

# Scientific Programming

## Practical 11

---

### Introduction

Luca Bianco - Academic Year 2017-18  
luca.bianco@fmach.it

# 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.

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

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

where `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 and `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") and `expect` (the e-value threshold).

# 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.

```
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 the `entrez_query` for example we can limit the search to specific organisms with:

```
entrez_query='"Malus Domestica" [Organism]'.
```

# BLAST: parsing the output

Query results can be parsed with the methods of the module

`Bio.Blast.NCBIXML`

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

or

```
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

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

We can save the entries in a file

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()
```

# The BLAST record class

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

It is composed of two types of information:

- Descriptions
- Alignments

1. *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 two types of information:

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](#)) on the human genome and report all the information reported by BLAST. (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|1046552723|ref|NM_000477.6| Homo sapiens albumin (ALB), mRNA
SCORE:4304.0
N.ALIGN:1
E-VAL:0.0
TITLE:gi|28591|emb|V00495.1| H.sapiens mRNA for serum albumin
SCORE:4252.0
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](#)) on the human genome and report all the information reported by BLAST. (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", "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))

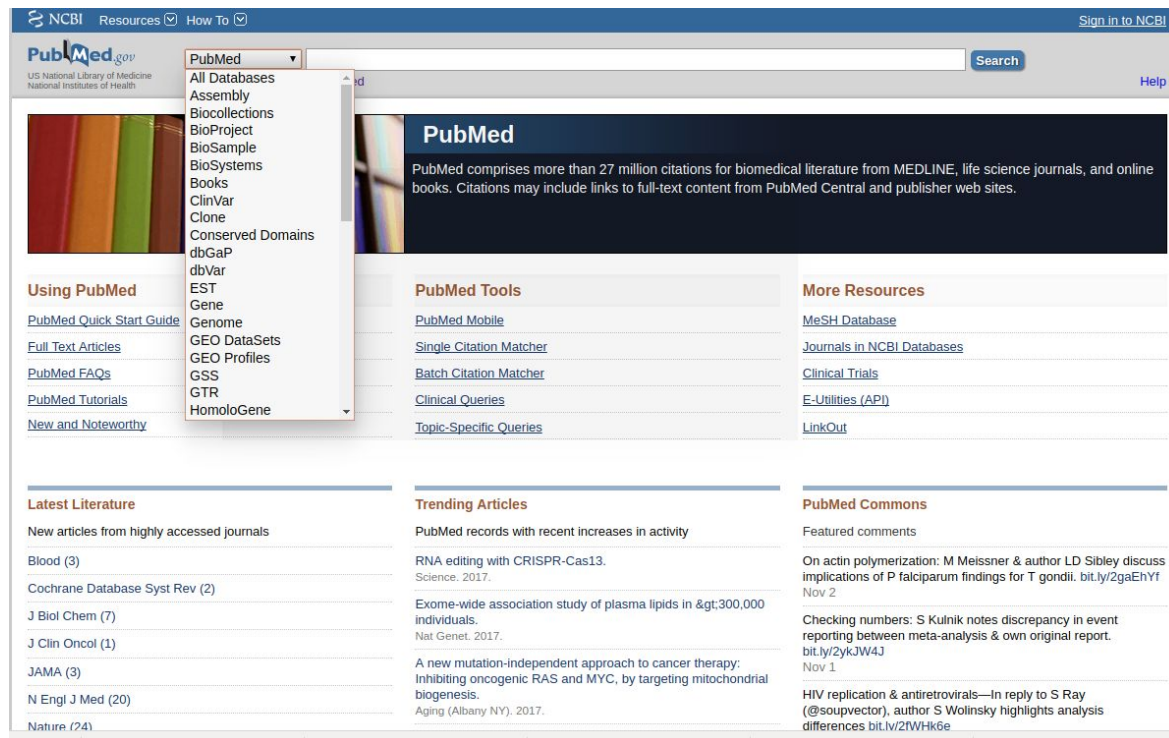
    for a in res.alignments:
        print("Align Title:{}\nAlign Len: {}".format(a.title, a.length))

        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()
```

# Getting data from NCBI



The screenshot shows the NCBI PubMed homepage. The top navigation bar includes the NCBI logo, 'Resources', and 'How To'. The main header features the 'PubMed.gov' logo, a search bar, and a 'Sign in to NCBI' link. A dropdown menu is open under the 'PubMed' logo, listing various databases and tools. The main content area is divided into several sections: 'PubMed' (overview), 'PubMed Tools' (links to mobile, citation matchers, queries, and topic-specific queries), 'More Resources' (links to MeSH, journals, clinical trials, utilities, and linkout), 'Latest Literature' (new articles from highly accessed journals), 'Trending Articles' (recent increases in activity), and 'PubMed Commons' (featured comments).

**PubMed**

PubMed comprises more than 27 million citations for biomedical literature from MEDLINE, life science journals, and online books. Citations may include links to full-text content from PubMed Central and publisher web sites.

**PubMed Tools**

- [PubMed Mobile](#)
- [Single Citation Matcher](#)
- [Batch Citation Matcher](#)
- [Clinical Queries](#)
- [Topic-Specific Queries](#)

**More Resources**

- [MeSH Database](#)
- [Journals in NCBI Databases](#)
- [Clinical Trials](#)
- [E-Utilities \(API\)](#)
- [LinkOut](#)

**Using PubMed**

- [PubMed Quick Start Guide](#)
- [Full Text Articles](#)
- [PubMed FAQs](#)
- [PubMed Tutorials](#)
- [New and Noteworthy](#)

**Latest Literature**

New articles from highly accessed journals

- Blood (3)
- Cochrane Database Syst Rev (2)
- J Biol Chem (7)
- J Clin Oncol (1)
- JAMA (3)
- N Engl J Med (20)
- Nature (24)

**Trending Articles**

PubMed records with recent increases in activity

- RNA editing with CRISPR-Cas13. Science. 2017.
- Exome-wide association study of plasma lipids in >300,000 individuals. Nat Genet. 2017.
- A new mutation-independent approach to cancer therapy: Inhibiting oncogenic RAS and MYC, by targeting mitochondrial biogenesis. Aging (Albany NY). 2017.

**PubMed Commons**

Featured comments

- On actin polymerization: M Meissner & author LD Sibley discuss implications of P falciparum findings for T gondii. bit.ly/2gaEhYf Nov 2
- Checking numbers: S Kulnik notes discrepancy in event reporting between meta-analysis & own original report. bit.ly/2yk3W4J Nov 1
- HIV replication & antiretrovirals—In reply to S Ray (@soupvector), author S Wolinsky highlights analysis differences bit.ly/2fWHk6e



# 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`.

In particular the module (complete info on [Entrez module are here](#)) provides, among the others, the following functions:

1. `res_handle = Entrez.einfo(db)` returns a summary of the Entez databases as a results handle.  
`db` is an optional paramter 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',  
'nucgss', 'nucest', 'structure', 'sparcle', 'genome',  
'annotinfo', 'assembly', 'bioproject', 'biosample',  
'blastdbinfo', 'books', 'cdd', 'clinvar', 'clone',  
'gap', 'gapplus', 'grasp', 'dbvar', 'gene', 'gds',  
'geoprofiles', 'homologene', 'medgen', 'mesh',  
'ncbisearch', 'nlmcatalog', 'omim', 'orgtrack', 'pmc',  
'popset', 'probe', 'proteinclusters', 'pcassay',  
'biosystems', 'pccompound', 'pcsubstance',  
'pubmedhealth', 'seqannot', 'snp', 'sra', 'taxonomy',  
'biocollections', 'unigene', 'gencoll', 'gtr']
```

Entries count: 4666883

LastUpdate: 2017/10/31 20:38

Description: SRA Database

```
from Bio import Entrez  
  
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)  
print("")  
print("Entries count:", res["DbInfo"]["Count"])  
print("LastUpdate:", res["DbInfo"]["LastUpdate"])  
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('TTTGGTCTGCTGGGTAGGTACTCATAAAAAACAAACCAACCGAAGCCTCCGAACC...AAA', IUPACAmbiguousDNA(

SEQUENCE:
TTTGGTCTGCTGGGTAGGTACTCATAAAAAACAAACCAACCGAAGCCTCCGAACCGACCAACCAATGACGGCCCCAAACGGCGCGTC
```

```
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)

PDB is a database of structural information of 3D shapes of proteins, nucleic acids, and complex assemblies. The database currently contains more than 134,000 total structures.

RCSB PDB

Deposit ▾

Search ▾

Visualize ▾

Analyze ▾

Download ▾

Learn ▾

More ▾

MyPDB Login ▾

RCSB

PDB

PROTEIN DATA BANK

134854 Biological  
Macromolecular Structures  
Enabling Breakthroughs in  
Research and Education

Go

[Advanced Search](#) | [Browse by Annotations](#)

PDB-101

Worldwide  
PDB  
PROTEIN DATA BANK

EMDataBank

EMBio

WORLDWIDE  
NUCLEIC ACID  
DATABASE

Worldwide  
Protein Data Bank  
Foundation

[f](#)[t](#)[v](#)[y](#)

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.

### Video: How Enzymes Work



## November Molecule of the Month



Aspartate Transcarbamoylase

Latest Entries

As of Tuesday Oct 31

Features & Highlights

News

Publications ▾



# Protein Data Bank (PDB)

```
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.

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`

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/")
```

```
Structure exists: 'file_samples/3c2k.cif'
Structure exists: 'file_samples/3c2l.cif'
```

# 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.

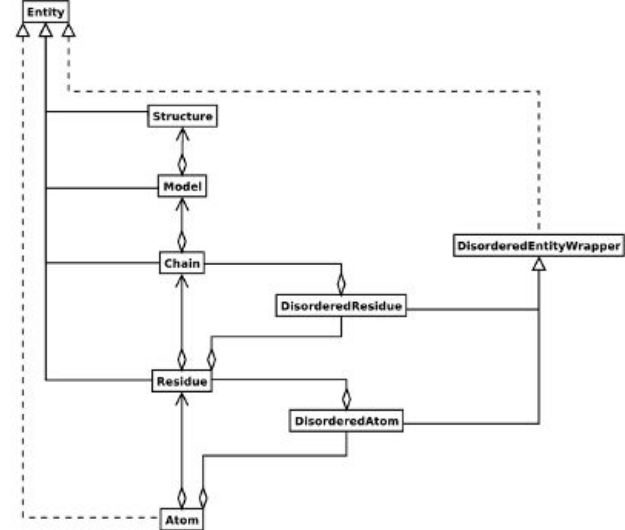
The basic syntax is:

```
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.



<http://biopython.org/DIST/docs/api/>

Check:

Blast.Record.Blast  
Bio.Entrez  
PDB.Structure

Table of Contents

[Everything](#)

**Modules**

[Bio](#)

[Bio.Affy](#)

[Bio.Affy.CelFile](#)

[Bio.Align](#)

[Bio.Align.AlignInfo](#)

[Bio.Align.Applications](#)

[Bio.Align.Applications.ClustalOmega](#)

[Bio.Align.Applications.Clustalw](#)

[Bio.Align.Applications.Dialign](#)

[Bio.Align.Applications.MSAProbs](#)

[Bio.Align.Applications.Mafft](#)

**Everything**

**All Classes**

[Bio.Affy.CelFile.ParserError](#)

[Bio.Affy.CelFile.Record](#)

[Bio.Align.AlignInfo.PSSM](#)

[Bio.Align.AlignInfo.SummaryInfo](#)

[Bio.Align.Applications.ClustalOmega.ClustalOmegaCommandline](#)

[Bio.Align.Applications.Clustalw.ClustalwCommandline](#)

[Bio.Align.Applications.Dialign.DialignCommandline](#)

[Bio.Align.Applications.MSAProbs.MSAProbsCommandline](#)

[Bio.Align.Applications.Mafft.MafftCommandline](#)

[Bio.Align.Applications.Muscle.MuscleCommandline](#)

[Bio.Align.Applications.Prank.PrankCommandline](#)

[Bio.Align.Applications.Probcons.ProbconsCommandline](#)

[Bio.Align.Applications.TCoffee.TCoffeeCommandline](#)

[Bio.Align.MultipleSeqAlignment](#)

[Bio.AlignIO.ClustalIO.ClustalIterator](#)

[Bio.AlignIO.ClustalIO.ClustalWriter](#)

[Bio.AlignIO.EmbossIO.EmbossIterator](#)

[Bio.AlignIO.EmbossIO.EmbossWriter](#)

[Bio.AlignIO.Interfaces.AlignmentIterator](#)

[Bio.AlignIO.Interfaces.AlignmentWriter](#)

[Bio.AlignIO.Interfaces.SequentialAlignmentWriter](#)

[Bio.AlignIO.MafftIO.MafftIndex](#)

[Bio.AlignIO.MafftIO.MafftWriter](#)

[Bio.AlignIO.MauveIO.MauveIterator](#)

[Bio.AlignIO.MauveIO.MauveWriter](#)

[Bio.AlignIO.NexusIO.NexusWriter](#)

[Bio.AlignIO.PhylipIO.PhylipIterator](#)

[Bio.AlignIO.PhylipIO.PhylipWriter](#)

[Bio.AlignIO.PhylipIO.RelaxedPhyIpIterator](#)

[Bio.AlignIO.PhylipIO.RelaxedPhyIpWriter](#)

[Bio.AlignIO.PhylipIO.SequentialPhyIpIterator](#)

[Bio.AlignIO.PhylipIO.SequentialPhyIpWriter](#)

Trees Indices Help

[\[ Module Hierarchy | Class Hierarchy \]](#)

**Module Hierarchy**

- Bio:** Collection of modules for dealing with biological data in Python.
  - Bio.Affy:** Deal with Affymetrix related data such as cel files.
    - Bio.Affy.CelFile:** Reading information from Affymetrix CEL files version 3 and 4.
  - Bio.Align:** Code for dealing with sequence alignments.
    - Bio.Align.AlignInfo:** Extract information from alignment objects.
    - Bio.Align.Applications:** Alignment command line tool wrappers.
      - Bio.Align.Applications.ClustalOmega:** Command line wrapper for the multiple alignment program Clustal Omega.
      - Bio.Align.Applications.Clustalw:** Command line wrapper for the multiple alignment program Clustal W.
      - Bio.Align.Applications.Dialign:** Command line wrapper for the multiple alignment program DIALIGN2-2.
      - Bio.Align.Applications.MSAProbs:** Command line wrapper for the multiple sequence alignment program MSAProbs.
      - Bio.Align.Applications.Mafft:** Command line wrapper for the multiple alignment programme MAFFT.
      - Bio.Align.Applications.Muscle:** Command line wrapper for the multiple alignment program MUSCLE.
      - Bio.Align.Applications.Prank:** Command line wrapper for the multiple alignment program PRANK.
      - Bio.Align.Applications.Probcons:** Command line wrapper for the multiple alignment program PROBCONS.
      - Bio.Align.Applications.TCoffee:** Command line wrapper for the multiple alignment program TCOFFEE.
  - Bio.AlignIO:** Multiple sequence alignment input/output as alignment objects.
    - Bio.AlignIO.ClustalIO:** Bio.AlignIO support for "clustal" output from CLUSTAL W and other tools.
    - Bio.AlignIO.EmbossIO:** Bio.AlignIO support for "emboss" alignment output from EMBOSS tools.
    - Bio.AlignIO.FastaIO:** Bio.AlignIO support for "fasta-m10" output from Bill Pearson's FASTA tools.
    - Bio.AlignIO.Interfaces:** AlignIO support module (not for general use).
    - Bio.AlignIO.MafftIO:** Bio.AlignIO support for the "ma" multiple alignment format.
    - Bio.AlignIO.MauveIO:** Bio.AlignIO support for "mafa" output from Mauve/ProgressiveMauve.
    - Bio.AlignIO.NexusIO:** Bio.AlignIO support for the "nexus" file format.
    - Bio.AlignIO.PhyIpIO:** AlignIO support for "phyip" format from Joe Felsenstein's PHYLIP tools.
    - Bio.AlignIO.StockholmIO:** Bio.AlignIO support for "stockholm" format (used in the PFAM database).
  - Bio.Alphabet:** Alphabets used in Seq objects etc to declare sequence type and letters.
    - Bio.Alphabet.IUPAC:** Standard nucleotide and protein alphabets defined by IUPAC.
    - Bio.Alphabet.Reduced:** Reduced alphabets which lump together several amino-acids into one letter.
  - Bio.Application:** General mechanisms to access applications in Biopython.
  - Bio.Blast:** Code for dealing with BLAST programs and output.
    - Bio.Blast.Applications:** Definitions for interacting with BLAST related applications.
    - Bio.Blast.NCBIStandalone:** Code for calling standalone BLAST and parsing plain text output (DEPRECATED).
    - Bio.Blast.NCBIWWW:** Code to invoke the NCBI BLAST server over the internet.
    - Bio.Blast.NCBIXML:** Code to work with the BLAST XML output.
    - Bio.Blast.ParseBlastTable:** A parser for the NCBI blastpgp version 2.2.5 output format. Currently only supports the '-m 9' option, (table w/ annotations). Returns a BlastTableRec instance
    - Bio.Blast.Record:** Record classes to hold BLAST output.
  - Bio.CAPS:** Cleaved amplified polymorphic sequence (CAPS) markers.
  - Bio.Cluster:** Cluster Analysis.
    - Bio.Cluster.cluster:** C Clustering Library
  - Bio.Compass:** Code to deal with COMPASS output, a program for profile/profile comparison.
  - Bio.Crystal:** Represent the NDB Atlas structure (a minimal subset of PDB format).
  - Bio.Data:** Collections of various bits of useful biological data.
    - Bio.Data.CodonTable:** Codon tables based on those from the NCBI.
    - Bio.Data.IUPACData:** Information about the IUPAC alphabets.
    - Bio.Data.SCOPData:** Additional protein alphabets used in the SCOP database and PDB files.
  - Bio.DocSQL:** Bio.DocSQL: easy access to DB API databases (DEPRECATED).

## Exercises

1. Write a python function `retrieve_sequences(search_term, number, outfile)` that retrieves the first `number` of sequences from NCBI's "nucleotide" database having a search term `term` (hint: use `term` and `retmax` parameters of `Entrez.esearch`) and stores them in a fasta file `outfile` (hint: use `SeqIO.write`). Test your code retrieving the first 5 entries having search term "starch AND Malus Domestica [Organism]"

Show/Hide Solution

2. Write a python function that aligns the sequences in the file created at point 1. ([here](#) you can find mine) against the NCBI nr database limiting the hits to the Malus Domestica organism (parameter `entrez_query="Malus Domestica" [Organism]"` in `qblast`) and prints to screen the following info for each hsp:
  1. The title;
  2. Score and e-value;
  3. The number of alignments on the same subject, the number of identities and positives and the alignment length;
  4. The number of mismatches and the list of their positions (hint: you can use the `match` string and look for `" "`).

Show/Hide Solution

3. Write a python function `getPublicationInfo(title_term, other_term)` that retrieves the first 20 pubmed publications having the `title_term` in the title and `other_term` somewhere else in the text (hint use: "Title" and "[Other Term]" as `esearch` parameter `term`). For each publication print:
  1. the title
  2. authors
  3. journal
  4. year of publication (hint: get and split properly the "PubDate" entry)