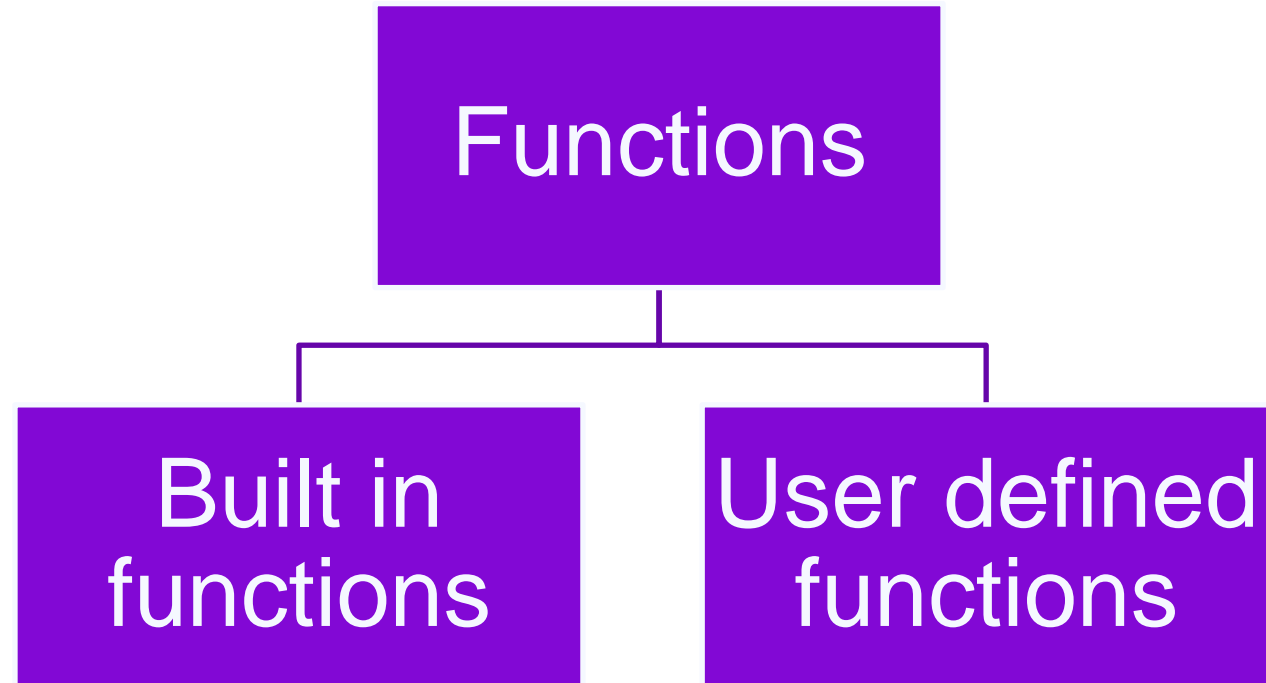
This code will print the count from 0 to 4:

Code

```
Count: 0  
Count: 1  
Count: 2  
Count: 3  
Count: 4
```

What are Functions?

- A **function** is a block of reusable code that performs a specific task.
- Functions **help avoid repetition** and **make code modular**.
- Functions are defined using `def`.



In Python, functions are categorized based on their origin and behavior:

- Built-in functions:**

These are pre-defined functions readily available in Python without requiring additional code or imports. Examples include

`print()`, `len()`, `sum()`, `min()`, `max()`, `abs()`, `range()`, `type()`, `str()`, `float()`, `int()`.

- User-defined functions:**

These are functions created by users to perform specific tasks as per their requirements. They enhance code reusability and organization.

Concept	Description
Variables & Data Types	Store different kinds of data (text, numbers, lists, etc.)
Lists	Store multiple values in a sequence
Conditional Statements	Make decisions using <code>if-else</code>
Loops	Repeat actions (<code>for</code> , <code>while</code>)
Functions	Reusable blocks of code with inputs and outputs
Modules	Use built-in or external libraries (e.g., <code>math</code> , <code>Biopython</code>)
File Handling	Read and write files for data processing

What is Biopython?

Biopython is a collection of free Python tools for computational biology and bioinformatics. It's written by an international team of developers.

It is a distributed collaborative effort to develop Python libraries and applications which address the needs of current and future work in bioinformatics.

What can Biopython do?

- Read and write to a variety of file formats
- Represent biological sequences and sequence annotations
- Access online databases of biological information
- Perform sequence alignment
- Work with protein structure
- Analyze phylogenetics
- Analyze sequence motifs
- Use machine learning
- Analyze population genetics

What is Google Colab ?

Google Colab, or Colaboratory, is a free, cloud-based service that lets you write and run Python code in your browser. You can use it for data analysis, machine learning, and other tasks.

Features

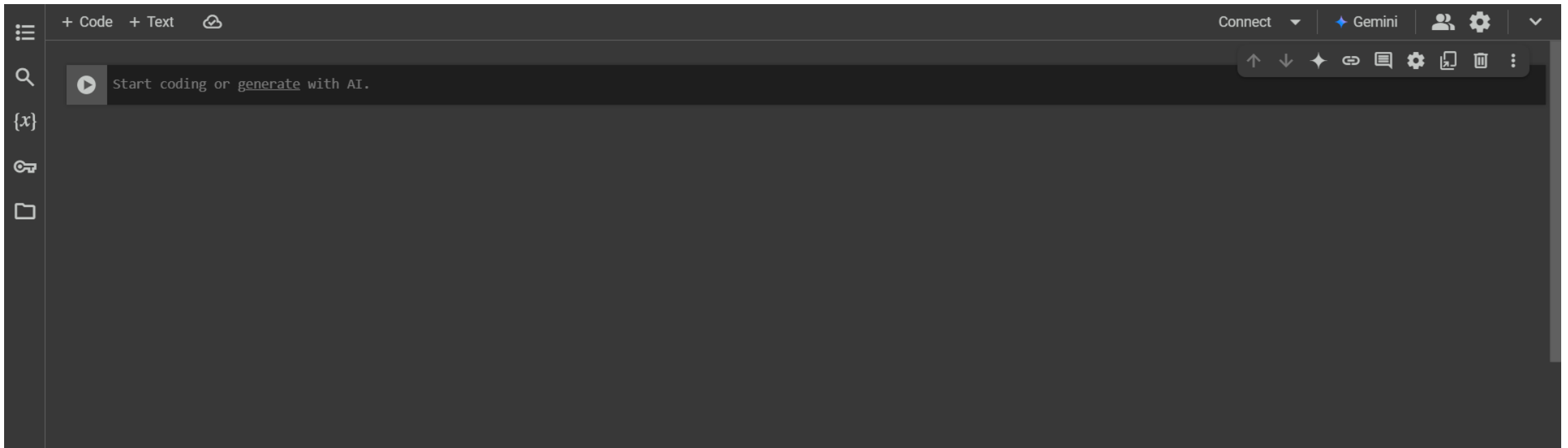
- **Write and execute Python:** You can combine code, images, HTML, and more in your notebooks.
- **Access Google Drive:** You can store and share your notebooks in your Google Drive account.
- **Free access to GPUs:** You can use GPUs and TPUs for free, making Colab a good choice for resource-intensive tasks.
- **Pre-loaded libraries:** Colab has many common libraries pre-loaded for easy import into programs.
- **Rich text:** You can use Markdown formatting, insert images, and more in your text cells.

Use cases

- Analyze and visualize data using Python libraries
- Import image datasets, train image classifiers, and evaluate models
- Leverage the power of GPUs and TPUs for free

Setting Up and Installing Biopython in Google Colab

Google Colab is a cloud-based Jupyter notebook environment that allows you to run Python code without needing to install anything locally. Here's how you can set up Biopython in Colab:



Installing Biopython

Biopython is not pre-installed in Colab, so you need to install it using pip. Run the following command in a Colab cell:

```
!pip install biopython
```



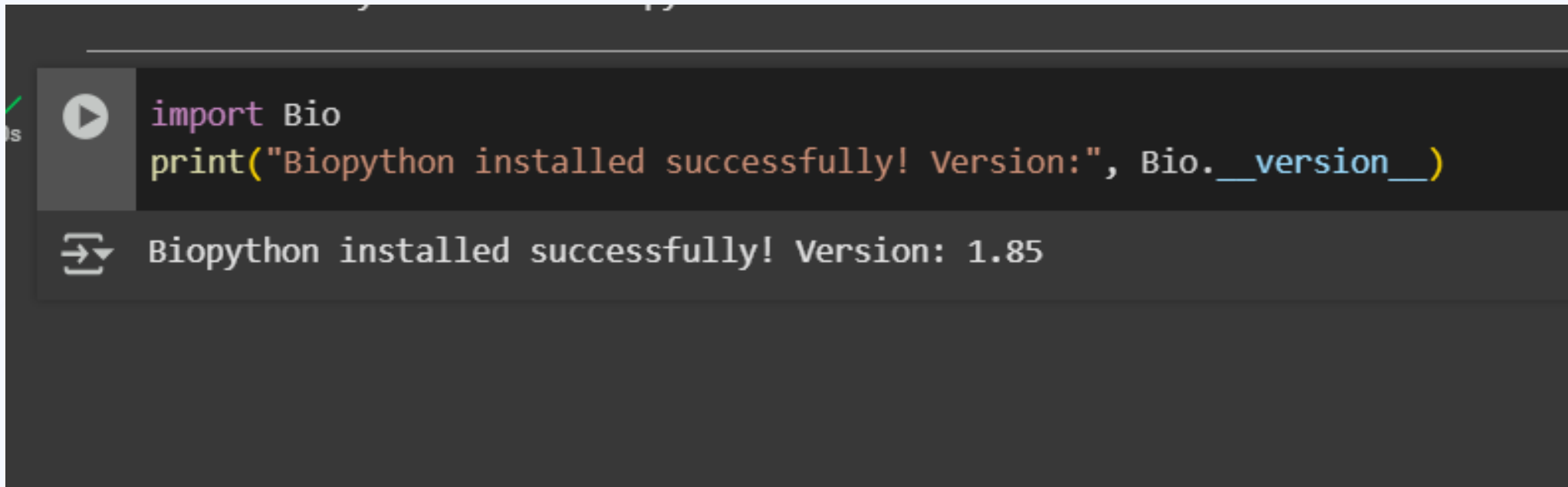
The screenshot shows a Google Colab interface with a code cell. The cell contains the command `!pip install biopython`. The output of the command is displayed below the code, showing the process of collecting and installing the package. The output includes the following text:

```
Collecting biopython
  Downloading biopython-1.85-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (13 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from biopython) (1.26.4)
Downloading biopython-1.85-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.3 MB)
3.3/3.3 MB 18.6 MB/s eta 0:00:00
Installing collected packages: biopython
Successfully installed biopython-1.85
```

Import Biopython Modules

After installation, you can import Biopython and check if it's installed correctly:

```
import Bio
print("Biopython installed successfully! Version:", Bio.__version__)
```



The screenshot shows a Jupyter Notebook interface. The top part is a code editor with a dark background, containing the following Python code:

```
import Bio
print("Biopython installed successfully! Version:", Bio.__version__)
```

Below the code editor is an output area. It features a play button icon on the left and the text output of the code execution:

```
Biopython installed successfully! Version: 1.85
```

Seq and SeqRecord in Biopython

Biopython provides powerful objects to handle biological sequences efficiently. Two of the most important classes in Biopython are **Seq** and **SeqRecord**.

Seq Object

The **Seq** object in Biopython represents a biological sequence (DNA, RNA, or protein). Unlike a normal Python string, it comes with additional built-in functions for biological operations such as **reverse complement**, **transcription**, and **translation**.

Seq Object Operations

Operation	Code Example
Accessing sequence elements	<code>dna_seq[0]</code>
Getting the length	<code>len(dna_seq)</code>
Reverse complement	<code>dna_seq.reverse_complement()</code>
Transcription (DNA → RNA)	<code>dna_seq.transcribe()</code>
Translation (RNA → Protein)	<code>dna_seq.translate()</code>

Creating a Seq Object

✓
0s

```
[3] from Bio.Seq import Seq

# Creating a DNA sequence
dna_seq = Seq("ATGCGA")

print(dna_seq) # Output: ATGCGA
print(type(dna_seq)) # Output: <class 'Bio.Seq.Seq'>
```

```
⇒ ATGCGA
  <class 'Bio.Seq.Seq'>
```


✓
0s



```
from Bio.Seq import Seq

# Creating a DNA sequence
dna_seq = Seq("ATGCGA")

print(dna_seq) # Output: ATGCGA
print(type(dna_seq)) # Output: <class 'Bio.Seq.Seq'>
```



```
ATGCGA
<class 'Bio.Seq.Seq'>
```

✓
0s

```
[4] print("Reverse Complement:", dna_seq.reverse_complement())
```



```
Reverse Complement: TCGCAT
```

✓
0s

```
[5] print("Transcription:", dna_seq.transcribe())
```



```
Transcription: AUGCGA
```

✓
0s



```
print("Translation:", dna_seq.translate())
```



```
Translation: MR
```

Reading FASTA Sequences in Biopython


What is a FASTA File?

- A text-based format for storing DNA, RNA, or protein sequences.
- Each sequence starts with a header line (>), followed by the sequence.

Format of a FASTA definition line


>Seq1 [organism=Carpodacus mexicanus] [clone=6b] actin (act) mRNA, partial CDS
CCTTTATCTAATCTTTGGAGCAYGAGCTGGCATAGTTGGAACCGCCCTCAGCCTCCTCATC


✓
0s



```
from Bio import SeqIO
```

```
# Open and parse the FASTA file
```

```
with open("/content/drive/MyDrive/Colab Notebooks/example.fasta", "r") as fasta_file:  
    for record in SeqIO.parse(fasta_file, "fasta"):  
        print("Sequence ID:", record.id)  
        print("Sequence:", record.seq)  
        print("Sequence Length:", len(record.seq))
```



```
Sequence ID: seq1  
Sequence: ATGCGTAGCTAGCTAGCTAG  
Sequence Length: 20  
Sequence ID: seq2  
Sequence: CGTAGCTAGCTAGCGTACGT  
Sequence Length: 20  
Sequence ID: seq3  
Sequence: GCTAGCTAGCTAGCTAGCTA  
Sequence Length: 20
```

0s



```
!wget -O downloaded_sequence.fasta "https://www.ncbi.nlm.nih.gov/sviewer/viewer.cgi?db=nuccore&id=NM_001301717.2&report=fasta&retmode=text"
```



```
--2025-02-20 15:13:34-- https://www.ncbi.nlm.nih.gov/sviewer/viewer.cgi?db=nuccore&id=NM_001301717.2&report=fasta&retmode=text
```

```
Resolving www.ncbi.nlm.nih.gov (www.ncbi.nlm.nih.gov)... 130.14.29.110, 2607:f220:41e:4290::110
```

```
Connecting to www.ncbi.nlm.nih.gov (www.ncbi.nlm.nih.gov)[130.14.29.110]:443... connected.
```

```
HTTP request sent, awaiting response... 200 OK
```

```
Length: unspecified [text/plain]
```

```
Saving to: 'downloaded_sequence.fasta'
```

```
downloaded_sequence [ <=> ] 2.26K --.-KB/s in 0s
```

```
2025-02-20 15:13:35 (37.6 MB/s) - 'downloaded_sequence.fasta' saved [2319]
```



Hands on Session