

# 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

```
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*RENYEIHVGFLVTSMHQFKIYIHKQRX
HVIK*YQSYSQANPLQDV*VYI*CIKTF**RTKXRFSLIPLDLFRPRMDHLQALCSLNS
LSLXSLLSSTLLEXVLLSSPLVSKVEDX*RTPTPSXVRWME*PKXEEMIS*NSSWWPA
LCVLEREICFLLXNQHKXXELFTXKFL*HKETSQQLDFLSPSLWXALLCFVWALGF
YFKSSYA*IKXXWVXAQWAQLNXNVSLSPXRSYIAFMISLDFLINHT*LIQLIISIXQ*
LLTTRVYXCTIILGSN*QGSEVIGTNPIDYI*SNHLVN*NLILLHLL*RLHLII*KNSQ
AMSDI*PYIMATQANVEVVRTYSVGITM*FDPNLNNTLNHTIRVWIIYVKPLM*LFLLI
*FN*VV*ERFPLCQTPNVIIIPSYMIQLSRIGTLEGKFMRFLEWDFIMTIMHIQIKFNIRK
QRKHL*HIIKAIVMQIPFKVHRFXXXXXXSXTKRRLVLYLLS*T*DQGWTTSKPFAP*NP
```

# Exercise 4

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

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

[illegible]

**Consensus:**

[illegible]

```
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
```

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



4 alignments present in "file samples/alpha-globin.phy"

ENA | BAA205

ATGAGTCTCTCTGTAAGCAAGAGGCTGGTGAAGGCGCTATGGGCTAAGATCAGCCGCAAGGCGCATGATATCGGCGCTGAAGCTCTCGGCGAAGATGCTGACCCGCTCTACCCCT  
ACGACCAAGCATCTCTCTCTCCTGGGATGACCTCAGGCGCTGGGCTCCGGCTCTGTGAAGAAAGCATGGCAAGGTTATATCATGGTGCTGCTGGGCAAGCTGCGCTCTCTCAAAAATAGAC  
GACCTCTGTGGGAGGCTGGGCTCTCTGGGCAACCTCTCATGTTCTCAAGCTGCTGGTTGACCGGGCAACCTTCATGAGTCTCGCACAAATGTCATCTGTGCTCATCGGATCGCTC  
TCTCCGGAAGATCTCCCGCCAGAGGTTACATGTCATGTCAGCAAGTTTCTCGAAGATCTGGCTCTGGCTCTCTCTGAAAGAGTACGCTGA

FNA | CAA284

[illegible]

ENA | CAA237

ATGGTGCTGTCTCTCTGGCCGACAGAACCAAGCTCAAGCGCGCTGGGGTAAGGTGCGCGCCACGCTGGCGAGTATGGTGGCGAAGGCCCTGGAGAAGGATGTTCTCTGCTCTCTCCC  
ACACCAAGAGCTACTTCTCCGCACTTCGAC --- CTGAGCCACGGCTCTGCGCAAGTTAAGGCGCAGCCGAAGAAGGTGGCCGACGCGCTGACCAACGCTGGCGACGCTGGGAC  
GGTGGTGGCCCAAGCGGTGGTTCGCCCTGAGGCGACTCGACGCGCAAGCTCTGGTGGTGGCCGCGGTCAATCAAGCTCTTAAGCACTCTGGGCTGGTGGTGGCGCCAC  
TCCCGCCGCGAGTTCACCCCTGCGGTGCACGCTCTCCCTGGACAAGTTCCTGGCTCTGCTGAGCACCCTGCTGACCTCCAAATACCGGTTAA  
TNN174824

ENA | CAA240

ATGGTGTCTCTCTGGGGAAGCAAAACGATCAATCAAGGCTCCTGGGGGAAAGATTTGGTGAGCATGGTGCTGAATATGGAGCTGAAGCTCTGGAAAGGATGTTGTCTAGCTTCTCCC  
ACCCACCAAGCAAGCTATCTTCTCATCTTTGAT --- GTAAAGCAAGGCTCTGACCAAGTCAAGGTGACGCAAGGAAGTGCTGCATGCTGGCCGGTGTCTGACGACCAACCTGAT  
GACCTCGGCTGCTTCTGTCTGCTCGAGGACGATCGATCATCCCAAGCTGCTGTGTGGAATCCCGTCAACTTCTCAAGCTCTCGAGCACTCTGCTGCTGTGATCACTTGGCATGACCA  
CACCTTGGCATTTTACCCCGCCGGTACATGCTCTCTGGACAAAATCTTGCTCTCTGTAGCAAGCTGCTGCACTCCAAAGTCAGCTGTA

**Consensus:**

[illegible]

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`.

```
def computeConsensusV2(aldata, minFreq = 0.51):
    #this solution uses sets.
    #set(list("ATTAC")) 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
```

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

# Exercise 4

```
j=0    {"A": 4)}
j=1    {"T":2,"A":2}
j=2    {"-":3, "G":1}
j=3    {"G":3,"T":1}
```

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"

EN1A|BAA205  
ATGATGCTCTCTGTGAAGGACAAGCGTCTGTGGAAGCCCTATGGGCTAAGATGACGCCCAAGCGCATGATATCTGGCGCTGAAGCTCTCGGAGAATGCTGACCGCTACCTCT  
CAGACCAAGACCTACTCTCGCTCATGGGATGACCTGACCTGGGTGGGTCTGTGGAAGAGAGCTGCAAGTTGATCATGGGCGAGTGGCATGGCTCGGTTCCAAAATAGAC  
GAGTCTGTGGGAGATCGTGGCTCTCCGATGACGCAATCTATGCTCTCAAGTCGGTGTTGATCGACGGCCAACTCAATCTAGCATCTCGACACAATGCTGATGGCTCGGCATGCTCT  
TCTCTGGAGAGCTCTCCCCCAGAGGTTACATCTGTCAGTGGACAAAGTTTTCTCAGAAGCTTGGCTCTGGCTCTCTCGAGAAGTACCGCTAA  
EN1A|CAA284  
ATGTTCTGTAGCCAGGACTGAGAAGGACATCTGCTCTGCTGTGGAGCAAGATCTCCACAGGCGAGACGTCATTTGGCCAGGAGCTCGGAGAGGCTCTCTCTCTGCTACCTC  
CAGCGCAAGCAATCTCTCCGCACTTCGAC --- CTGACCTGGGCTCGGCGAGCTCGGCGGAGCGGCTGCAAGTGCGCTCAAGTGGGTGGCGCTGGGCGGCGCGGTCAAGAGACATCGAC  
AAGCTGAGCGAGCGGCTGTCCAAGTGGAGCGAGCTGACGCTACATGCTCTGCGCTGAGGACCGGTCAACTTCAGTTCTGTTCCCACTGCTCTGCTGGTACGTTGGCTCTGCGAC  
TCTCCCGCGACCTCTACGGCGACGCGCAGCTCGGCTGGGACAAGTCTGGTGGCTGATGCTGCTGCTGGGCTGCTGATGACGAGGAAGTACCGCTGA  
EN1A|CAA237  
ATGTGTCGTCTCTCTGCGCACAAGACCAAGTCAAGCGCTGCTGGGTGAAGTGTCCGCGCGACGCTGCGAGTATGTTGTCGGAGGCGCTGGAGAAGATGTTCTCTGCTCTCCC  
ACCAACCAAGACCTACTCTCCGCACTTCGAC --- CTGAGCAAGGCTCTGCCCAGATTAAGGCGACGCGCAAGGAAGTGTTGGCCGACGCTGCTGACCAACCGCGTGGCGACGTGGAG  
GACATGCGCAACCGGCTGTCGCGCTGAGCGAGCTGACGCGCAAGGCTTGGGTGGAGACCGGTCAACTTCAGTCTTAAGCACAAGCTGCTGGTGGACCTGCTGGGCGGCCAC  
CTCCCGCGGAGTTCACCCCTCGGGTGGCGACGCTCTCTGGACAAGTTCTGCGCTCTGTGAGCAACGCTGCTGACCTTCAAATAGCGTTAA  
EN1A|CAA240  
ATGTGTCCTCTCTGGGGAAGACAAGAACCAATCAAGCGTCTCTGGGGAGAGATTTGGTGGGCTCATGTTGCTGAATATGGAGCTGAAGCCCTGGAAAGAGTGTTCGCTAGCTCTCCG  
ACCCACCAAGACATCTTTTCCATCTTGAAT --- GTAAGCACAGGCTCTGCCCAGTCAAGGCTACGCGCAAGAGTTCGGCATGCTGGCGAGTGTGACGAGGACATCGAC  
GACCTGCGCGGCTGCTTGTCTGCTCTGAGCGACGTCGATGCTGCCCAAGCTGGGCTGTGGAGACCTCGGTCAACTTCAGTCTCTGAGCACAAGCTGCTGCTGGTGGACCTTGGCTAGCCAC  
CACTCTCGGATTTTCCGCGCGGCTATGTCCTCTTGG6ACAAGTTCTCTGCTCTGTGATGACCGTGTGACCTCAAGTACGCTGAA

consensus: ATG????CTCT?????GACAAAGACCA?C?TCAAG6CC????TG6GG?AAGATC?GC?C?CA?GC?G?GA????TG6?GC?GA?GCCCTGG?AGGATGTT?C???CT?CC  
 ????T?ACCAAGACTACTTCCC?CACTT?GA?---CTAG6CT?GGCT?GC?CA?GT?AAGG?CAGCGAAGAAGT?G?G?G?GC?TG6CT?A?GC?GT?G??T?AC?T?G  
 ACAG?T?C?G?G?GC?GTGC?C?T?GACG?C?TGCA?GC?A?AAGACTG?G?GTGACCGGTCACCT?CAAG?TCT?C?CCACTGCTGCTGTTGAGCT?GGC?C?  
 AC?TCC?GC?CGA?T?CAC?CC?G?G?GTG?CAGGC?T?C?TGGAACAAGTTCT?C?T?G?G?G?G?C?GT?C?TG?C?T?G?A?G?T?G?AAGT?CAAG?G?TAA

```
from collections import Counter
```

```
def computeConsensusV(aldata, minFreq = 0.51):
    #this solution uses collections.Counter
    #collections.Counter("ATTAC") returns the dictionary: {'T': 3, 'A': 2, 'C': 1}
    #
    consensus = ""
    if minFreq < 0.51:
        print("Warning: minFreq ({}).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...

```
#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** (NCBI'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
<b>nr (nt) default</b>	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
<b>nr default</b>	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

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 .
```

Query results can be  
parsed with the  
methods of the  
**module**

`Bio.Blast.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 through 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 iterator 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 evaluate 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/amino acids 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 PR00903 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: TCTCTTCTGTCAACCCACGCGCCTTTGGCACAATGAAGTGGGTAACCTTTATTTCCCTTCTTTTCTCTTTAGCTCGG
Match: |||
Subjc: TCTCTTCTGTCAACCCACGCGCCTTTGGCACAATGAAGTGGGTAACCTTTATTTCCCTTCTTTTCTCTTTAGCTCGG
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: TCTCTTCTGTCAACCCACGCGCCTTTGGCACAATGAAGTGGGTAACCTTTATTTCCCTTCTTTTCTCTTTAGCTCGG
Match: |||
Subjc: TCTCTTCTGTCAACCCACGCGCCTTTGGCACAATGAAGTGGGTAACCTTTATTTCCCTTCTTTTCTCTTTAGCTCGG
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: TCTCTTCTGTCAACCCACGCGCCTTTGGCACAATGAAGTGGGTAACCTTTATTTCCCTTCTTTTCTCTTTAGCTCGG
Match: |||
Subjc: TCTCTTCTGTCAACCCACGCGCCTTTGGCACAATGAAGTGGGTAACCTTTATTTCCCTTCTTTTCTCTTTAGCTCGG
```

```
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/>

NCBI

Resources

How To

luca.bianco@fmach.it My NCBI Sign Out

NCBI

National Center for Biotechnology Information

All Databases

Search

NCBI Home

Resource List (A-Z)

All Resources

Chemicals & Bioassays

Data & Software

DNA & RNA

Domains & Structures

Genes & Expression

Genetics & Medicine

Genomes & Maps

Homology

Literature

Proteins

Sequence Analysis

Taxonomy

Training & Tutorials

Variation

Welcome to NCBI

The National Center for Biotechnology Information advances science and health by providing access to biomedical and genomic information.

[About the NCBI](#) | [Mission](#) | [Organization](#) | [NCBI News & Blog](#)

Submit

Deposit data or manuscripts into NCBI databases

Download

Transfer NCBI data to your computer

Learn

Find help documents, attend a class or watch a tutorial

Develop

Use NCBI APIs and code libraries to build applications

Analyze

Identify an NCBI tool for your data analysis task

Research

Explore NCBI research and collaborative projects

Popular Resources

[PubMed](#)

[Bookshelf](#)

[PubMed Central](#)

[BLAST](#)

[Nucleotide](#)

[Genome](#)

[SNP](#)

[Gene](#)

[Protein](#)

[PubChem](#)

NCBI News & Blog

Bulk track hub settings now in Genome Data Viewer

24 Oct 2019

You now have access to bulk settings options for track hubs in the Genome

New PubMed Updates: User Guide, MyNCBI, and more

21 Oct 2019

As you may have heard, we are working on a new version of PubMed and we've

Genome in a Bottle Structural Variants released in dbVar

21 Oct 2019

The latest dbVar data release includes the Genome in a Bottle benchmark

[More...](#)

# Getting data from NCBI

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

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`.

1. `res_handle = Entrez.einfo(db)` returns a summary of the Entrez 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'.

As a list:

['pubmed', 'protein', 'nucore', '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

```
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"])
```

# 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('TTTGGTCTGCTGGGTAGGTACTCATAAAACAAACCAACCGAAGCCTCCGAACC...AAA', IUPACAmbiguousDNA(

SEQUENCE:
TTTGGTCTGCTGGGTAGGTACTCATAAAACAAACCAACCGAAGCCTCCGAACCGACCACCAATGACGGCCCCAAACGGCGCCGTC
```

```
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.

The screenshot displays the Protein Data Bank (PDB) website homepage. The header features the RCSB PDB logo and navigation links: Deposit, Search, Visualize, Analyze, Download, Learn, and More. A search bar is prominently displayed with the placeholder text "Enter search term(s)". Below the header, the main content area is divided into several sections. On the left, a vertical sidebar contains links for Welcome, Deposit, Search, Visualize, Analyze, Download, and Learn. The central section is titled "A Structural View of Biology" and includes a paragraph about the database's mission to provide structural information to the scientific community. Below this, there is a section for "COVID-19 CORONAVIRUS Resources" featuring a 3D model of the virus. To the right, the "October Molecule of the Month" section highlights the Capsaicin Receptor TRPV1 with a 3D model. The bottom of the page features a "Latest Entries" section with a 3D model of a protein structure (6KMN) and a "Features & Highlights" section listing various tools and resources available on the site. The footer contains links for About, Help, and Partners, along with a statement that the PDB is a member of the International Nucleic Acid Database (INSD) and the International Protein Data Bank (IPIB).

RCSB PDB Deposit Search Visualize Analyze Download Learn More

170172 Biological Macromolecular Structures Enabling Breakthroughs in Research and Education

Enter search term(s)

Advanced Search | Browse Annotations

PDB-101 PDB PDB-Data Bank PDB-Data Bank PDB-Data Bank PDB-Data Bank PDB-Data Bank

Welcome

Deposit

Search

Visualize

Analyze

Download

Learn

A Structural View of Biology

This resource is powered by the Protein Data Bank archive-information about the 3D shapes of proteins, nucleic acids, and complex assemblies that helps students and researchers understand all aspects of biomedicine and agriculture, from protein synthesis to health and disease.

As a member of the wwPDB, the RCSB PDB curates and annotates PDB data. The RCSB PDB builds upon the data by creating tools and resources for research and education in molecular biology, structural biology, computational biology, and beyond.

COVID-19 CORONAVIRUS Resources

October Molecule of the Month

Capsaicin Receptor TRPV1

Latest Entries As of Tue Oct 20 2020

6KMN

Crystal Structure of Dye Decolorizing peroxidase from *Bacillus subtilis*

Features & Highlights

Sequence Alignments in Search Results

For Sequence Similarity searches, display search results as "Polymer Entities" to view amino acid mismatches.

Build Customizable Tabular Reports of PDB Data

After searching for a set of structures, create a table of information by selecting the columns of your choice. Tables can be saved in CSV or JSON formats.

Documentation for New and Improved APIs

Learn the concepts and syntax behind these new services. Legacy APIs will be discontinued November 2020

News

Publications

PDB Turns 49

Today is the anniversary of the 1971 announcement of the PDB. wwPDB is celebrating by looking ahead to golden anniversary symposia and events planned for 2021 = 10/20/2020

Happy Birthday, Irving Geis

Celebrate Geis' birthday (October 18, 1908) with a tour of the Tour the Geis Digital Archive developed in collaboration with HHMI. = 10/18/2020

Join Us at the STEMteachersEXPO

RCSB PDB will be part of Strengthening virtual Chemistry and Biology instruction Through 3D Modeling on Saturday October 24 = 10/24/2020

PDB at a Glance 170172 Structures 49585 Structures of Human Sequences 12418 Nucleic Acid Containing Structures More Statistics

About About Us Citing Us Publications Team Careers Usage & Privacy

Help Contact Us Help Topics Website FAQ Glossary

RCSB Partners Nucleic Acid Database

wwPDB Partners RCSB PDB PDB-Data Bank PDB-Data Bank PDB-Data Bank

RCSB PDB (collaborator) is hosted by

RUTGERS UC San Diego SDSC UCSF

RCSB PDB is a member of the

# Protein Data Bank (PDB)

First of all:

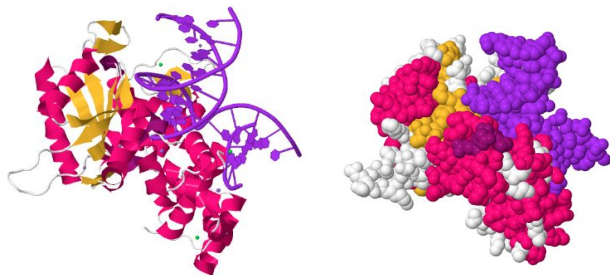
```
from Bio.PDB import *
```

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.

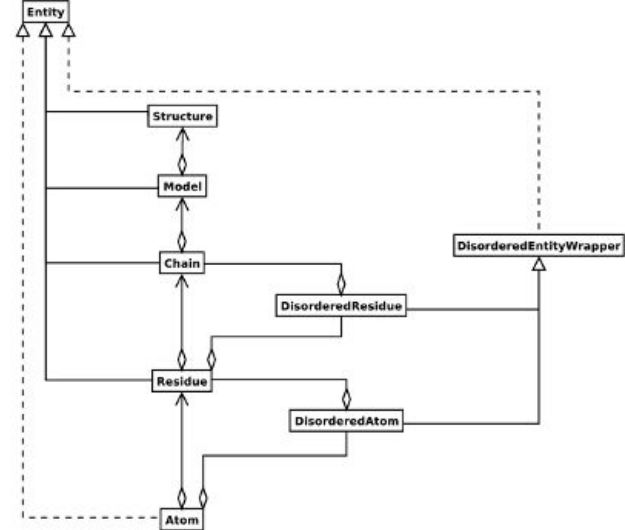
The basic syntax is:

```
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](#)



Python Tools for  
Computational  
Molecular Biology

[Documentation](#)

[Download](#)

[Mailing lists](#)

[News](#)

[Biopython Contributors](#)

[Scriptcentral](#)

[Source Code](#)

[GitHub project](#)

Biopython version 1.70

© 2017. All rights reserved.

## 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 development 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.


<a href="#">Get Started</a>	<a href="#">Get help</a>	<a href="#">Contribute</a>
<a href="#">Download Biopython</a>	<a href="#">Tutorial (PDF)</a>	<a href="#">What's being worked on</a>
<a href="#">Installation help (PDF)</a>	<a href="#">Documentation on this wiki</a>	<a href="#">Developing on Github</a>
	<a href="#">Cookbook (working examples)</a>	<a href="#">Google Summer of Code</a>
	<a href="#">Discuss and ask questions</a>	<a href="#">Report bugs (older issues)</a>

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



biopython

Search docs

API CONTENTS:

- Bio package
  - Subpackages
    - Bio.Affy package
    - Bio.Align package
    - Bio.AlignIO package
    - Bio.Alphabet package
    - Bio.Application package
    - Bio.Blast package
      - Submodules
      - Module contents
    - Bio.CAPS package
    - Bio.Cluster package
    - Bio.Compass package
    - Bio.Crystal package
    - Bio.Data package
    - Bio.Emboss package
    - Bio.Entrez package
    - Bio.ExPASy package
    - Bio.FSPP package
    - Bio.GenBank package
    - Bio.Geo package
    - Bio.Graphics package

Biopython v: 1.78

» Bio package » Bio.Blast package » Bio.Blast.Record module

[Edit on GitHub](#)

[Previous](#)

[Next](#)

## Bio.Blast.Record module

Record classes to hold BLAST output.

Classes: Blast Holds all the information from a blast search. PSIBlast Holds all the information from a psi-blast search.

Header Holds information from the header. Description Holds information about one hit description. Alignment Holds information about one alignment hit. HSP Holds information about one HSP. MultipleAlignment Holds information about a multiple alignment. DatabaseReport Holds information from the database report. Parameters Holds information from the parameters.

**Bio.Blast.Record.fmt\_(value, format\_spec='%s', default\_str='<unknown>')**

Ensure the given value formats to a string correctly.

**class Bio.Blast.Record.Header**

Bases: `object`

Saves information from a blast header.

Members: application The name of the BLAST flavor that generated this data. version Version of blast used. date Date this data was generated. reference Reference for blast.

query Name of query sequence. query\_letters Number of letters in the query sequence. (int)

database Name of the database. database\_sequences Number of sequences in the database. (int) database\_letters Number of letters in the database. (int)

**\_\_init\_\_(self)**

Initialize the class.

**class Bio.Blast.Record.Description**

Bases: `object`



## 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: 9966672
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(aligned, minBitscore = 0, minAlignLen = 0, minPerIdent = 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:0.8629032250064516
....
....
```