# Scientific Programming Practical 11

## Introduction

Luca Bianco - Academic Year 2020-21
luca.bianco@fmach.it

# Part A...

**Mock midterm:**

Monday, November 2nd at 14,30- 16,30 on zoom

**Midterm:**

Friday, November 6th at 11,30- 13,30 on zoom

**THE DETAILS FOR THE ZOOM CALLS WILL BE SENT BY EMAIL AND WILL BE IN THE MOODLE WEB SITE**

**PART B (theory) starts on Tuesday, November 3rd**
**PART B (lab) starts on Monday, November 9th (green time on November 5th)**

# Exercise 5

Load the contigs present in the filtered_contigs.fasta file and translate each DNA sequence into the corresponding protein. Count the number of stop codons (i.e. *) for each sequence and print them to the user (e.g. MDC020656.85 51). Finally, write the translated proteins in another .fasta file (e.g. filtered_contigs_translated.fasta).

```
MDC001115.177    65
MDC013284.379    93
MDC018185.243    418
MDC018185.241    467
MDC004527.213    48
MDC012176.157    30
MDC001204.810    155
MDC004389.256    43
MDC018297.229    317
MDC001802.364    45
  ...
MDC005174.220    132
MDC040033.7      142
MDC019674.147    89
MDC010450.877    40
MDC007097.457    89
MDC016278.70     69

44 sequences written to output file
```

```python
from Bio import SeqIO

seqIterator = SeqIO.parse("file_samples/filtered_contigs.fasta", "fasta")

sRcds = []
for s in seqIterator:
    #to avoid problems if not divisible by 3!
    seq1 = s.seq.tomutable()
    if len(seq1) % 3 != 0:
        while len(seq1) % 3 != 0:
            seq1 +="N"
    transl = seq1.toseq().translate()
    stop_cnt = transl.count("*")
    print(s.id,"\t", stop_cnt)
    s.seq = transl
    sRcds.append(s)

N = SeqIO.write(sRcds,"file_samples/filtered_contigs_translated.fasta", "fasta")
print("")
print("{} sequences written to output file".format(N))
```

mutable seq does not have the method translate

```
biancol@bludell:~/work/courses/QCBsciprolab2020$ cat file_samples/filtered_contigs_translated.fasta | head
>MDC001115.177
*MVKISQKIFSTHDICI*TNIQYCSLKP*MT*RENYEIHVGFLVTSMHIQFKIYIHKQRX
HVIK*YQSYSQANPLQDV*VYI*CIKTF**RTKXRFSLIPLDLDFRPRMDHLQALCSLNS
LSLXSLLXSSTLLEXVLLSSPLVSKVEDX*RXTPTSXVRWMEE*PKXEEMIS*NSSWWPA
LCVLEREICFLLLXNQHKXXXELFTXKFLL*HKETSQQLDFLSPSLWXALLCFVWALGFX
YFKSSYA*IKXXWVXAQWAQLNXNVSLSPXRSYIAFMISLDFLINHNT*LIQLIISIXQ*
LLTTRVYXCTIILGSN*QGSEVIGTNPIDYI*SNHLVN*NLLLIHLLL*RLHLII*KNSQ
AMSDI*PYIMATQANVEVVRRTYSVGITM*FDPSLNNTLNHTIRVWIYYVKPLM*LFLLI
*FN*VV*ERFPLCQTPNVIIPSYMIQLSRIGTLEGKFMRFLEWDFIMTIMHIQIKFNIRK
QRKHL*HIIKAIVMQIPFKVHRFXXXXXXSXTKRRLVLYLLS*T*DQGWTTSKPFAP*NP
```

# Exercise 4

4. Given a multiple sequence alignment stored in "phylip" format, write three methods: *readAlignment* that reads the input file and prints the number of sequences present, *printAlignments(alignments)* that prints the alignments to the screen and *computeConsensus(alignments,minFrequency)* that creates the consensus of all the alignments. MinFrequency is the minimum frequency (that has to be > 0.5) to keep a base in the consensus. "?" is considered as the consensus if frequency < minFrequency for all possible bases.

Ex. if alignments are:

```
ATC-G
AAC-G
AAG-G
ATCGT
```

computeConsensus(alignments,0.6) is:

```
A?C-G
```

Test the script with the file alpha-globin.phy.

| | |
|---|---|
| j=0 | [("A",4)] |
| j=1 | [("T",2),("A",2)] |
| j=2 | [("-",3),("G",1)] |
| j=3 | [("G",3),("T",1)] |

```
4 alignments present in "file_samples/alpha-globin.phy"

ENA|BAA205
ATGAGTCTCTCTGATAAGGACAAGGCTGCTGTGAAAGCCCTATGGGCTAAGATCAGCCCCAAAGCCGATGATATCGGCGCTGAAGCTCTCGGCAGAATGCTGACCGTCTACCCT
CAGACCAAGACCTACTTCGCTCACTGGGATGACCTGAGCCCTGGGTCCGGTCCTGTGAAGAAGCATGGCAAGGTTATCATGGGTGCAGTGGCCGATGCCGTTTCAAAAATAGAC
GACCTTGTGGGAGGTCTGGCCTCCCTGAGCGAACTTCATGCTTCCAAGCTGCGTGTTGACCCGGCCAACTTCAAGATCCTCGCACACAATGTCATCGTGGTCATCGGCATGCTC
TTCCCTGGAGACTTCCCCCCAGAGGTTCACATGTCAGTTGACAAGTTTTTCCAGAACTTGGCTCTGGCTCTCTCTGAGAAGTACCGCTAA
ENA|CAA284
ATGTCTCTGACCAGGACTGAGAGGACCATCATCCTGTCCCTGTGGAGCAAGATCTCCACACAGGCAGACGTCATTGGCACCGAGACCCTGGAGAGGCTCTTCTCCTGCTACCCG
CAGGCCAAGACCTACTTCCCGCACTTCGAC---CTGCACTCGGGCTCCGCGCAGCTGCGCGCGCACGGCTCCAAGGTGGTGGCCGCCGTGGGCGACGCGGTCAAGAGCATCGAC
AACGTGACGAGCGCGCTGTCCAAGCTGAGCGAGCTGCACGCCTACGTGCTGCGCGTGGACCCGGTCAACTTCAAGTTCCTGTCCCACTGCCTGCTGGTCACGTTGGCCTCGCAC
TTCCCCGCCGACTTCACGGCCGACGCGCACGCCGCCTGGGACAAGTTCCTGTCCATCGTGTCCGGCGTCCTGACGGAGAAGTACCGCTGA
ENA|CAA237
ATGGTGCTGTCTCCTGCCGACAAGACCAACGTCAAGGCCGCCTGGGGTAAGGTCGGCGCGCACGCTGGCGAGTATGGTGCGGAGGCCCTGGAGAGGATGTTCCTGTCCTTCCCC
ACCACCAAGACCTACTTCCCGCACTTCGAC---CTGAGCCACGGCTCTGCCCAAGTTAAGGGCCACGGCAAGAAGGTGGCCGACGCGCTGACCAACGCCGTGGCGCACGTGGAC
GACATGCCCAACGCGCTGTCCGCCCTGAGCGACCTGCACGCGCACAAGCTTCGGGTGGACCCGGTCAACTTCAAGCTCCTAAGCCACTGCCTGCTGGTGACCCTGGCCGCCCAC
CTCCCCGCCGAGTTCACCCCTGCGGTGCACGCTTCCCTGGACAAGTTCCTGGCTTCTGTGAGCACCGTGCTGACCTCCAAATACCGTTAA
ENA|CAA240
ATGGTGCTCTCTGGGGAAGACAAAAGCAACATCAAGGCTGCCTGGGGGAAGATTGGTGGCCATGGTGCTGAATATGGAGCTGAAGCCCTGGAAAGGATGTTTGCTAGCTTCCCC
ACCACCAAGACCTACTTTCCTCACTTTGAT---GTAAGCCACGGCTCTGCCCAGGTCAAGGGTCACGGCAAGAAGGTCGCCGATGCGCTGGCCAGTGCTGCAGGCCACCTCGAT
GACCTGCCCGGTGCCTTGTCTGCTCTGAGCGACCTGCATGCCCACAAGCTGCGTGTGGATCCCGTCAACTTCAAGCTCCTGAGCCACTGCCTGCTGGTGACCTTGGCTAGCCAC
CACCCTGCCGATTTCACCCCCGCGGTACATGCCTCTCTGGACAAATTCCTTGCCTCTGTGAGCACCGTGCTGACCTCCAAGTACCGTTAA

Consensus:
ATG???CT?TCT??????GACAAGACCA?C?TCAAGGCC???TGGGG?AAGATC?GC?C?CA?GC?G??GA???TGG?GC?GA?GCCCTGGA?AGGATGTT??C???CT?CC
C????ACCAAGACCTACTTCCC?CACTT?GA?---CTGAGCC??GGCTC?GC?CA?GT?AAGG??CACGGCAAGAAGGT?G??G??GC??TGGCC?A?GC?GT?????AC?T?G
ACGAC?TG?C??G?GC?CTGTCC?C?CTGAGCGA?CTGCA?GC??ACAAGCTGCG?GTGGACCCGGTCAACTTCAAG?TCCT???CCACTGCCTGCTGGTGACC?TGGCC???C
AC?TCCC?GCCGA?TTCACCCC?G?GGT?CACGC?TC??TGGACAAGTTCCT??C????GTG??C??CGT?CTGAC????AAGTACCG?TAA
```

```python
from Bio import AlignIO
#needed for computeConsensusV3
from collections import Counter

def readAlignment(fn):
    alignments = AlignIO.read(fn, "phylip")
    print("{} alignments present in \"{}\"\n".format(len(alignments),fn))
    return alignments

def printAlignments(aldata):
    for align in aldata:
        print("{}\n{}".format(align.id, align.seq))

def computeConsensus(aldata, minFreq = 0.51):
    consensus = ""
    if minFreq < 0.51:
        print("Warning: minFreq ({}) not valid!".format(minFreq))
        return None
    for j in range(len(aldata[0])):
        chrSeq = aldata[:,j]
        singleChars = [(chrSeq[x],chrSeq.count(chrSeq[x]))
                        for x in range(len(chrSeq))
                                if chrSeq[x] not in chrSeq[x+1:] ]
        print(singleChars)
        if len(singleChars) == 1:
            consensus +=chrSeq[0][0]
        else:
            cons = [x[0] for x in singleChars if x[1] > minFreq*len(chrSeq)]
            print(cons)

            if len(cons) == 1:
                consensus +=cons[0]
            else:
                consensus += "?"
    return consensus
```

```python
file = "file_samples/alpha-globin.phy"
als = readAlignment(file)
printAlignments(als)
out = computeConsensus(als,minFreq = 0.7)
print("\nConsensus:\n{}".format(out))
```

# Exercise 4

4. Given a multiple sequence alignment stored in "phylip" format, write three methods: *readAlignment* that reads the input file and prints the number of sequences present, *printAlignments(alignments)* that prints the alignments to the screen and *computeConsensus(alignments,minFrequency)* that creates the consensus of all the alignments. MinFrequency is the minimum frequency (that has to be > 0.5) to keep a base in the consensus. "?" is considered as the consensus if frequency < minFrequency for all possible bases.

Ex. if alignments are:

```
ATC-G
AAC-G
AAG-G
ATCGT
```

computeConsensus(alignments,0.6) is:

```
A?C-G
```

Test the script with the file alpha-globin.phy.

```
4 alignments present in "file_samples/alpha-globin.phy"

ENA|BAA205
ATGAGTCTCTCTGATAAGGACAAGGCTGCTGTGAAAGCCCTATGGGCTAAGATCAGCCCCAAAGCCGATGATATCGGCGCTGAAGCTCTCGGCAGAATGCTGACCGTCTACCCT
CAGACCAAGACCTACTTCGCTCACTGGGATGACCTGAGCCCTGGGTCCGGTCCTGTGAAGAAGCATGGCAAGGTTATCATGGGTGCAGTGGCCGATGCCGTTTCAAAAATAGAC
GACCTTGTGGGAGGTCTGGCCTCCCTGAGCGAACTTCATGCTTCCAAGCTGCGTGTTGACCCGGCCAACTTCAAGATCCTCGCACACAATGTCATCGTGGTCATCGGCATGCTC
TTCCCTGGAGACTTCCCCCCAGAGGTTCACATGTCAGTTGACAAGTTTTTCCAGAACTTGGCTCTGGCTCTCTCTGAGAAGTACCGCTAA
ENA|CAA284
ATGTCTCTGACCAGGACTGAGAGGACCATCATCCTGTCCCTGTGGAGCAAGATCTCCACACAGGCAGACGTCATTGGCACCGAGACCCTGGAGAGGCTCTTCTCCTGCTACCCG
CAGGCCAAGACCTACTTCCCGCACTTCGAC---CTGCACTCGGGCTCCGCGCAGCTGCGCGCGCACGGCTCCAAGGTGGTGGCCGCCGTGGGCGACGCGGTCAAGAGCATCGAC
AACGTGACGAGCGCGCTGTCCAAGCTGAGCGAGCTGCACGCCTACGTGCTGCGCGTGGACCCGGTCAACTTCAAGTTCCTGTCCCACTGCCTGCTGGTCACGTTGGCCTCGCAC
TTCCCCGCCGACTTCACGGCCGACGCGCACGCCGCCTGGGACAAGTTCCTGTCCATCGTGTCCGGCGTCCTGACGGAGAAGTACCGCTGA
ENA|CAA237
ATGGTGCTGTCTCCTGCCGACAAGACCAACGTCAAGGCCGCCTGGGGTAAGGTCGGCGCGCACGCTGGCGAGTATGGTGCGGAGGCCCTGGAGAGGATGTTCCTGTCCTTCCCC
ACCACCAAGACCTACTTCCCGCACTTCGAC---CTGAGCCACGGCTCTGCCCAAGTTAAGGGCCACGGCAAGAAGGTGGCCGACGCGCTGACCAACGCCGTGGCGCACGTGGAC
GACATGCCCAACGCGCTGTCCGCCCTGAGCGACCTGCACGCGCACAAGCTTCGGGTGGACCCGGTCAACTTCAAGCTCCTAAGCCACTGCCTGCTGGTGACCCTGGCCGCCCAC
CTCCCCGCCGAGTTCACCCCTGCGGTGCACGCTTCCCTGGACAAGTTCCTGGCTTCTGTGAGCACCGTGCTGACCTCCAAATACCGTTAA
ENA|CAA240
ATGGTGCTCTCTGGGGAAGACAAAAGCAACATCAAGGCTGCCTGGGGGAAGATTGGTGGCCATGGTGCTGAATATGGAGCTGAAGCCCTGGAAAGGATGTTTGCTAGCTTCCCC
ACCACCAAGACCTACTTTCCTCACTTTGAT---GTAAGCCACGGCTCTGCCCAGGTCAAGGGTCACGGCAAGAAGGTCGCCGATGCGCTGGCCAGTGCTGCAGGCCACCTCGAT
GACCTGCCCGGTGCCTTGTCTGCTCTGAGCGACCTGCATGCCCACAAGCTGCGTGTGGATCCCGTCAACTTCAAGCTCCTGAGCCACTGCCTGCTGGTGACCTTGGCTAGCCAC
CACCCTGCCGATTTCACCCCCGCGGTACATGCCTCTCTGGACAAATTCCTTGCCTCTGTGAGCACCGTGCTGACCTCCAAGTACCGTTAA

Consensus:
ATG???CT?TCT??????GACAAGACCA?C?TCAAGGCC???TGGGG?AAGATC?GC?C?CA?GC?G??GA???TGG?GC?GA?GCCCTGGA?AGGATGTT??C???CT?CC
C????ACCAAGACCTACTTCCC?CACTT?GA?---CTGAGCC??GGCTC?GC?CA?GT?AAGG??CACGGCAAGAAGGT?G??G??GC??TGGCC?A?GC?GT?????AC?T?G
ACGAC?TG?C??G?GC?CTGTCC?C?CTGAGCGA?CTGCA?GC??ACAAGCTGCG?GTGGACCCGGTCAACTTCAAG?TCCT???CCACTGCCTGCTGGTGACC?TGGCC???C
AC?TCCC?GCCGA?TTCACCCC?G?GGT?CACGC?TC??TGGACAAGTTCCT??C????GTG??C??CGT?CTGAC????AAGTACCG?TAA
```

```python
def computeConsensusV2(aldata, minFreq = 0.51):
    #this solution uses sets.
    #set(list("ATTTAC")) returns {'A', 'T', 'C'}
    consensus = ""
    if minFreq < 0.51:
        print("Warning: minFreq ({}) not valid!".format(minFreq))
        return None
    for j in range(len(aldata[0])):
        chrSeq = aldata[:,j]
        singleChars = set(list(chrSeq))
        if len(singleChars) == 1:
            consensus +=chrSeq[0]
        else:
            cons = [x for x in singleChars if chrSeq.count(x) > minFreq*len(chrSeq)]
            if len(cons) == 1:
                consensus +=cons[0]
            else:
                consensus += "?"
    return consensus
```

```python
file = "file_samples/alpha-globin.phy"
als = readAlignment(file)
printAlignments(als)
out = computeConsensusV2(als,minFreq = 0.7)
print("Consensus:\n{}".format(out))
```

# Exercise 4

4. Given a multiple sequence alignment stored in "phylip" format, write three methods: *readAlignment* that reads the input file and prints the number of sequences present, *printAlignments(alignments)* that prints the alignments to the screen and *computeConsensus(alignments,minFrequency)* that creates the consensus of all the alignments. MinFrequency is the minimum frequency (that has to be > 0.5) to keep a base in the consensus. "?" is considered as the consensus if frequency < minFrequency for all possible bases.

Ex. if alignments are:

```
ATC-G
AAC-G
AAG-G
ATCGT
```

computeConsensus(alignments,0.6) is:

```
A?C-G
```

Test the script with the file alpha-globin.phy.

| | |
|---|---|
| j=0 | {“A”: 4)} |
| j=1 | {“T”:2,”A”:,2} |
| j=2 | {“-”:3, “G”:1} |
| j=3 | {“G”:3,”T”:1} |

```
4 alignments present in "file_samples/alpha-globin.phy"

ENA|BAA205
ATGAGTCTCTCTGATAAGGACAAGGCTGCTGTGAAAGCCCTATGGGCTAAGATCAGCCCCAAAGCCGATGATATCGGCGCTGAAGCTCTCGGCAGAATGCTGACCGTCTACCCT
CAGACCAAGACCTACTTCGCTCACTGGGATGACCTGAGCCCTGGGTCCGGTCCTGTGAAGAAGCATGGCAAGGTTATCATGGGTGCAGTGGCCGATGCCGTTTCAAAAATAGAC
GACCTTGTGGGAGGTCTGGCCTCCCTGAGCGAACTTCATGCTTCCAAGCTGCGTGTTGACCCGGCCAACTTCAAGATCCTCGCACACAATGTCATCGTGGTCATCGGCATGCTC
TTCCCTGGAGACTTCCCCCCAGAGGTTCACATGTCAGTTGACAAGTTTTTCCAGAACTTGGCTCTGGCTCTCTCTGAGAAGTACCGCTAA
ENA|CAA284
ATGTCTCTGACCAGGACTGAGAGGACCATCATCCTGTCCCTGTGGAGCAAGATCTCCACACAGGCAGACGTCATTGGCACCGAGACCCTGGAGAGGCTCTTCTCCTGCTACCCG
CAGGCCAAGACCTACTTCCCGCACTTCGAC---CTGCACTCGGGCTCCGCGCAGCTGCGCGCGCACGGCTCCAAGGTGGTGGCCGCCGTGGGCGACGCGGTCAAGAGCATCGAC
AACGTGACGAGCGCGCTGTCCAAGCTGAGCGAGCTGCACGCCTACGTGCTGCGCGTGGACCCGGTCAACTTCAAGTTCCTGTCCCACTGCCTGCTGGTCACGTTGGCCTCGCAC
TTCCCCGCCGACTTCACGGCCGACGCGCACGCCGCCTGGGACAAGTTCCTGTCCATCGTGTCCGGCGTCCTGACGGAGAAGTACCGCTGA
ENA|CAA237
ATGGTGCTGTCTCCTGCCGACAAGACCAACGTCAAGGCCGCCTGGGGTAAGGTCGGCGCGCACGCTGGCGAGTATGGTGCGGAGGCCCTGGAGAGGATGTTCCTGTCCTTCCCC
ACCACCAAGACCTACTTCCCGCACTTCGAC---CTGAGCCACGGCTCTGCCCAAGTTAAGGGCCACGGCAAGAAGGTGGCCGACGCGCTGACCAACGCCGTGGCGCACGTGGAC
GACATGCCCAACGCGCTGTCCGCCCTGAGCGACCTGCACGCGCACAAGCTTCGGGTGGACCCGGTCAACTTCAAGCTCCTAAGCCACTGCCTGCTGGTGACCCTGGCCGCCCAC
CTCCCCGCCGAGTTCACCCCTGCGGTGCACGCTTCCCTGGACAAGTTCCTGGCTTCTGTGAGCACCGTGCTGACCTCCAAATACCGTTAA
ENA|CAA240
ATGGTGCTCTCTGGGGAAGACAAAAGCAACATCAAGGCTGCCTGGGGGAAGATTGGTGGCCATGGTGCTGAATATGGAGCTGAAGCCCTGGAAAGGATGTTTGCTAGCTTCCCC
ACCACCAAGACCTACTTTCCTCACTTTGAT---GTAAGCCACGGCTCTGCCCAGGTCAAGGGTCACGGCAAGAAGGTCGCCGATGCGCTGGCCAGTGCTGCAGGCCACCTCGAT
GACCTGCCCGGTGCCTTGTCTGCTCTGAGCGACCTGCATGCCCACAAGCTGCGTGTGGATCCCGTCAACTTCAAGCTCCTGAGCCACTGCCTGCTGGTGACCTTGGCTAGCCAC
CACCCTGCCGATTTCACCCCCGCGGTACATGCCTCTCTGGACAAATTCCTTGCCTCTGTGAGCACCGTGCTGACCTCCAAGTACCGTTAA

Consensus:
ATG???CT?TCT??????GACAAGACCA?C?TCAAGGCC???TGGGG?AAGATC?GC?C?CA?GC?G??GA???TGG?GC?GA?GCCCTGGA?AGGATGTT??C???CT?CC
C????ACCAAGACCTACTTCCC?CACTT?GA?---CTGAGCC??GGCTC?GC?CA?GT?AAGG??CACGGCAAGAAGGT?G??G??GC??TGGCC?A?GC?GT?????AC?T?G
ACGAC?TG?C??G?GC?CTGTCC?C?CTGAGCGA?CTGCA?GC??ACAAGCTGCG?GTGGACCCGGTCAACTTCAAG?TCCT???CCACTGCCTGCTGGTGACC?TGGCC???C
AC?TCCC?GCCGA?TTCACCCC?G?GGT?CACGC?TC??TGGACAAGTTCCT??C????GTG??C??CGT?CTGAC????AAGTACCG?TAA
```

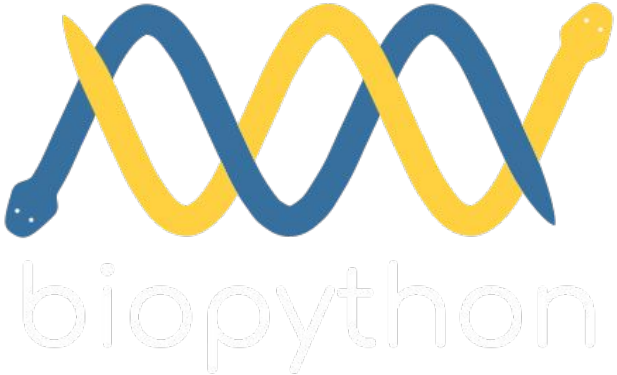
```python
from collections import Counter

def computeConsensusV3(aldata, minFreq = 0.51):
    #this solution uses collections.Counter
    #collections.Counter("ATTTAC") returns the dictionary: {'T': 3, 'A': 2, 'C': 1}
    #
    consensus = ""
    if minFreq < 0.51:
        print("Warning: minFreq ({}) not valid!".format(minFreq))
        return None
    for j in range(len(aldata[0])):
        chrSeq = aldata[:,j]
        singleChars = Counter(chrSeq)
        if len(singleChars) == 1:
            consensus +=chrSeq[0]
        else:
            cons = [x for x in singleChars if singleChars[x] > minFreq*len(chrSeq)]
            if len(cons) == 1:
                consensus +=cons[0]
            else:
                consensus += "?"
    return consensus


file = "file_samples/alpha-globin.phy"
als = readAlignment(file)
printAlignments(als)
out = computeConsensusV2(als,minFreq = 0.7)
print("\nConsensus:\n{}".format(out))
out1 = computeConsensusV2(als,minFreq = 0.7)
#print("Consensus:\n{}".format(out1))
out2 = computeConsensusV3(als,minFreq = 0.7)
#print("Consensus:\n{}".format(out2))
assert(out1 == out2)
assert(out == out1)
```

# New things seen...

```python
#Reminder of the new things seen...

from collections import Counter

my_seq = "ATTAGATCACATAAAA"
print("Sequence: {}".format(my_seq))

#The set data structure
S = set(my_seq)
print("The set: {}".format(S))

#The counter object
counts = Counter(my_seq)
print("The counts:{}".format(counts))
print("")
minFreq = 0.51
print([x for x in counts if counts[x] > minFreq*len(my_seq)])

#Asserts to make sure some conditions hold...
assert(10 > 5)
assert(my_seq[0] == 'A')
assert(len(my_seq) == sum(list(counts.values())))
assert(len(my_seq) > sum(list(counts.values())))
```

```
Sequence: ATTAGATCACATAAAA
The set: {'A', 'T', 'G', 'C'}
The counts:Counter({'A': 9, 'T': 4, 'C': 2, 'G': 1})

['A']

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-25-2e6861380230> in <module>
     21 assert(my_seq[0] == 'A')
     22 assert(len(my_seq) == sum(list(counts.values())))
---> 23 assert(len(my_seq) > sum(list(counts.values())))

AssertionError:
```

# Biopython




**FROM Biopython's website:**

The Biopython Project is an international association of developers of freely available **Python tools for computational molecular biology**.

The goal of Biopython is to make it as easy as possible to use **Python for bioinformatics** by creating high-quality, reusable modules and classes.

# BLAST

Blast (Basic logical alignment search tool) is a well known tool to find similarities between biological sequences. It **compares DNA or protein sequences and calculates the statistical significance of the matches found**.

The online version of blast can be accessed through the Biopython's `Bio.Blast.NCBIWWW.qblast()` function.

Its basic syntax is the following (first import `from Bio.Blast import NCBIWWW`):

```
result_handle = Bio.Blast.NCBIWWW.qblast(blast_program, database, query_str)
```

`blast_program` is the program to perform the alignment. The options are **blastn, blastp, blastx, tblast or tblastx**.

`database` is the database to search against

`query_str` is a string containing the query to search against the database.

The **query** can be a **sequence or a fasta file entry or an identifier like a GI number** (NCIBI's sequence identification number).

Among the others, some **optional parameters** are the output format (`format_type` that by default is "XML" which is the most stable output format but results can be stored also as text with "Text"). It is also possible to specify an expectation value cut-off to filter out alignments `expect` (the e-value threshold, default value is 10.0).

# BLAST: search DBs

Blast (Basic logical alignment search tool) is a well known tool to find similarities between biological sequences. It compares DNA or protein sequences and calculates the statistical significance of the matches found.

The online version of blast can be accessed through the Biopython's `Bio.Blast.NCBIWWW.qblast()` function.

**Table 2. Contents of the common BLAST sequence databases**

| Database | Type | Content |
|---|---|---|
| **nr (nt) default** | Nucleotide | All GenBank + EMBL + DDBJ + PDB sequences, excluding sequences from PAT, EST, STS, GSS, WGS, TSA and phase 0, 1 or 2 HTGS sequences, mostly non-redundant. |
| refseq_rna | Nucleotide | Curated (NM_, NR_) plus predicted (XM_, XR_) sequences from NCBI Reference Sequence Project. |
| refseq_genomic | Nucleotide | Genomic sequences from NCBI Reference Sequence Project. |
| refseq_representative_genomes | Nucleotide | NCBI RefSeq Reference and Representative genomes across broad taxonomy groups including eukaryotes, bacteria, archaea, viruses and viroids. These genomes are among the best quality genomes available with minimum redundancy - one genome per species for eukaryotes and diverse isolates for the same species for others. |
| chromosome | Nucleotide | Complete genomes and complete chromosomes from the NCBI Reference Sequence project. |
| Human G+T | Nucleotide | The genomic sequences plus curated and predicted RNAs from the current build of the human genome. |
| Mouse G+T | Nucleotide | The genomic sequences plus curated and predicted RNAs from the current build of the mouse genome. |
| est | Nucleotide | Database of GenBank + EMBL + DDBJ sequences from EST division |
| HTGS | Nucleotide | Unfinished High Throughput Genomic Sequences; Sequences: phases 0, 1 and 2 |
| wgs | Nucleotide | Assemblies of Whole Genome Shotgun sequences. |
| pat | Nucleotide | Nucleotides from the Patent division of GenBank. |
| pdb | Nucleotide | Nucleotide sequences from the 3-dimensional structure records from Protein Data Bank. |
| TSA | Nucleotide | Transcriptome Shotgun Assemblies, assembled from RNA-seq SRA data |
| 16S microbial | Nucleotide | 16S Microbial rRNA sequences from Targeted Loci Project |
| | | |
| **nr default** | Protein | Non-redundant GenBank CDS translations + RefSeq + PDB + SwissProt + PIR + PRF, excluding those in PAT, TSA, and env_nr. |
| refseq_protein | Protein | Protein sequences from NCBI Reference Sequence project. |
| swissprot | Protein | Last major release of the UniProtKB/SWISS-PROT protein sequence database (no incremental updates). |
| Landmark | Protein | The landmark database includes proteomes from representative genomes spanning a wide taxonomic range |
| pat | Protein | Proteins from the Patent division of GenBank. |
| pdb | Protein | Protein sequences from the 3-dimensional structure records from the Protein Data Bank. |
| env_nr | Protein | Protein sequences translated from the CDS annotation of metagenomic nucleotide sequences. |
| tsa_nr | Protein | Protein sequences translated from CDSs annotated on transcriptome shotgun assemblies. |

# BLAST: the query

Blast (Basic logical alignment search tool) is a well known tool to find similarities between biological sequences. It compares DNA or protein sequences and calculates the statistical significance of the matches found.

The online version of blast can be accessed through the Biopython's `Bio.Blast.NCBIWWW.qblast()` function.

The query string can be obtained by reading a fasta file into a string

```
from Bio.Blast import NCBIWWW
fasta_string = open("myfile.fasta").read()
result_handle = NCBIWWW.qblast("blastn", "nt", fasta_string)
```

or we can give a SeqRecord:

```
from Bio.Blast import NCBIWWW
from Bio import SeqIO
record = SeqIO.read("myfile.fasta", format="fasta")
result_handle = NCBIWWW.qblast("blastn", "nt", record.seq)
```

It is also possible to specify some **optional parameters** in `entrez_query` for example we can limit the search to specific organisms with: `entrez_query='"Malus Domestica" [Organism]'`.

**NOTE: qblast returns a result_handle not the results!**

# BLAST: parsing the output

Query results can be parsed with the methods of the **module** `Bio.Blast.NCBIXML`

Single result

```
blast_record = NCBIXML.read(result_handle)
```

or (multiple results)

```
blast_records = NCBIXML.parse(result_handle)
```

Note that to use these methods we first need to import the `NCBIXML` module with `from Bio.Blast import NCBIXML`.

# BLAST: saving the output

We can save the entries in a file

```
out_f = open("my_blast_result.xml", "w")
out_f.write(result_handle.read())
out_f.close
result_handle.close()
```

If we have more than one entry we need to loop thorugh all the entries and save them in the file:

```
out_f = open("my_blast_result.xml", "w")
for entry in result_handle.parse():
    out_f.write(entry)
out_f.close
result_handle.close()
```

# BLAST: reading the input

A BLAST output file can be read by opening the file to get the handler and then parse it with the method **parse**

```
from Bio.Blast import NCBIXML
result_handle = open("my_blast.xml")
blast_records = NCBIXML.parse(result_handle)
```

This returns an iteratort to **Bio.Blast.Record.Blast** objects that hold the results of the alignment

# The BLAST record class

The `Bio.Blast.Record.Blast` class holds the results of the alignment.

It is composed of three types of information:

**query**
**Descriptions**
**Alignments**

1. *query*: the identifier of the query (a string).
2. *Descriptions* : a list of Description objects. Each `Description` holds the following information:
   - `Description.title` : a string with the title of the hit;
   - `Description.score` : a float with the score of the alignment;
   - `Description.num_alignments` : an int with the number of alignments with the same subject;
   - `Description.e` : a float with the e-value of the alignment.

# The BLAST record class

The `Bio.Blast.Record.Blast` class holds the results of the alignment.

It is composed of three types of information:

**query**
**Descriptions**
**Alignments**

2. *Alignments* : a list of Alignment objects. Each `Alignment` holds the following information:

- `Alignment.title` : a string with the title of the hit (identical to `Description.title`);
- `Alignment.length` : an int with the length of the alignment;
- `Alignment.hsps` : a list of HSP objects (High Scoring Pair). Each `HSP` has the following info:
  - `HSP.score` : the BLAST score of the hit
  - `HSP.bits` : the bits score of the hit (x: on average 2^x pairs to find such a good hit by chance)
  - `HSP.expect` : the evalue of the hit
  - `HSP.num_alignments` : the number of alignments for the same subject
  - `HSP.identities` : the numbe of identities between query and subject
  - `HSP.positives` : the number of identical bases/aminos or having similar chemical properties
  - `HSP.gaps` : the number of gaps between query and subject
  - `HSP.strand` : a **tuple** with (query,subject) strands
  - `HSP.frame` : a **tuple** with the frame shifts
  - `HSP.query/HSP.sbjct` : query/subject sequence
  - `HSP.query_start/HSP.sbjct_start` :query/subject start point
  - `HSP.match` : the match sequence (basically "|" for matches and spaces for mismatches)
  - `HSP.align_length` : the alignment length.

# BLAST

**Example:**Let's blast the serum albumin sequence (gi number 23307792) to the human genome and report all the information. (warning might take a while to run!)

```
TITLE:gi|23307792|gb|AF542069.1| Homo sapiens serum albumin (HSA) mRNA, complete cds
SCORE:4352.0
N.ALIGN:1
E-VAL:0.0
TITLE:gi|1519245814|ref|NM_000477.7| Homo sapiens albumin (ALB), mRNA
SCORE:4305.0
N.ALIGN:1
E-VAL:0.0
TITLE:gi|28591|emb|V00495.1| H.sapiens mRNA for serum albumin
SCORE:4253.0
N.ALIGN:2
E-VAL:0.0
TITLE:gi|7770116|gb|AF119840.1|AF119840 Homo sapiens PRO0903 mRNA, complete cds
SCORE:4062.0
N.ALIGN:1
E-VAL:0.0
```

```
from Bio.Blast import NCBIWWW
from Bio.Blast import NCBIXML

result_handle = NCBIWWW.qblast("blastn", "nr", "23307792",
                               entrez_query='"Homo Sapiens" [Organism]'
                               )



for res in NCBIXML.parse(result_handle):
    for d in res.descriptions:

        print("TITLE:{}\nSCORE:{}\nN.ALIGN:{}\nE-VAL:{}".format(
            d.title,d.score, d.num_alignments,d.e))

    for a in res.alignments:
        print("Align Title:{}\nAlign Len: {}".format(a.title, a.length))



        for h in a.hsps:
            s = h.score
            b = h.bits
            e = h.expect
            n = h.num_alignments
            i = h.identities
            p = h.positives
            g = h.gaps
            st = h.strand
            f = h.frame
            q = h.query
            sb = h.sbjct
            qs = h.query_start
            ss = h.sbjct_start
            qe = h.query_end
            se = h.sbjct_end
            m = h.match
            al = h.align_length

            print("Score: {} Bits: {} E-val: {}".format(s,b,e))
            print("N.aligns:{} Ident:{} Pos.:{} Gaps:{} Align len:{}".format(
                n,i,p,g,al))
            print("Strand: {} Frame: {}".format(st,f))
            print("Query:", q, " start:", qs, " end:", qe)
            print("Match:",m)
            print("Subjc:",sb, " start:", ss, " end:", se)


result_handle.close()
```

# BLAST

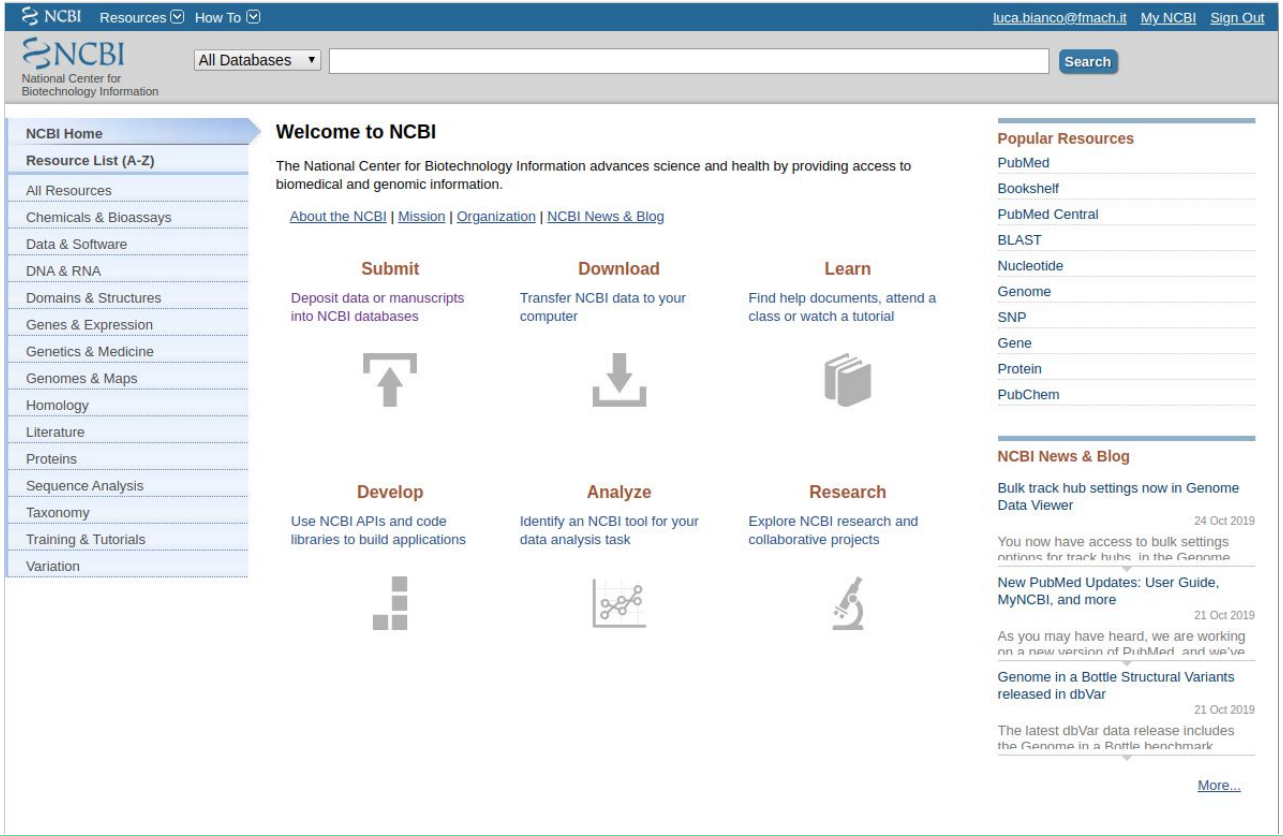
**Example:**Let's blast the serum albumin sequence (gi number 23307792) to the human genome and report all the information. (warning might take a while to run!)

```
N.aligns: None Idents: 2176 Positives: 2176 Gaps: 0 Align len: 2176
Strand: (None, None) Frame: (1, 1)
Query: TCTCTTCTGTCAACCCCACGCGCCTTTGGCACAATGAAGTGGGTAACCTTTATTTCCCTTCTTTTTCTCTTTAGCTCGG
Match: ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Subjc: TCTCTTCTGTCAACCCCACGCGCCTTTGGCACAATGAAGTGGGTAACCTTTATTTCCCTTCTTTTTCTCTTTAGCTCGG
Align Title:gi|1046552723|ref|NM_000477.6| Homo sapiens albumin (ALB), mRNA
Align Len: 2335
Score: 4304.0 Bits: 3882.14 E-val: 0.0
N.aligns: None Idents: 2168 Positives: 2168 Gaps: 1 Align len: 2177
Strand: (None, None) Frame: (1, 1)
Query: TCTCTTCTGTCAACCCCACGCGCCTTTGGCACAATGAAGTGGGTAACCTTTATTTCCCTTCTTTTTCTCTTTAGCTCGG
Match: |||||||||||||||||||| |||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Subjc: TCTCTTCTGTCAACCCCACACGCCTTTGGCACAATGAAGTGGGTAACCTTTATTTCCCTTCTTTTTCTCTTTAGCTCGG
Align Title:gi|28591|emb|V00495.1| H.sapiens mRNA for serum albumin
Align Len: 2251
Score: 4252.0 Bits: 3835.25 E-val: 0.0
N.aligns: None Idents: 2159 Positives: 2159 Gaps: 4 Align len: 2177
Strand: (None, None) Frame: (1, 1)
Query: TCTCTTCTGTCAACCCCACGCGCCTTTGGCACAATGAAGTGGGTAACCTTTATTTCCCTTCTTTTTCTCTTTAGCTCGG
Match: ||||||||||||||||||||  |||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Subjc: TCTCTTCTGTCAACCCCACGC--CTTTGGCACAATGAAGTGGGTAACCTTTATTTCCCTTCTTTTTCTCTTTAGCTCGG
```

```
from Bio.Blast import NCBIWWW
from Bio.Blast import NCBIXML

result_handle = NCBIWWW.qblast("blastn", "nt", "23307792",
                               entrez_query='"Homo Sapiens" [Organism]'
                               )



for res in NCBIXML.parse(result_handle):
    for d in res.descriptions:

        print("TITLE:{}\nSCORE:{}\nN.ALIGN:{}\nE-VAL:{}".format(
            d.title,d.score, d.num_alignments,d.e))

    for a in res.alignments:
        print("Align Title:{}\nAlign Len: {}".format(a.title, a.length))



        for h in a.hsps:
            s = h.score
            b = h.bits
            e = h.expect
            n = h.num_alignments
            i = h.identities
            p = h.positives
            g = h.gaps
            st = h.strand
            f = h.frame
            q = h.query
            sb = h.sbjct
            qs = h.query_start
            ss = h.sbjct_start
            qe = h.query_end
            se = h.sbjct_end
            m = h.match
            al = h.align_length

            print("Score: {} Bits: {} E-val: {}".format(s,b,e))
            print("N.aligns:{} Ident:{} Pos.:{} Gaps:{} Align len:{}".format(
                n,i,p,g,al))
            print("Strand: {} Frame: {}".format(st,f))
            print("Query:", q, " start:", qs, " end:", qe)
            print("Match:",m)
            print("Subjc:",sb, " start:", ss, " end:", se)


result_handle.close()
```

# Getting data from NCBI

https://www.ncbi.nlm.nih.gov/



# Getting data from NCBI

First of all we need to import the Entrez module with ( `from Bio import Entrez` ) and then we can start interacting with Entrez, then we should specify (optional) an email setting `Entrez.email`.

Biopython provides a module (`Bio.Entrez`) to **pull data** off resources like PubMed or GenBank, and other repositories programmatically through Entrez.

1. `res_handle = Entrez.einfo(db)` returns a summary of the Entez databases as a results handle. `db` is an optional parameter specifying the resource of interest;
2. `res_handle = Entrez.esearch(db, term,id)` returns all the entries in `db` having query matching the term `term`. It is also possible to specify an `id` to get the information relative to that resource id;
3. `res_handle = Entrez.efetch(db, id, rettype, retmode)` returns full record corresponding to the identifier `id` from the database `db` formatted in `rettype` (eg. gb, fasta,... complete list) and return mode `retmode` (eg. text);
4. `res_handle = Entrez.esummary(db, id)` returns the summary of the entry `id` from the database `db` as a handle;
5. `result = Entrez.read(res_handle)` reads the information on the XML handle `res_handle` and stores them in a dictionary, list or string, depending on the case.

# Getting data from NCBI

Let's get a list of all available databases in Entrez as a dictionary. Let's then get a summary of the entries in 'sra'.

```
from Bio import Entrez
import datetime


Entrez.email = "my_email"
handle = Entrez.einfo()
res = Entrez.read(handle)
#print(res)
print("")
print("As a list:")
print(res['DbList'])

res = Entrez.read(Entrez.einfo(db = "sra"))
#uncomment to see all the information captured
#print(res)
#for el in res["DbInfo"].keys():
#    print(el)
date = res["DbInfo"]["LastUpdate"]
dt = datetime.datetime.strptime(date, "%Y/%m/%d %H:%M")
print("")
print("Entries count: {:,}".format(int(res["DbInfo"]["Count"])))
print("LastUpdate: {}/{}/{} {}:{}".format(dt.day,
                                           dt.month,
                                           dt.year,
                                           dt.hour,
                                           dt.minute))
print("Description:", res["DbInfo"]["Description"])
```

As a list:
['pubmed', 'protein', 'nuccore', 'ipg', 'nucleotide', 'structure', 'sparcle', 'protfam', 'genome', 'annotinfo', 'assembly', 'bioproject', 'biosample', 'blastdbinfo', 'books', 'cdd', 'clinvar', 'gap', 'gapplus', 'grasp', 'dbvar', 'gene', 'gds', 'geoprofiles', 'homologene', 'medgen', 'mesh', 'ncbisearch', 'nlmcatalog', 'omim', 'orgtrack', 'pmc', 'popset', 'proteinclusters', 'pcassay', 'biosystems', 'pccompound', 'pcsubstance', 'seqannot', 'snp', 'sra', 'taxonomy', 'biocollections', 'gtr']

Entries count: 12,222,409
LastUpdate: 27/10/2020 4:39
Description: SRA Database

# Getting data from NCBI

**Example:** Retrieve genbank formatted information of the Malus x domestica MYB domain class transcription factor (MYB1) mRNA complete cds (nucleotide database id:HM122614.1). Parse it as a SeqRecord, printing only the sequence (remember previous practical's SeqIO).

```
ID: HM122614.1
Name: HM122614
Description: Malus x domestica MYB domain class transcription factor (MYB1) mRNA, comp
Number of features: 3
/source=Malus domestica (apple)
/data_file_division=HTC
/organism=Malus domestica
/sequence_version=1
/references=[Reference(title='Transcription Factors in Apple', ...), Reference(title='
/accessions=['HM122614']
/keywords=['HTC']
/date=15-AUG-2010
/taxonomy=['Eukaryota', 'Viridiplantae', 'Streptophyta', 'Embryophyta', 'Tracheophyta'
/topology=linear
/molecule_type=mRNA
Seq('TTTGGTCTGCTGGGTAGGTACTCATAAAAACAAACCAACCGAAGCCTCCGAACC...AAA', IUPACAmbiguousDNA(

SEQUENCE:
TTTGGTCTGCTGGGTAGGTACTCATAAAAACAAACCAACCGAAGCCTCCGAACCGACCACCAATGACGGCCCCAAACGGCGCCGTC
```

```python
from Bio import Entrez
from Bio import SeqIO

Entrez.email = "my_email"
handle = Entrez.efetch(db="nucleotide",
                       id = "HM122614.1",
                       rettype = "gb",
                       retmode="text")
my_seq = SeqIO.read(handle, format = "genbank")
print(handle.read())
print(my_seq)
print("")
print("SEQUENCE:")
print(my_seq.seq)
```

# Protein Data Bank (PDB)

https://www.rcsb.org/

PDB is a database of structural information of 3D shapes of proteins, nucleic acids, and complex assemblies. The database currently contains more than 170,000 total structures.

# Protein Data Bank (PDB)

**First of all:**

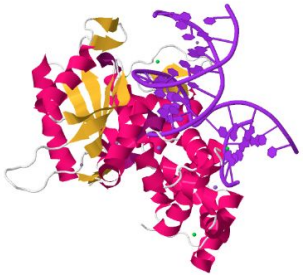
```
from Bio.PDB import *
```

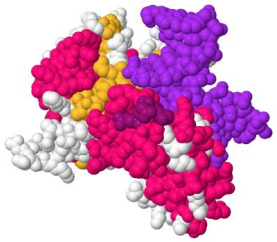
Then it is possible to download a structure directly from PDB by using a `PDBList` object that features a function called `download_pdb_files`

```
PDBList.download_pdb_files(pdb_codes, pdir, file_format)
```

that downloads the `file_format` formatted structures defined in the `pdb_codes` list of 4 symbols structure Ids from PDB, stores them in the directory `pdir`. The safer `file_format` to use is "mmCif". The function will not download the structures more than once. If a file is already present in the specified directory, a message **Structure exists** will be displayed.

Let's programmatically download two different structures of the DNA polymerase 3C2K and 3C2L





```
from Bio.PDB import *

pdbl = PDBList()
structures = ["3C2K", "3C2L"]
el = pdbl.download_pdb_files(structures,
                             file_format = "mmCif",
                             pdir = "file_samples/")
```

```
Downloading PDB structure '3C2K'...
Downloading PDB structure '3C2L'...
```

# Protein Data Bank (PDB)

Once the structures are available locally, one can start parsing them to do something useful. Parsing can be done through the `MMCIFParser` object

```
parser = MMCIFParser()
```

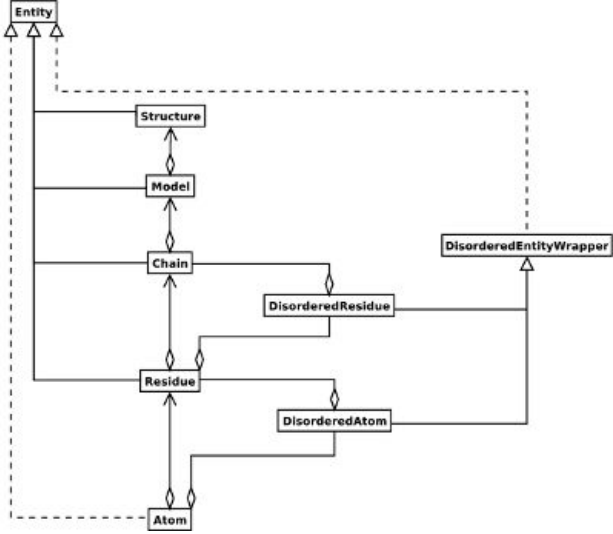
The `parser` object has several methods able to deal with structures. One of these is the `get_structure` that creates a `PDB.Structure.Structure` object with all the data present in the structure file.

The basic syntax is:

```
structure = parser.get_structure(pdb_code, filename)
```

where `pdb_code` is the PDB code of the structure contained in the file `filename`. The method returns a `PDB.Structure.Structure` that contains one or more models.

# PDB.Structure.Structure




**A `Structure` consists of a collection of one or more `Model` (different 3D conformations of the very same structure)** that is a **collection of `Chain`** that is a **collection of `Residues`** that is a **collection of `Atoms`**

Given a `Structure` we can obtain iterators to **models**, **chains**, **residues** or **atoms** with:

```
Structure.get_models()
Structure.get_chains()
Structure.get_residues()
Structure.get_atoms()
```

For each model obtained with `structure.get_models()` function we can loop through its chains, residues and atoms. For atoms we can get the 3D coordinates with `Atom.get_coord()`.

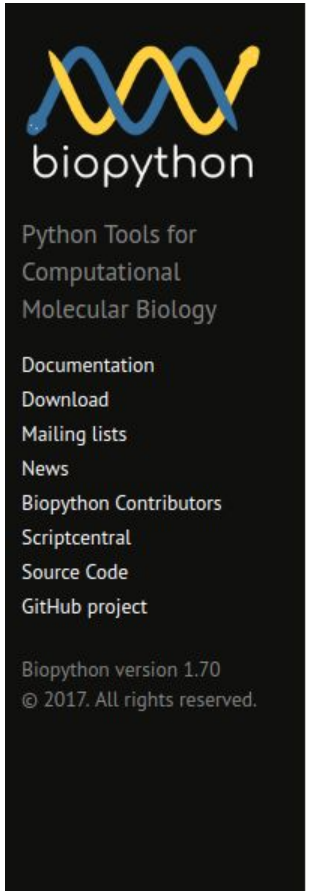
# PDB

**Example:** Let's loop through all the models, chain, residues and atoms of the DNA polymerase structure 3C2K. Print the 3D coordinates of each atom.

```
from Bio.PDB import *
parser = MMCIFParser(QUIET=True) #To disable warnings
filename = "file_samples/3c2l.cif"
structure = parser.get_structure("3c2l", filename)

for model in structure.get_models():
    print("model", model, "has {} chains".format(len(model)))

    for chain in model:
        print(" - chain ", chain, "has {} residues".format(len(chain)))

        for residue in chain:
            print ("      - residue", residue.get_resname(), "has {}
atoms".format(len(residue)))

            for atom in residue:
                x,y,z = atom.get_coord()
                print("        - atom:", atom.get_name(), "x: {} y:{} z:{}".format(x,y,z))
```

```
model <Model id=0> has 4 chains
 - chain  <Chain id=T> has 41 residues
      - residue DC has 16 atoms
        - atom: O5' x: 30.740999221801758 y:-2.2209999561309814 z:16.618999481201172
        - atom: C5' x: 31.167999267578125 y:-0.9599999785423279 z:16.062999725341797
        - atom: C4' x: 29.996000289916992 y:-0.009999999776482582 z:15.932999610900879
        - atom: O4' x: 28.96299934387207 y:-0.6069999933242798 z:15.107000350952148
        - atom: C3' x: 29.320999145507812 y:0.38499999046325684 z:17.253000259399414
      - residue DC has 19 atoms
        - atom: P x: 28.641000747680664 y:2.5 z:18.69099998474121
        - atom: OP1 x: 29.559999465942383 y:3.625 z:19.025999069213867
```

# http://biopython.org




Edit this page on GitHub

## Biopython

See also our News feed and Twitter.

### Introduction

Biopython is a set of freely available tools for biological computation written in Python 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. The source code is made available under the Biopython License, which is extremely liberal and compatible with almost every license in the world.

We are a member project of the Open Bioinformatics Foundation (OBF), who take care of our domain name and hosting for our mailing list etc. The OBF used to host our developement repository, issue tracker and website but these are now on GitHub.

This wiki will help you download and install Biopython, and start using the libraries and tools.

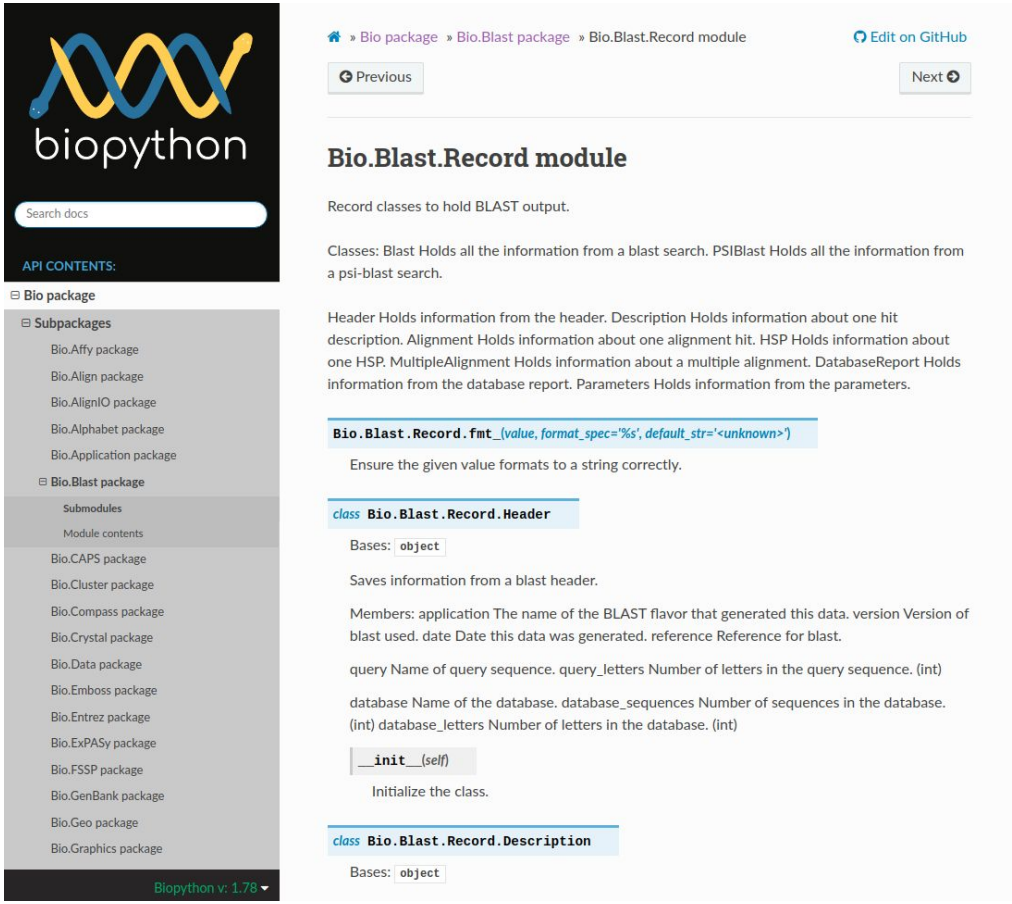
| Get Started | Get help | Contribute |
|---|---|---|
| Download Biopython | Tutorial (PDF) | What's being worked on |
| Installation help (PDF) | Documentation on this wiki | Developing on Github |
| | Cookbook (working examples) | Google Summer of Code |
| | Discuss and ask questions | Report bugs (older issues) |

The latest release is Biopython 1.70, released on 10 July 2017.

# https://biopython.org/docs/1.78/api/Bio.html#subpackages

Check:

Bio. Blast.Record.Blast
Bio.Entrez
PDB.Structure



http://qcbprolab2020.readthedocs.io/en/latest/practical11.html

## Exercises

1. Write a python script that retrieves all the information present in SRA regarding PacBio sequencing performed on E.coli strain K12 (query term is "E.coli K12 wgs PacBio"). Print the number of results and for each id report the title, the accession id, the total number of spots and total number of bases sequenced.

Sample output:

```
Entries found: 11
Results for id: 9966072
WGS of E. coli K12 with PacBio HiFi
 - acc="SRR10971019"
 - total_spots="95514"
 - total_bases="1389500381"
Results for id: 6705337
PacBio RSII sequencing of E. coli K12
 - acc="SRR8154667"
 - total_spots="163482"
 - total_bases="1561717136"
Results for id: 6705336
PacBio RSII sequencing of E. coli K12
 - acc="SRR8154668"
 - total_spots="163482"
 - total_bases="897324802"
 ...
 ...
```

Show/Hide Solution

2. Write a python function that reads all the entries of a blast alignment file in .xml format (like blast_res_apple.xml and outputs all the HSPs (see example below) having bitscore > B, alignment length > A and minimum percentage of identity > I, where B, A and I are input thresholds. Hint: implement a filtering function: *filterHSPs(align, minBitscore = 0, minAlignLen = 0, minPercIdent = 0.1)*.

```
Alignments of MDC020656.85
    MDC020656.85: 1939-2593
    gi|125995253|dbj|AB270792.1|: 201263-201917
    Score:820.917 AlignLen:579 Id/Len:0.8812785388127854
    MDC020656.85: 1446-1935
    gi|125995253|dbj|AB270792.1|: 306490-306017
    Score:582.873 AlignLen:428 Id/Len:0.8629032258064516
    ....
    ....
```