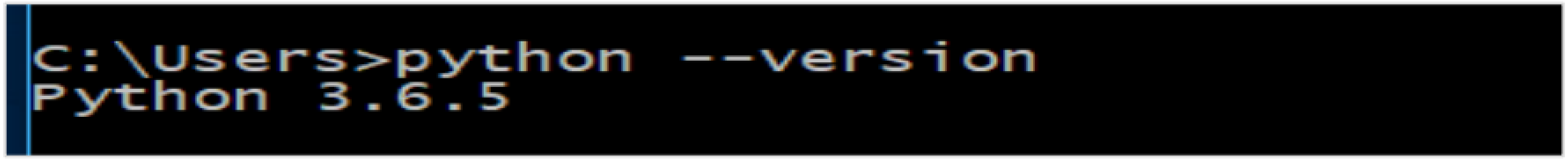
Installation

Step 1: Verifying Python Installation

Biopython is designed to work with Python 2.5 or higher versions. So, it is mandatory that python be installed first. Run the below command in your command prompt:

```
> python --version
```

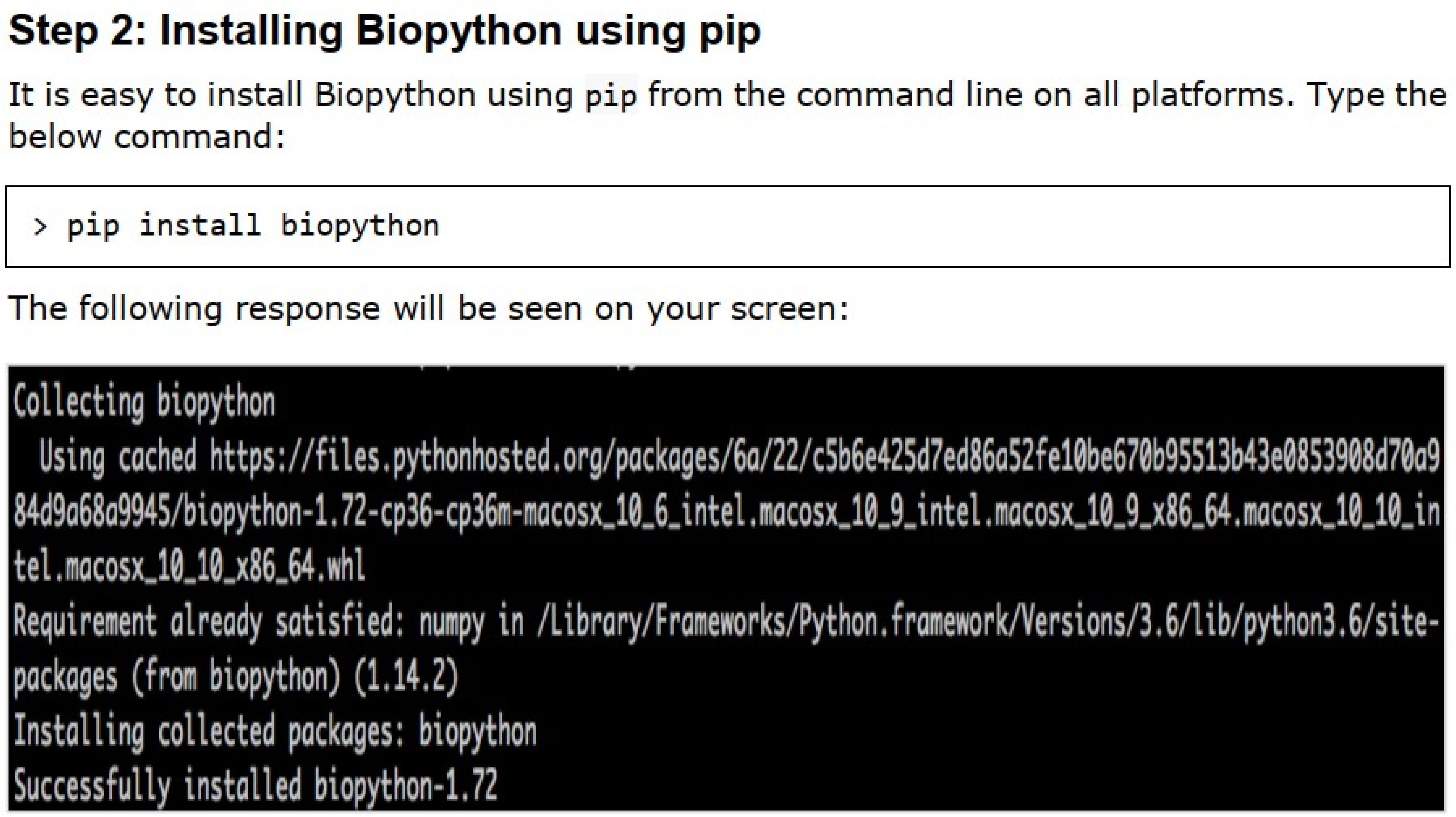
It is defined below:



```
C:\Users>python --version
Python 3.6.5
```

It shows the version of python, if installed properly. Otherwise, download the latest version of the python, install it and then run the command again.

Installation



Step 2: Installing Biopython using pip

It is easy to install Biopython using `pip` from the command line on all platforms. Type the below command:

```
> pip install biopython
```

The following response will be seen on your screen:

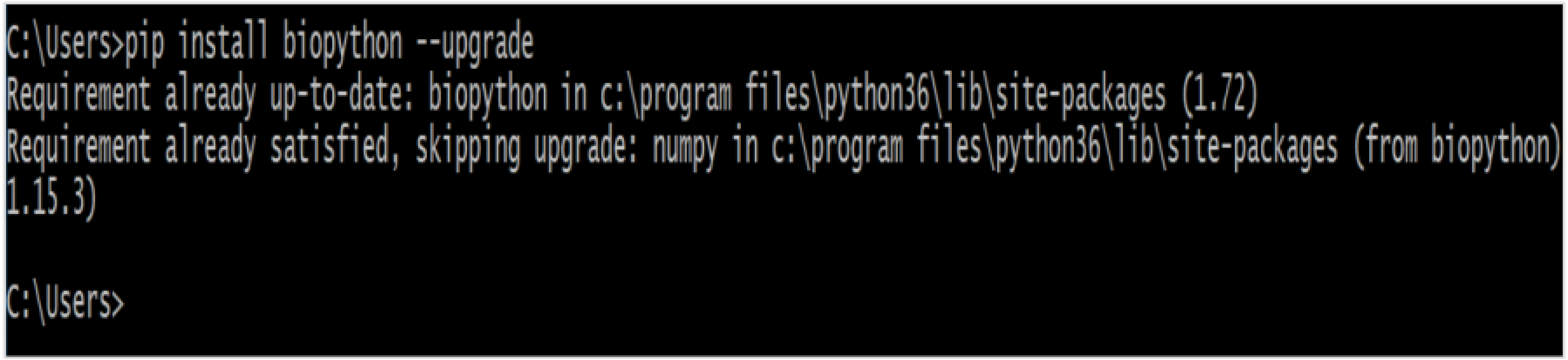
```
Collecting biopython
  Using cached https://files.pythonhosted.org/packages/6a/22/c5b6e425d7ed86a52fe10be670b95513b43e0853908d70a9
84d9a68a9945/biopython-1.72-cp36-cp36m-macosx_10_6_intel.macosx_10_9_intel.macosx_10_9_x86_64.macosx_10_10_in
tel.macosx_10_10_x86_64.whl
Requirement already satisfied: numpy in /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-
packages (from biopython) (1.14.2)
Installing collected packages: biopython
Successfully installed biopython-1.72
```

Installation

For updating an older version of Biopython:

```
> pip install biopython --upgrade
```

The following response will be seen on your screen:

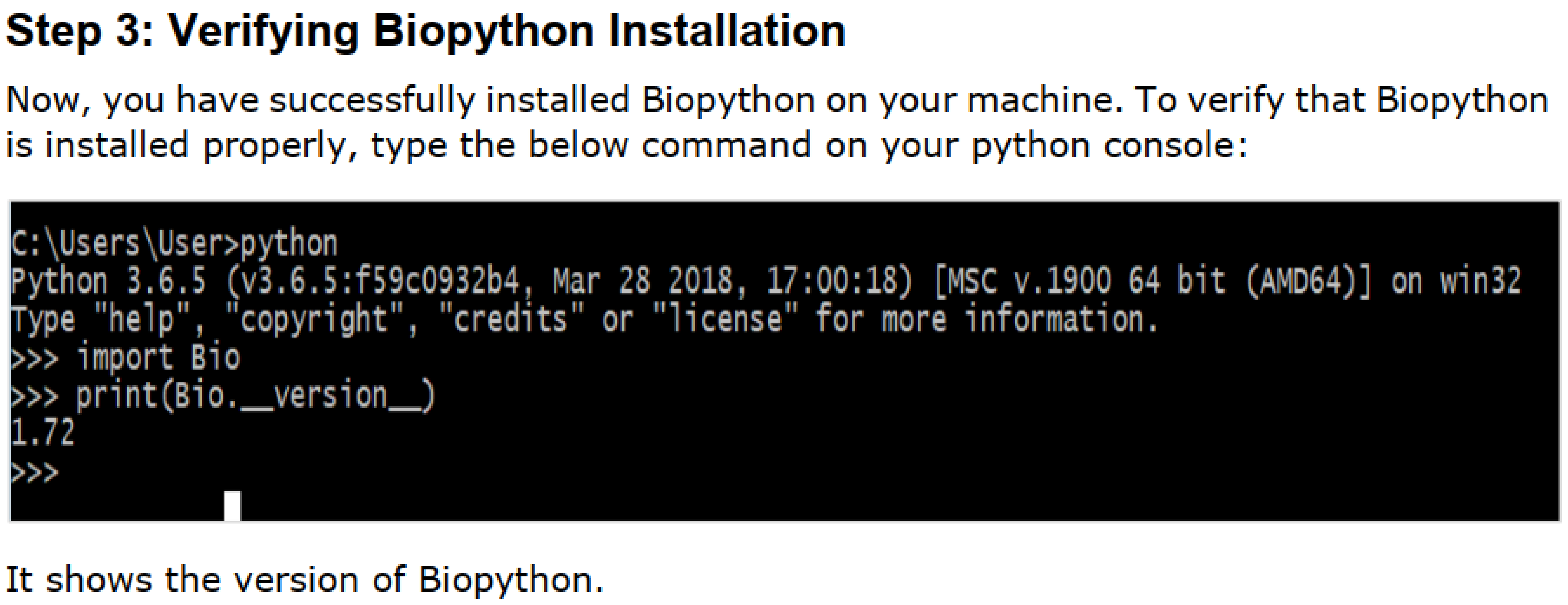


```
C:\Users>pip install biopython --upgrade
Requirement already up-to-date: biopython in c:\program files\python36\lib\site-packages (1.72)
Requirement already satisfied, skipping upgrade: numpy in c:\program files\python36\lib\site-packages (from biopython)
1.15.3)

C:\Users>
```

After executing this command, the older versions of Biopython and NumPy (Biopython depends on it) will be removed before installing the recent versions.

Installation



Step 3: Verifying Biopython Installation

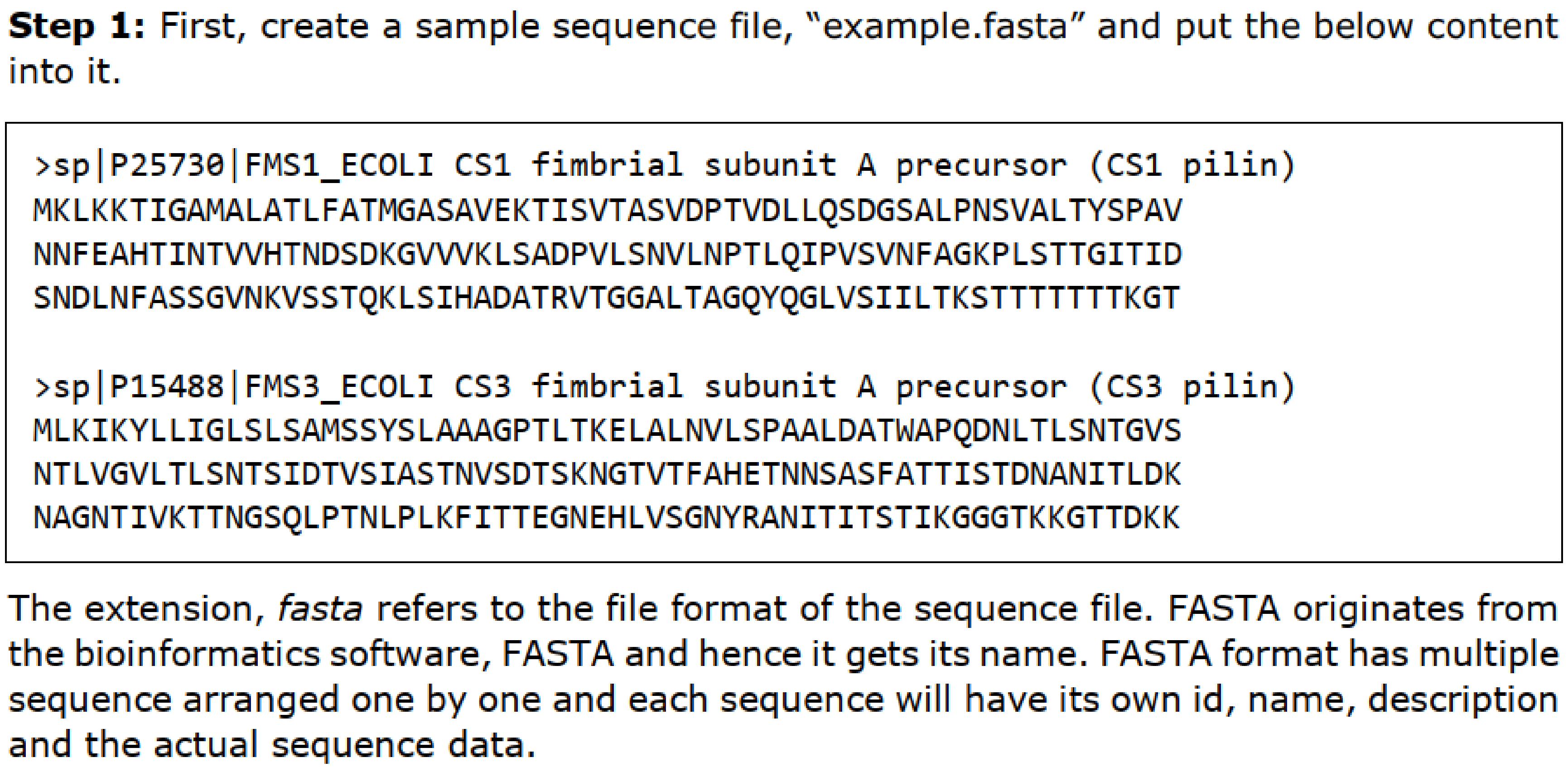
Now, you have successfully installed Biopython on your machine. To verify that Biopython is installed properly, type the below command on your python console:

```
C:\Users\User>python
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import Bio
>>> print(Bio.__version__)
1.72
>>>
```

It shows the version of Biopython.

Creating Simple Application

- Let us create a simple Biopython application to parse a bioinformatics file and print the content.



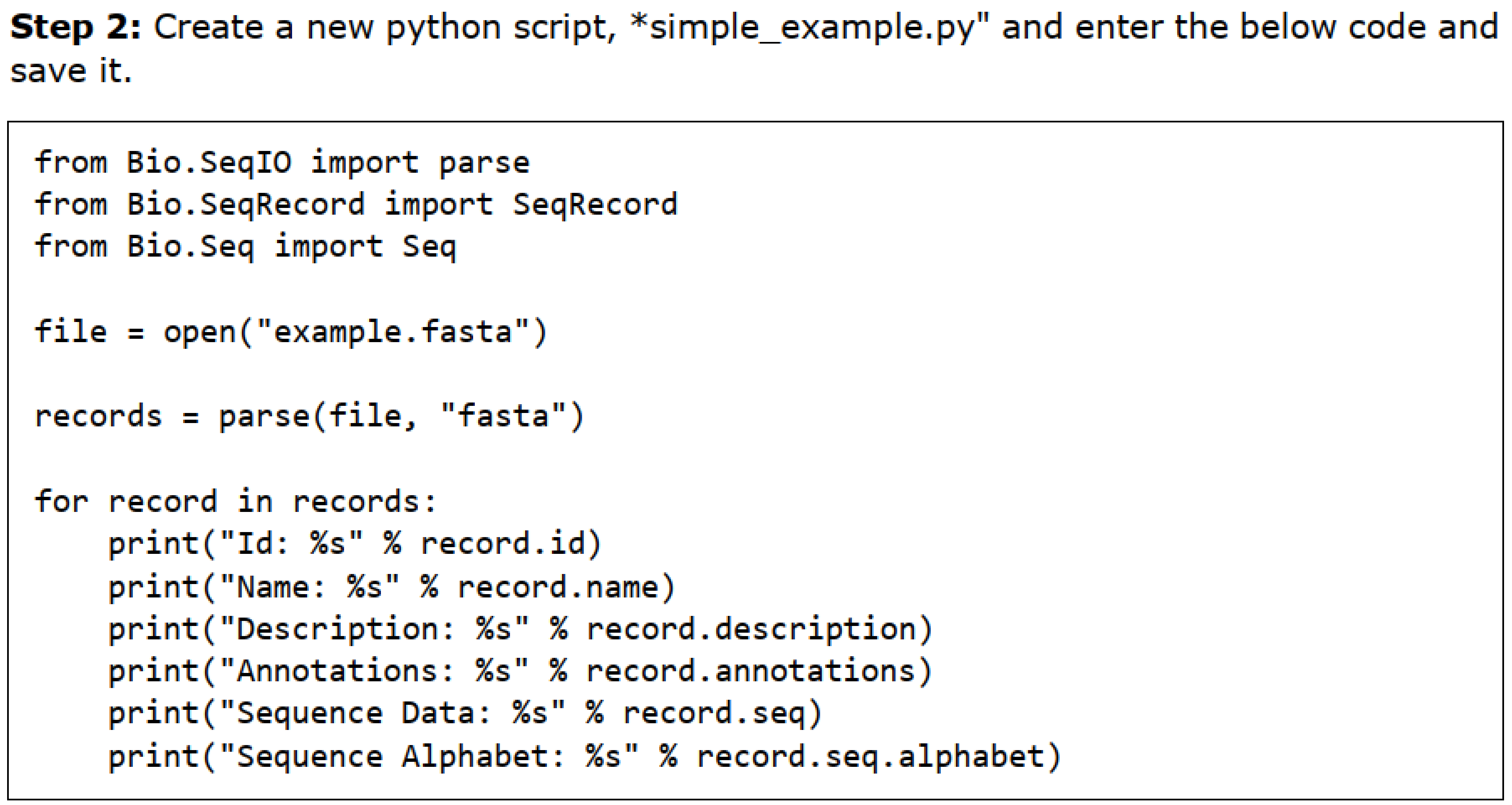
Step 1: First, create a sample sequence file, "example.fasta" and put the below content into it.

```
>sp|P25730|FMS1_ECOLI CS1 fimbrial subunit A precursor (CS1 pilin)
MKLKKTIGAMALATLFATMGASAVEKTISVTASVDPTVDLLQSDGSALPNSVALTYSPAV
NNFEAHTINTVVHTNDSDKGVVVKLSADPVLSNVLNPTLQIPVSVNFAGKPLSTTGITID
SNDLNFASSGVNKVSSTQKLSIHADATRVTGGALTAGQYQGLVSIILTKSTTTTTTTKGT

>sp|P15488|FMS3_ECOLI CS3 fimbrial subunit A precursor (CS3 pilin)
MLKIKYLLIGLSLSAMSSYSLAAAGPTLTKELALNVLSPAALDATWAPQDNLTLSNTGVS
NTLVGVLTLSNTSIDTVSIASTNVSDTSKNGTVTFAHETNNSASFATTISTDNANITLDK
NAGNTIVKTTNGSQLPTNLPLKFITTEGNEHLVSGNYRANITITSTIKGGGTKKGTTDKK
```

The extension, *fasta* refers to the file format of the sequence file. FASTA originates from the bioinformatics software, FASTA and hence it gets its name. FASTA format has multiple sequence arranged one by one and each sequence will have its own id, name, description and the actual sequence data.

Creating Simple Application



Step 2: Create a new python script, *simple_example.py" and enter the below code and save it.

```
from Bio.SeqIO import parse
from Bio.SeqRecord import SeqRecord
from Bio.Seq import Seq

file = open("example.fasta")

records = parse(file, "fasta")

for record in records:
    print("Id: %s" % record.id)
    print("Name: %s" % record.name)
    print("Description: %s" % record.description)
    print("Annotations: %s" % record.annotations)
    print("Sequence Data: %s" % record.seq)
    print("Sequence Alphabet: %s" % record.seq.alphabet)
```

Step 3: Open a command prompt and go to the folder containing sequence file, "example.fasta" and run the below command:

```
> python simple_example.py
```

Step 4: Python runs the script and prints all the sequence data available in the sample file, "example.fasta". The output will be similar to the following content.

```
Id: sp|P25730|FMS1_ECOLI
Name: sp|P25730|FMS1_ECOLI
Decription: sp|P25730|FMS1_ECOLI CS1 fimbrial subunit A precursor (CS1 pilin)
Annotations: {}
Sequence Data:
MKLKKTIGAMALATLFATMGASAVEKTISVTASVDPTVDLLQSDGSALPNSVALTYSPAVNNFEAHTINTVVHTNDSD
KGVVVKLSADPVLSNVLNPTLQIPVSVNFAGKPLSTTGITIDSNDLNFASSGVNKVSSTQKLSIHADATRVTGGALTA
GQYQGLVSIILTKSTTTTTTTKGT
Sequence Alphabet: SingleLetterAlphabet()
Id: sp|P15488|FMS3_ECOLI
Name: sp|P15488|FMS3_ECOLI
Decription: sp|P15488|FMS3_ECOLI CS3 fimbrial subunit A precursor (CS3 pilin)
Annotations: {}
Sequence Data:
MLKIKYLLIGLSLSAMSSYSLAAAGPTLTKELALNVLSPAALDATWAPQDNLTLSNTGVSNTLVGVLTLSNTSIDTVS
IASTNVSDTSKNGTVTFAHETNNSASFATTISTDNANITLDKNAGNTIVKTTNGSQLPTNLPLKFITTEGNEHLVSGN
YRANITITSTIKGGGTKKGTTDKK
Sequence Alphabet: SingleLetterAlphabet()
```

Sequence

A sequence is series of letters used to represent an organism's protein, DNA or RNA. It is represented by Seq class. Seq class is defined in Bio.Seq module.

Let's create a simple sequence in Biopython as shown below:

```
>>> from Bio.Seq import Seq
>>> seq = Seq("AGCT")
>>> seq
Seq('AGCT')
>>> print(seq)
AGCT
```

Alphabet Module

Seq objects contain Alphabet attribute to specify sequence type, letters and possible operations. It is defined in Bio.Alphabet module. Alphabet can be defined as below:

```
>>> from Bio.Seq import Seq
>>> myseq = Seq("AGCT")
>>> myseq
Seq('AGCT')
>>> myseq.alphabet
Alphabet()
```

Alphabet module provides below classes to represent different types of sequences. Alphabet - base class for all types of alphabets.

SingleLetterAlphabet - Generic alphabet with letters of size one. It derives from Alphabet and all other alphabets type derives from it.

Alphabet Module

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import single_letter_alphabet
>>> test_seq = Seq('AGTACACTGGT', single_letter_alphabet)
>>> test_seq
Seq('AGTACACTGGT', SingleLetterAlphabet())
```

ProteinAlphabet - Generic single letter protein alphabet.

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import generic_protein
>>> test_seq = Seq('AGTACACTGGT', generic_protein)
>>> test_seq
Seq('AGTACACTGGT', ProteinAlphabet())
```

NucleotideAlphabet - Generic single letter nucleotide alphabet.

Alphabet Module

Also, Biopython exposes all the bioinformatics related configuration data through Bio.Data module. For example, IUPACData.protein_letters has the possible letters of IUPACProtein alphabet.

```
>>> from Bio.Data import IUPACData
>>> IUPACData.protein_letters
'ACDEFGHIKLMNPQRSTVWY'
```

Basic Operations

To get the first value in sequence.

```
>>> seq_string = Seq("AGCTAGCT")
>>> seq_string[0]
'A'
```

To print the first two values.

```
>>> seq_string[0:2]
Seq('AG')
```

To print all the values.

```
>>> seq_string[ : ]
Seq('AGCTAGCT')
```

Basic Operations

To perform length and count operations.

```
>>> len(seq_string)
8
>>> seq_string.count('A')
2
```

To add two sequences.

```
>>> from Bio.Alphabet import generic_dna, generic_protein
>>> seq1 = Seq("AGCT", generic_dna)
>>> seq2 = Seq("TCGA", generic_dna)
>>> seq1+seq2
Seq('AGCTTCGA', DNAAlphabet())
```

Basic Operations

Here, the above two sequence objects, `seq1`, `seq2` are generic DNA sequences and so you can add them and produce new sequence. You can’t add sequences with incompatible alphabets, such as a protein sequence and a DNA sequence as specified below:

```
>>> dna_seq = Seq('AGTACACTGGT', generic_dna)
>>> protein_seq = Seq('AGUACACUGGU', generic_protein)
>>> dna_seq + protein_seq
.....
.....
TypeError: Incompatible alphabets DNAAlphabet() and ProteinAlphabet()
>>>
```

Basic Operations

To add two or more sequences, first store it in a python list, then retrieve it using 'for loop' and finally add it together as shown below:

```
>>> from Bio.Alphabet import generic_dna
>>> list =
[Seq("AGCT",generic_dna),Seq("TCGA",generic_dna),Seq("AAA",generic_dna)]
>>> for s in list:
...     print(s)
...
AGCT
TCGA
AAA
>>> final_seq = Seq(" ",generic_dna)
>>> for s in list:
...     final_seq = final_seq + s
...
>>> final_seq
Seq('AGCTTCGAAAA', DNAAlphabet())
```

Basic Operations

To change the case of sequence.

```
>>> from Bio.Alphabet import generic_rna
>>> rna = Seq("agct", generic_rna)
>>> rna.upper()
Seq('AGCT', RNAAlphabet())
```

To check python membership and identity operator.

```
>>> rna = Seq("agct", generic_rna)
>>> 'a' in rna
True
>>> 'A' in rna
False
>>> rna1 = Seq("AGCT", generic_dna)
>>> rna is rna1
False
```

Basic Operations

To find single letter or sequence of letter inside the given sequence.

```
>>> protein_seq = Seq('AGUACACUGGU', generic_protein)
>>> protein_seq.find('G')
1
>>> protein_seq.find('GG')
8
```

To perform splitting operation.

```
>>> protein_seq = Seq('AGUACACUGGU', generic_protein)
>>> protein_seq.split('A')
[Seq('', ProteinAlphabet()), Seq('GU', ProteinAlphabet()), Seq('C',
ProteinAlphabet()), Seq('CUGGU', ProteinAlphabet())]
```

To perform strip operations in the sequence.

```
>>> strip_seq = Seq("    AGCT    ")
>>> strip_seq
Seq('    AGCT    ')
>>> strip_seq.strip()
Seq('AGCT')
```