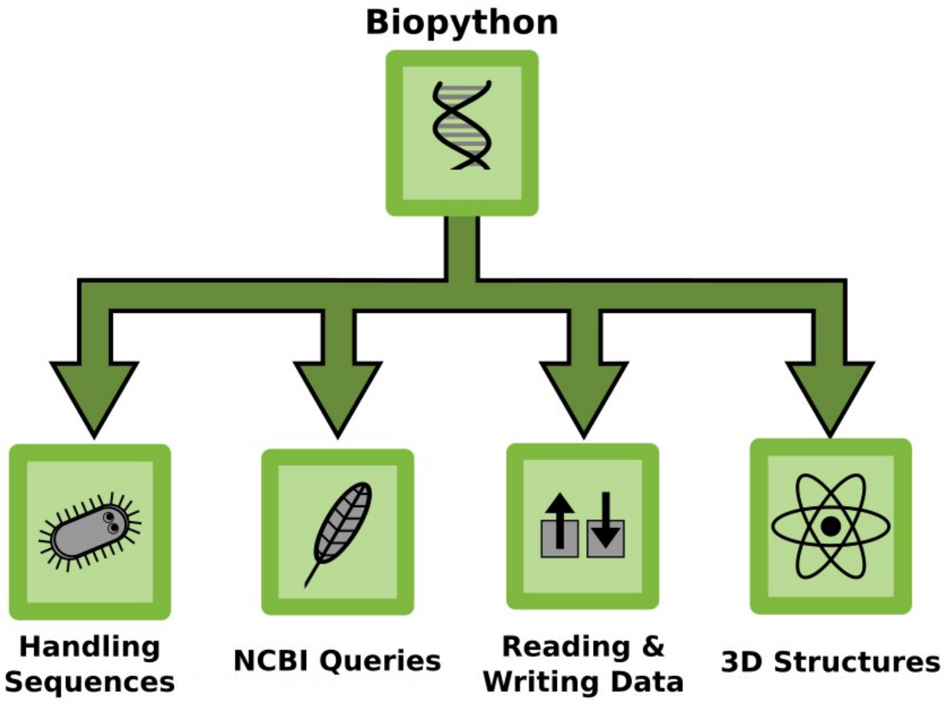
Lecture 5
Biopython

Lecturer: Pieter De Bleser

Bioinformatics Core Facility, IRC

Slides derived from: I. Holmes, Department of Statistics, University of Oxford; M. Schroeder, M. Adasme, A. Henschel, Biotechnology Center, TU Dresden; S. Spielman, CCBB, University of Texas; Jehan-François Pâris, jfparis@uh.edu**;Didi Amar, rozovr@tau.ac.il**

What is Biopython?

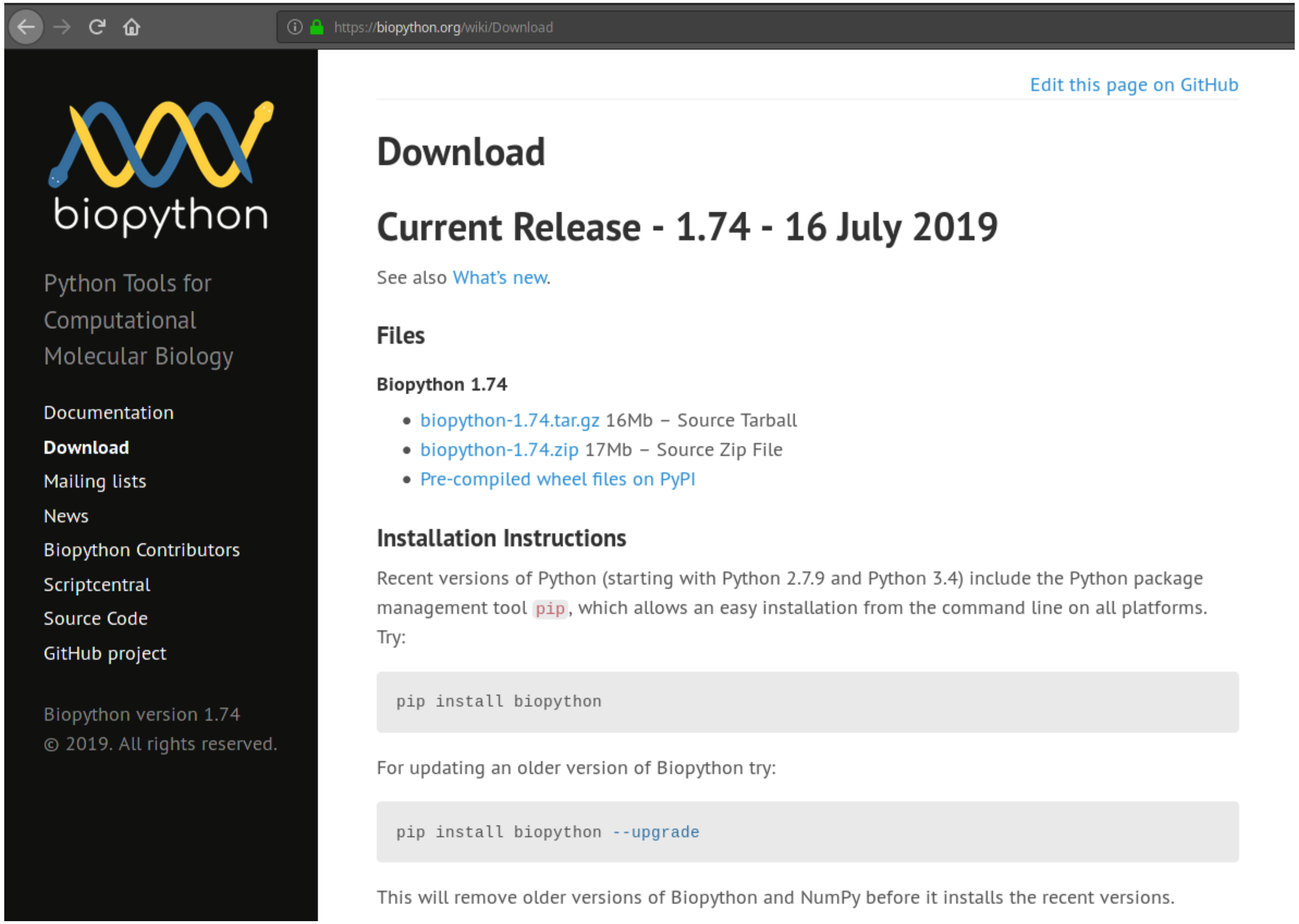


Biopython is a Python library for reading and writing many common biological data formats. It contains some functionality to perform calculations, in particular on 3D structures.
Started in 1999 (only Bioperl starting in 1995 is older).
The library and documentation can be found at: https://biopython.org/

Biopython is not ...

- one tool for everything
- a collection of algorithms
- a solution for your specific problem
- specialized on NGS
- calculating statistics..
- it is quite useful anyway

How to get Biopython?


https://biopython.org/wiki/Download

biopython

Python Tools for Computational Molecular Biology

- Documentation
- **Download**
- Mailing lists
- News
- Biopython Contributors
- Scriptcentral
- Source Code
- GitHub project

Biopython version 1.74
© 2019. All rights reserved.

Edit this page on GitHub

Download

Current Release - 1.74 - 16 July 2019

See also What's new.

Files

Biopython 1.74

- biopython-1.74.tar.gz 16Mb – Source Tarball
- biopython-1.74.zip 17Mb – Source Zip File
- Pre-compiled wheel files on PyPI

Installation Instructions

Recent versions of Python (starting with Python 2.7.9 and Python 3.4) include the Python package management tool `pip`, which allows an easy installation from the command line on all platforms. Try:

```
pip install biopython
```

For updating an older version of Biopython try:

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

This will remove older versions of Biopython and NumPy before it installs the recent versions.

Data formats supported by biopython

Uses	Notes
Blast	finds regions of local similarity between sequences
ClustalW	multiple sequence alignment program
GenBank	NCBI sequence database
PubMed and Medline	Document database
ExPASy	SIB resource portal (Enzyme and Prosite)
SCOP	Structural Classification of Proteins (e.g. ‘dom’,’lin’)
UniGene	computationally identifies transcripts from the same locus
SwissProt	annotated and non-redundant protein sequence database

Some of the other principal functions of biopython.

- A standard sequence class that deals with sequences, ids on sequences, and sequence features.
- Tools for performing common operations on sequences, such as translation, transcription and weight calculations.
- Code to perform classification of data using k Nearest Neighbors, Naive Bayes or Support Vector Machines.
- Code for dealing with alignments, including a standard way to create and deal with substitution matrices.
- Code making it easy to split up parallelizable tasks into separate processes.
- GUI-based programs to do basic sequence manipulations, translations, BLASTing, etc.

Biopython – The Sequence object

bioseq1.py

```
from Bio.Seq import Seq
from Bio.Alphabet.IUPAC import unambiguous_dna, ambiguous_dna


unamb_dna_seq = Seq("ACGT", unambiguous_dna)
ambig_dna_seq = Seq("ACRGT", ambiguous_dna)


print( unamb_dna_seq )
print( ambig_dna_seq.alphabet )

# A Seq object in python acts like a normal python string.

for letter in ambig_dna_seq:
    print( letter )

print( 'Length: ', len(ambig_dna_seq) )
print( ambig_dna_seq[4:12] )
print( ambig_dna_seq[::-1] )
print( str(ambig_dna_seq) )
```

```
> python3 bioseq1.py
ACGT
IUPACAmbiguousDNA()
A
C
R
G
T
Length:  5
T
TGRCA
ACRGT
```

Nucleotide counts, transcription, translation

bioseq2.py

```
from Bio.Seq import Seq
from Bio.Alphabet.IUPAC import unambiguous_dna, ambiguous_dna

my_seq = Seq( "AGTACACTGGTA", unambiguous_dna )

print( my_seq )

# How many As,Cs, Ts and Gs in this sequence?
print( '# A: ',my_seq.count("A") )
print( '# C: ',my_seq.count("C") )
print( '# G: ',my_seq.count("G") )
print( '# T: ',my_seq.count("T") )

# To get the GC nucleotide content:
from Bio.SeqUtils import GC
print( 'GC content: ', GC(my_seq), '%' )

# Transcription and translation
my_mRNA = my_seq.transcribe()
print( my_mRNA )
my_pept = my_seq.translate()
print( my_pept )

# Complement and reverse complement
print( 'sequence: ',str(my_seq) )
print( 'comp.    : ',my_seq.complement() )
print( 'revcomp : ',my_seq.reverse_complement() )
```

```
> python3 bioseq2.py
AGTACACTGGTA
# A:  4
# C:  2
# G:  3
# T:  3
GC content:  41.666666666666664 %
AGUACACUGGUA
STLV
sequence:  AGTACACTGGTA
comp.    :  TCATGTGACCAT
revcomp :  TACCAGTGTACT
```

Translation II

bioseq3.py

```
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC

messenger_rna = Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG", IUPAC.unambiguous_rna)
print( messenger_rna )
print( messenger_rna.translate() )

# Now, you may want to translate the nucleotides up to the first in frame stop codon,
# and then stop (as happens in nature):
print( messenger_rna.translate(to_stop=True) )
```

```
> python3 bioseq3.py
AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG
MAIVMGR*KGAR*
MAIVMGR
```

The Sequence record object

- The **SeqRecord** objects are the basic data type for the SeqIO objects and they are very similar to **Seq** objects,however, there are a few **additional attributes**.
- **seq** - The sequence itself, typically a Seq object.
- **id** - The primary ID used to identify the sequence – a string. In most cases this is something like an accession number.
- **name** - A ‘common’ name/id for the sequence – a string. In some cases this will be the same as the accession number, but it could also be a clone name. Analagous to the LOCUS id in a GenBank record.
- **description** - A human readable description or expressive name for the sequence – a string.

We can think of the SeqRecord as a container that has the above attributes including the Seq.

SeqRecord -Example

seqrecord.py

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord

## create a simple SeqRecord object
simple_seq = Seq( "GATCAGGATTAGGCC" )
simple_seq_r = SeqRecord( simple_seq )
simple_seq_r.id = "AC12345"
simple_seq_r.description = "I am not a real sequence"

## print summary
print( simple_seq_r.id )
print( simple_seq_r.description )
print( str(simple_seq_r.seq) )
print( simple_seq_r.seq.alphabet )

## translate the sequence
translated_seq = simple_seq_r.seq.translate()
print ( translated_seq )
```

```
> python3 seqrecord.py
AC12345
I am not a real sequence
GATCAGGATTAGGCC
Alphabet()
DQD*A
```

```
Modify the script:
# exercise 1 -- translate the sequence only until the stop codon
# exercise 2 -- get the reverse complement of the sequence
# exercise 3 -- get the reverse of the sequence (just like for
lists)
# exercise 4 -- get the GC nucleotide content
```

The Sequence IO object

seqio.py

```python
import os
from Bio import SeqIO

## save the sequence records to a list
allSeqRecords = []
AllSeqIDs     = []

# fasta file is in current directory
pathToFile = os.path.join("./","multi_fasta_gb_header.fasta")
for seq_record in SeqIO.parse(pathToFile, "fasta"):
    allSeqRecords.append(seq_record)
    allSeqIDs.append(seq_record.id.split("|")[1])
    print( seq_record.id )
    # print 1st 10 nucleotides only
    print( str(seq_record.seq[:10]) )
    print( len(seq_record) )

## print out fun stuff about the sequences
print( "We found ", len(allSeqIDs), "sequences" )
print( "information on the third sequence:" )
ind = 2
seqRec = allSeqRecords[ind]
print( "\t", "GI number    ", allSeqIDs[ind] )
print( "\t", "full id      ", seqRec.id )
print( "\t", "num nucleo.  ", len(seqRec.seq) )
print( "\t", "1st 10 nucleo.", seqRec.seq[:10] )
```

```
...
gi|2765564|emb|Z78439.1|PBZ78439
CATTGTTGAG
592
We found  94 sequences
information on the third sequence:
       GI number      2765656
       full id        gi|2765656|emb|Z78531.1|CFZ78531
       num nucleo.    748
       1st 10 nucleo. CGTAACAAGG
```

The SeqIO object is like a container for multiple SeqRecord objects.

SeqIO

- The standard Sequence Input/Output interface for BioPython
- Provides a simple uniform interface to input and output assorted sequence file formats
- Deals with sequences as SeqRecord objects
- There is a sister interface Bio.AlignIO for working directly with sequence alignment files as Alignments

Parsing a FASTA file

```python
# Parse a simple fasta file
from Bio import SeqIO
for seq_record in SeqIO.parse("multi_fasta_gb_header.fasta", "fasta"):
    print( seq_record.id )
    #print( repr(seq_record.seq) )
    print( seq_record.seq )
    print( len(seq_record) )
```

```
> python3 parse_simple_fasta.py
gi|2765658|emb|Z78533.1|CIZ78533
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGGAATAAACGATCGAGTGA
ATCCGGAGGACCGGTGTACTCAGCTCACCGGGGGCATTGCTCCCGTGGTGACCCTGATTTGTTGTTGGGCC
GCCTCGGGAGCGTCCATGGCGGGTTTGAACCTCTAGCCCGGCGCAGTTTGGGCGCCAAGCCATATGAAAGC
ATCACCGGCGAATGGCATTGTCTTCCCCAAAACCCGGAGCGGCGGCGTGCTGTCGCGTGCCCAATGAATTT
TGATGACTCTCGCAAACGGGAATCTTGGCTCTTTGCATCGGATGGAAGGACGCAGCGAAATGCGATAAGTG
GTGTGAATTGCAAGATCCCGTGAACCATCGAGTCTTTTGAACGCAAGTTGCGCCCGAGGCCATCAGGCTAA
GGGCACGCCTGCTTGGGCGTCGCGCTTCGTCTCTCTCCTGCCAATGCTTGCCCGGCATACAGCCAGGCCGG
CGTGGTGCGGATGTGAAAGATTGGCCCCTTGTGCCTAGGTGCGGCGGGTCCAAGAGCTGGTGTTTTGATGG
CCCGGAACCCGGCAAGAGGTGGACGGATGCTGGCAGCAGCTGCCGTGCGAATCCCCCATGTTGTCGTGCTT
GTCGGACAGGCAGGAGAACCCTTCCGAACCCCAATGGAGGGCGGTTGACCGCCATTCGGATGTGACCCCAG
GTCAGGCGGGGGCACCCGCTGAGTTTACGC
740
...
```

GenBank files

```python
# genbank files
from Bio import SeqIO
for seq_record in SeqIO.parse("sequence.gb", "genbank"):
    print(seq_record)
    #added to print just one record example
    break
```

‘Pelican orchid’

```
> python parse_genbank.py
ID: Z78533.1
Name: Z78533
Description: C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA
Number of features: 5
/molecule_type=DNA
/topology=linear
/data_file_division=PLN
/date=30-NOV-2006
/accessions=['Z78533']
/sequence_version=1
/gi=2765658
/keywords=['5.8S ribosomal RNA', '5.8S rRNA gene', 'internal transcribed spacer', 'ITS1', 'ITS2']
/source=Cypripedium irapeanum
/organism=Cypripedium irapeanum
/taxonomy=['Eukaryota', 'Viridiplantae', 'Streptophyta', 'Embryophyta', 'Tracheophyta', 'Spermatophyta', 'Magnoliophyta', 'Liliopsida',
'Asparagales', 'Orchidaceae', 'Cypripedioideae', 'Cypripedium']
/references=[Reference(title='Phylogenetics of the slipper orchids (Cypripedioideae: Orchidaceae): nuclear rDNA ITS sequences', ...),
Reference(title='Direct Submission', ...)]
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', IUPACAmbiguousDNA())
```

GenBank files

```
# genbank files
from Bio import SeqIO
for seq_record in SeqIO.parse("sequence.gb", "genbank"):
    print(seq_record.id)
    print(repr(seq_record.seq))
    print(len(seq_record))
```

```
> python parse_genbank_2.py
Z78533.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', IUPACAmbiguousDNA())
740
```

Sequence objects

- Support similar methods as standard strings
- Provide additional methods
  - Translate
  - Reverse complement
- Support different alphabets
- AGTAGTTAAA can be
  - DNA
  - Protein

Sequences and alphabets

- Bio.Alphabet.IUPAC provides basic definitions for proteins, DNA and RNA, but additionally provides the ability to extend and customize the basic definitions
- For example:
  - Adding ambiguous symbols
  - Adding special new characters

Example – specific sequences

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> my_seq = Seq("AGTACACTGGT", IUPAC.unambiguous_dna)
>>> my_seq
Seq('AGTACACTGGT', IUPACUnambiguousDNA())
>>> my_seq.alphabet
IUPACUnambiguousDNA()

>>> my_prot = Seq("AGTACACTGGT", IUPAC.protein)
>>> my_prot
Seq('AGTACACTGGT', IUPACProtein())
>>> my_prot.alphabet
IUPACProtein()
```

Sequences act like strings

- Access elements

```
>>> print(my_seq[0]) #first letter
A
>>> print(my_seq[2]) #third letter
T
>>> print(my_seq[-1]) #last letter
T
```

- Count without overlaps

```
>>> from Bio.Seq import Seq
>>> "AAAA".count("AA")
2
>>> Seq("AAAA").count("AA")
2
```

Calculate GC content

https://biopython.org/DIST/docs/api/Bio.SeqUtils-module.html#GC

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> from Bio.SeqUtils import GC
>>> my_seq = Seq('GATCGATGGGCCTATATAGGATCGAAAATCGC', IUPAC.unambiguous_dna)
>>> GC(my_seq)
46.875
```

Slicing

- Simple slicing

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC", IUPAC.unambiguous_dna)
>>> my_seq[4:12]
Seq('GATGGGCC', IUPACUnambiguousDNA())
```

- Start, stop, stride

```
>>> my_seq[0::3]
Seq('GCTGTAGTAAG', IUPACUnambiguousDNA())
>>> my_seq[1::3]
Seq('AGGCATGCATC', IUPACUnambiguousDNA())
>>> my_seq[2::3]
Seq('TAGCTAAGAC', IUPACUnambiguousDNA())
```

Concatenation

- Simple addition as in Python
- But, alphabets must fit!!!

```
>>> from Bio.Alphabet import IUPAC
>>> from Bio.Seq import Seq
>>> protein_seq = Seq("EVRNAK", IUPAC.protein)
>>> dna_seq = Seq("ACGT", IUPAC.unambiguous_dna)
>>> protein_seq + dna_seq
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/pieterdb/anaconda3/lib/python3.7/site-packages/Bio/Seq.py", line
354, in __add__
    self.alphabet, other.alphabet))
TypeError: Incompatible alphabets IUPACProtein() and IUPACUnambiguousDNA()
```

Changing case

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import generic_dna
>>> dna_seq = Seq("acgtACGT", generic_dna)
>>> dna_seq
Seq('acgtACGT', DNAAlphabet())
>>> dna_seq.upper()
Seq('ACGTACGT', DNAAlphabet())
>>> dna_seq.lower()
Seq('acgtacgt', DNAAlphabet())
```

Changing case

- Case is important for matching

```
>>> "GTAC" in dna_seq
False
>>> "GTAC" in dna_seq.upper()
True
```

- IUPAC names are upper case

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> dna_seq = Seq("ACGT", IUPAC.unambiguous_dna)
>>> dna_seq
Seq('ACGT', IUPACUnambiguousDNA())
>>> dna_seq.lower()
Seq('acgt', DNAAlphabet())
```

Reverse complement

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC",
IUPAC.unambiguous_dna)
>>> my_seq.complement()
Seq('CTAGCTACCCGGATATATCCTAGCTTTTAGCG', IUPACUnambiguousDNA())
>>> my_seq.reverse_complement()
Seq('GCGATTTTCGATCCTATATAGGCCCATCGATC', IUPACUnambiguousDNA())
```

Transcription

```
>>> coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG", IUPAC.unambiguous_dna)
>>> template_dna = coding_dna.reverse_complement()
>>> template_dna
Seq('CTATCGGGCACCCTTTCAGCGGCCCATTACAATGGCCAT', IUPACUnambiguousDNA())
>>> messenger_rna = coding_dna.transcribe()
>>> messenger_rna
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPACUnambiguousRNA())
```

As you can see, all this does is switch T → U, and adjust the alphabet.

Translation

Simple example

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> messenger_rna = Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG", IUPAC.unambiguous_rna)
>>> messenger_rna
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPACUnambiguousRNA())
>>> messenger_rna.translate()
Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
```

Stop codon!

Translation from the DNA

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG", IUPAC.unambiguous_dna)
>>> coding_dna
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG', IUPACUnambiguousDNA())
>>> coding_dna.translate()
Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
```

Using different translation tables

- In several cases we may want to use different translation tables
- Translation tables are given IDs in GenBank (standard=1)
- Vertebrate Mitochondrial is table 2

Differences from the Standard Code:

	Code 2		Standard	
AGA	Ter	*	Arg	R
AGG	Ter	*	Arg	R
AUA	Met	M	Ile	I
UGA	Trp	W	Ter	*

- More details in http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi

Using different translation tables

```
>>> coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG", IUPAC.unambiguous_dna)
>>> coding_dna
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG', IUPACUnambiguousDNA())
>>> coding_dna.translate()
Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
>>> coding_dna.translate(table="Vertebrate Mitochondrial")
Seq('MAIVMGRWKGAR*', HasStopCodon(IUPACProtein(), '*'))
>>> coding_dna.translate(table=2)
Seq('MAIVMGRWKGAR*', HasStopCodon(IUPACProtein(), '*'))
```

Translation tables in biopython

```
>>> from Bio.Data import CodonTable
>>> standard_table = CodonTable.unambiguous_dna_by_id[1]
>>> mito_table = CodonTable.unambiguous_dna_by_id[2]
>>> print(standard_table)
Table 1 Standard, SGC0

  |  T      |  C      |  A      |  G      |
--+---------+---------+---------+---------+--
T | TTT F   | TCT S   | TAT Y   | TGT C   | T
T | TTC F   | TCC S   | TAC Y   | TGC C   | C
T | TTA L   | TCA S   | TAA Stop| TGA Stop| A
T | TTG L(s)| TCG S   | TAG Stop| TGG W   | G
--+---------+---------+---------+---------+--
C | CTT L   | CCT P   | CAT H   | CGT R   | T
C | CTC L   | CCC P   | CAC H   | CGC R   | C
C | CTA L   | CCA P   | CAA Q   | CGA R   | A
C | CTG L(s)| CCG P   | CAG Q   | CGG R   | G
--+---------+---------+---------+---------+--
A | ATT I   | ACT T   | AAT N   | AGT S   | T
A | ATC I   | ACC T   | AAC N   | AGC S   | C
A | ATA I   | ACA T   | AAA K   | AGA R   | A
A | ATG M(s)| ACG T   | AAG K   | AGG R   | G
--+---------+---------+---------+---------+--
G | GTT V   | GCT A   | GAT D   | GGT G   | T
G | GTC V   | GCC A   | GAC D   | GGC G   | C
G | GTA V   | GCA A   | GAA E   | GGA G   | A
G | GTG V   | GCG A   | GAG E   | GGG G   | G
--+---------+---------+---------+---------+--
```

Translate up to the first stop in frame

```
>>> coding_dna.translate()
Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
>>> coding_dna.translate(to_stop=True)
Seq('MAIVMGR', IUPACProtein())
>>> coding_dna.translate(table = 2)
Seq('MAIVMGRWKGAR*', HasStopCodon(IUPACProtein(), '*'))
>>> coding_dna.translate(table = 2, to_stop = True)
Seq('MAIVMGRWKGAR', IUPACProtein())
```

Comparing sequences

Standard “==“ comparison is done by comparing the references, hence:

```
>>> seq1 = Seq("ACGT", IUPAC.unambiguous_dna)
>>> seq2 = Seq("ACGT", IUPAC.unambiguous_dna)
>>> seq1 == seq2
True
```

Mutable vs. Immutable

- Like strings standard seq objects are **immutable**
- If you want to create a mutable object you need to write it by either:
  - Use the “tomutable()” method
  - Use the mutable constructor

```
>>> from Bio.Seq import MutableSeq
>>> from Bio.Alphabet import generic_dna
>>> my_seq = MutableSeq("ACTCGTCGTCG", generic_dna)
>>> my_seq
MutableSeq('ACTCGTCGTCG', DNAAlphabet())
>>> my_seq[5]
'T'
>>> my_seq[5] = "A"
>>> my_seq
MutableSeq('ACTCGACGTCG', DNAAlphabet())
>>> my_seq[5]
'A'
>>> my_seq[5:8] = "NNN"
>>> my_seq
MutableSeq('ACTCGNNNTCG', DNAAlphabet())
>>> len(my_seq)
11
```

Unknown sequences example

In many biological cases we deal with unknown sequences:

```
>>> from Bio.Seq import UnknownSeq
>>> from Bio.Alphabet import IUPAC
>>> unk_dna = UnknownSeq(20, alphabet=IUPAC.ambiguous_dna)
>>> my_seq = Seq("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA", IUPAC.unambiguous_dna)
>>> unk_dna+my_seq
Seq('NNNNNNNNNNNNNNNNNNNNGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA', IUPACAmbiguousDNA())
```

MSA

Read MSA

- Use Bio.AlignIO.read(file, format)
- File – the file path
- Format support:
  - “stockholm”
  - “fasta”
  - “clustal”
  - …
- Use help(AlignIO) for details

Example

We want to parse files like this from PFAM

```
# STOCKHOLM 1.0
#=GF ID   Phage_Coat_Gp8
#=GF AC   PF05371.12
#=GF DE   Phage major coat protein, Gp8
#=GF AU   Finn RD;0000-0001-8626-2148
#=GF SE   Pfam-B_31655 (release 7.8)
#=GF GA   22.30 22.30;
#=GF TC   22.50 24.90;
#=GF NC   22.20 22.20;
#=GF BM   hmmbuild HMM.ann SEED.ann
#=GF SM   hmmsearch -Z 45638612 -E 1000 --cpu 4 HMM pfamseq
#=GF TP   Family
…
#=GS A0A291JUA2_9GAMM/34-84  AC A0A291JUA2.1
A0A1J0VHQ7_9GAMM/29-82                ....................agtanat----EP-TAAEAAFTALQSEADAMAGYAWPVVAGIVGSLLAIGLFKKFANKA...
A0A085DSG1_9GAMM/1-44                 ..........................a---------ASAAFDAVAEQGTEMAGYAWPVVGAITASLIGIGLFKKFANKA...
A0A0P9AAU4_9GAMM/24-72                ...........................AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA...
K8W5V2_9GAMM/28-78                    .....................aessip-------QPAQDAITSIGETATGMIDLAWPVIALVVGGFLAIKLFKKVSNK-v..
CAPSD_BPIF1/22-73                     ..........................f-AADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA...
#=GR CAPSD_BPIF1/22-73         SS     ..........................X-X-HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH...
G0AGW2_COLFT/20-80                    .........aalvatgaanaaatldps---------VAAAFTAIGDNATAMFGLAMPVVASVLGMFIVLRLFKKFGNKA...
CAPSD_BPIKE/30-81                     ...........................AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA...
#=GR CAPSD_BPIKE/30-81         SS     ...........................-HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH...
G0A214_METMM/20-72                    ...................sganaalp-------AAATTAFTDLQTDALSLVDLAWTAAIPITIAFIILRMFKRAASSA...
N6Z3H0_9RHOO/17-88                    adrarakqaaaaaalsvaamsaqaelp-------AAATAAITAVKEDGTDLVAAGWPVLVAITGGLILMGLFKKVLSRA...
A0A1I3UID1_9ALTE/3-51                 ....................ttlppev----------TAAFTGISDNFDAMAAEAWPVVATIVGGFILLKLFKKFANKA...
A0A085DVW1_9GAMM/10-63                ...................asaaahaq------EAAASAAFDAVAEQGTEMAGYAWPVVGAITASLIGIGLFKKFANKA...
A0A1S8Y744_9ENTR/27-78                ........................age---GASTDYAGQAMDSLLTQANDLIAKVWPVVVAVVGAGLAIRLFKKFSSKA...
A0A1U9RGB5_9GAMM/30-79                ........................ign-----AQADASAAFSEIQSTGADMAGQAWPVVAAITASLIGIKLFKKFANRA...
CAPSD_BPM13/24-72                     ...........................AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA...
#=GR CAPSD_BPM13/24-72         SS     ...........................-TTS-H---HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHT...
A0A090R4K9_9VIBR/20-68                ......................aalpa--------EAQAAMDELGTFATDMVAAAWVIVPIVVVGFIGIKLFKKAQ---tkq
A0A0P9AYN8_9PROT/24-72                ...........................AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA...
CAPSD_BPI22/32-83                     ..........................d-GTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA...
A0A2A3XI27_9GAMM/32-82                .......................vhaq----EA-SGAAAAFDEVSASGAEMAGYAWPVVASITAALIGIKLFKKFANRA...
A0A1E5CL89_9VIBR/22-76                ................asgsanaalpa--------HVTDAFTAVGTLVTDLEAQAWIIVPVVFIALAGITLFKKFGNKA...
A0A291JUA2_9GAMM/34-84                .....................ahaqea-------PGASAAFSAISSQASDFSGDAWPVVIGVTSALVGIKLFKKFISRA...
#=GC SS_cons                          ...........................-CCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH...
#=GC seq_cons                         ..........................s.........AsAAFsulpspAT-hhuhAWPVVsslsuuhIuI+LFKKFusKA...
//
```

https://pfam.xfam.org/family/PF05371/alignment/full

Example

```
>>> from Bio import AlignIO
>>> alignment = AlignIO.read("PF05371.sth", "stockholm")
>>> print(alignment)
SingleLetterAlphabet() alignment with 20 rows and 82 columns
--------------------agtanat----EP-TAAEAAFTAL...--- A0A1J0VHQ7_9GAMM/29-82
--------------------------a---------ASAAFDAV...--- A0A085DSG1_9GAMM/1-44
---------------------------AEGDDP---AKAAFNSL...--- A0A0P9AAU4_9GAMM/24-72
---------------------aessip-------QPAQDAITSI...v-- K8W5V2_9GAMM/28-78
--------------------------f-AADDATSQAKAAFDSL...--- CAPSD_BPIF1/22-73
---------aalvatgaanaaatldps---------VAAAFTAI...--- G0AGW2_COLFT/20-80
---------------------------AEPNAATNYATEAMDSL...--- CAPSD_BPIKE/30-81
-------------------sganaalp-------AAATTAFTDL...--- G0A214_METMM/20-72
adrarakqaaaaaalsvaamsaqaelp-------AAATAAITAV...--- N6Z3H0_9RHOO/17-88
--------------------ttlppev----------TAAFTGI...--- A0A1I3UID1_9ALTE/3-51
-------------------asaaahaq------EAAASAAFDAV...--- A0A085DVW1_9GAMM/10-63
------------------------age---GASTDYAGQAMDSL...--- A0A1S8Y744_9ENTR/27-78
------------------------ign-----AQADASAAFSEI...--- A0A1U9RGB5_9GAMM/30-79
---------------------------AEGDDP---AKAAFNSL...--- CAPSD_BPM13/24-72
----------------------aalpa--------EAQAAMDEL...tkq A0A090R4K9_9VIBR/20-68
---------------------------AEGDDP---AKAAFNSL...--- A0A0P9AYN8_9PROT/24-72
--------------------------d-GTSTATSYATEAMNSL...--- CAPSD_BPI22/32-83
-----------------------vhaq----EA-SGAAAAFDEV...--- A0A2A3XI27_9GAMM/32-82
----------------asgsanaalpa--------HVTDAFTAV...--- A0A1E5CL89_9VIBR/22-76
---------------------ahaqea-------PGASAAFSAI...--- A0A291JUA2_9GAMM/34-84
```

Alignment object example

```
>>> from Bio import AlignIO
>>> alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
>>> print(alignment[1])
ID: J7I0P6_BPM13/24-72
Name: J7I0P6_BPM13
Description: J7I0P6_BPM13/24-72
Number of features: 0
/accession=J7I0P6.1
/start=24
/end=72
Seq('AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA',
SingleLetterAlphabet())
```

Alignment object example

```
>>> print("Alignment length %i" % alignment.get_alignment_length())
Alignment length 52
>>> for record in alignment:
...     print("%s - %s" % (record.seq, record.id))
...
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA - Q9T0Q9_BPFD/1-49
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA - J7I0P6_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA - CAPSD_BPZJ2/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA - CAPSD_BPIF1/22-73
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA - Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA - CAPSD_BPI22/32-83
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA - CAPSD_BPIKE/30-81
```

Cross-references example

Did you notice in the raw file above that several of the sequences include database cross-references to the PDB and the associated known secondary structure?

```
>>> for record in alignment:
...     if record.dbxrefs:
...         print(record.id, record.dbxrefs)
...
Q9T0Q9_BPFD/1-49 ['PDB; 1NH4 A; 6-49;']
CAPSD_BPIF1/22-73 ['PDB; 1IFK A; 1-50;']
CAPSD_BPIKE/30-81 ['PDB; 1IFL A; 1-52;']
```

Comments

- Remember that almost all MSA formats are supported
- When you have more than one MSA in your files use AlignIO.parse()
  - Common example is PHYLIP’s output
    - Use AlignIO.parse("resampled.phy", "phylip")
    - The result is an iterator object that contains all MSAs

Write alignment to file

alignment2file.py

```
from Bio.Alphabet import generic_dna
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio.Align import MultipleSeqAlignment

align1 = MultipleSeqAlignment([
    SeqRecord(Seq("ACTGCTAGCTAG", generic_dna), id="Alpha"),
    SeqRecord(Seq("ACT-CTAGCTAG", generic_dna), id="Beta"),
    SeqRecord(Seq("ACTGCTAGDTAG", generic_dna), id="Gamma"), ])

from Bio import AlignIO

AlignIO.write(align1, "my_example.phy", "phylip")
```

```
 3 12
Alpha      ACTGCTAGCT AG
Beta       ACT-CTAGCT AG
Gamma      ACTGCTAGDT AG
```

my_example.phy

Slicing

Alignments work like numpy matrices

```
>>> print(alignment)
SingleLetterAlphabet() alignment with 7 rows and 52 columns
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA Q9T0Q9_BPFD/1-49
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA J7I0P6_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA CAPSD_BPZJ2/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKL...SRA CAPSD_BPIF1/22-73
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKL...SRA Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRL...SKA CAPSD_BPI22/32-83
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRL...SKA CAPSD_BPIKE/30-81
>>> print(alignment[2,6])
-
>>> # You can pull out a single column as a string like this:
...
>>> print(alignment[:,6])
---TTTT
>>> print(alignment[3:6,:6])
SingleLetterAlphabet() alignment with 3 rows and 6 columns
FAADDA CAPSD_BPIF1/22-73
AEPNAA Q9T0Q8_BPIKE/1-52
DGTSTA CAPSD_BPI22/32-83
>>> print(alignment[:,:6])
SingleLetterAlphabet() alignment with 7 rows and 6 columns
AEGDDP Q9T0Q9_BPFD/1-49
AEGDDP J7I0P6_BPM13/24-72
AEGDDP CAPSD_BPZJ2/1-49
FAADDA CAPSD_BPIF1/22-73
AEPNAA Q9T0Q8_BPIKE/1-52
DGTSTA CAPSD_BPI22/32-83
AEPNAA CAPSD_BPIKE/30-81
```

External applications

- How do we call MSA algorithms on unaligned set of sequences?
- Biopython provides wrappers
- The idea:
  - Create a command line object with the algorithm options
  - Invoke the command (Python uses subprocesses)
- Bio.Align.Applications module:
  - >>> import Bio.Align.Applications
  - >>> dir(Bio.Align.Applications)
    - ['ClustalwCommandline', 'DialignCommandline', 'MafftCommandline', 'MuscleCommandline', 'PrankCommandline', 'ProbconsCommandline', 'TCoffeeCommandline']

ClustalW example

- First step: download ClustalW from ftp://ftp.ebi.ac.uk/pub/software/clustalw2/2.1/
- Second step: install
- Third step: look for clustal exe files
- Now you can run ClustalW from your Python code

Run example

```
>>> import os
>>> from Bio.Align.Applications import ClustalwCommandline
>>> clustalw_exe = r"/usr/local/bin/clustalw2"
>>> clustalw_cline = ClustalwCommandline(clustalw_exe, infile="BRCA2_multi.fa")
>>> assert os.path.isfile(clustalw_exe), "Clustal W executable missing"
>>> stdout, stderr = clustalw_cline()
```

The command line is actually a function we can run!

ClustalW

```
>>> from Bio import AlignIO
>>> align = AlignIO.read("msa_example.aln", "clustal")
>>> print(align)
SingleLetterAlphabet() alignment with 3 rows and 58 columns
CTGTCTCCTGCCGACAAGACCAACGTCAAGGCCGCCTGGGGTAA...--- test2
-------------ACAAAAGCAACATCAAGGCTGCCTGGGGGAA...ATG test3
------ATGAGTCTCTCTGATAAGGACAAGGCTGCT--GTGAAA...--- test1
```

ClustalW - tree

In case you are interested, the opuntia.dnd file ClustalW creates is just a standard Newick tree file, and Bio.Phylo can parse these:

```
>>> from Bio import Phylo
>>> tree = Phylo.read("msa_example.dnd", "newick")
>>> Phylo.draw_ascii(tree)
  _______________________________________________________________________ test1
 |
_|______ test2
 |
 |________________________________________________________ test3
```

BLAST

Running BLAST over the internet

- We use the function qblast() in the Bio.Blast.NCBIWWW module. This has three non-optional arguments:
  - The blast program to use for the search, as a lower case string: works with blastn, blastp, blastx, tblast and tblastx.
  - The databases to search against. The options for this are available on the NCBI web pages at http://www.ncbi.nlm.nih.gov/BLAST/blast_databases.shtml.
  - A string containing your query sequence. This can either be the sequence itself, the sequence in fasta format, or an identifier like a GI number.

qblast additional parameters

- qblast can receive other parameters, analogous to the parameters of the actual server
- Important examples:
  - **format_type:** "HTML", "Text", "ASN.1", or "XML". The default is "XML", as that is the format expected by the parser (see next examples)
  - **expect** sets the expectation or e-value threshold.

Step 1: call BLAST

```
>>> from Bio.Blast import NCBIWWW
>>> # Option 1 - Use GI ID
...
>>> result_handle = NCBIWWW.qblast("blastn", "nt", "8332116")
>>> # Option 2 – read a fasta file
...
>>> fasta_string = open("insulin.fasta").read()
>>> result_handle = NCBIWWW.qblast("blastn", "nt", fasta_string)
>>> # option 3 – parse file to seq object
...
>>> from Bio import SeqIO
>>> record = SeqIO.read(open("insulin.fasta"), format="fasta")
>>> result_handle = NCBIWWW.qblast("blastn", "nt", record.seq)
```

Step2: parse the results

```
>>> from Bio.Blast import NCBIXML
>>> blast_record = NCBIXML.read(result_handle)
```

- Read can be used only once!
- blast_record object keeps the actual results

Remarks

- Basically, Biopython supports reading BLAST results from HTMLs and text files.
- These methods are not stable and sometimes fail because the servers change the format.
- XML is stable
  - You can save XML files
    - In the server
    - From result_handle objects (next slide)

Save results as XML

```python
from Bio.Blast import NCBIWWW
from Bio import SearchIO
from Bio import SeqIO

base_dir = "./"
inputs = ['insulin']

for input in inputs:
    blast_xml = base_dir + input + '.xml'
    blast_out = base_dir + input + '.fa'

    # run BLAST
    fasta_string = open(base_dir + input + '.fasta').read()
    with NCBIWWW.qblast("blastn", "nr", fasta_string) as result_handle:
        with open(blast_xml, 'w') as xml_file:
            xml_file.write(result_handle.read())

    # parse xml and write to fasta
    blast_qresult = SearchIO.read(blast_xml, 'blast-xml')
    records = []
    for hit in blast_qresult:
        records.append(hit[0].hit)
    SeqIO.write(records, blast_out, "fasta")
```

```
> python blast2xml.py
```

Silent generation of result files in the current folder:

```
-rw-rw-r-- 1 pieterdb pieterdb  42288 Oct 21 10:41 insulin.fa
-rw-rw-r-- 1 pieterdb pieterdb 178246 Oct 21 10:41 insulin.xml
-rw-rw-r-- 1 pieterdb pieterdb   1531 Oct 18 17:02 insulin.fasta
```

Parsing BLAST Results

- Example:

```
from Bio.Blast import NCBIXML
# open the file directly using python open method and use NCBIXML parse method

E_VALUE_THRESH = 1e-20
for record in NCBIXML.parse(open("insulin.xml")):
    if record.alignments:
        print("\n")
        print("query: %s" % record.query[:100])
        for align in record.alignments:
            for hsp in align.hsps:
                if hsp.expect < E_VALUE_THRESH:
                    print("match: %s " % align.title[:100])
```

```
> python parse_blast_results.py

query: NG_007114.1:4986-6416 Homo sapiens insulin (INS), RefSeqGene on chromosome 11
match: gi|1028630736|ref|NG_050578.1| Homo sapiens INS-IGF2 readthrough (INS-IGF2), RefSeqGene on chromosom
match: gi|161086962|ref|NG_007114.1| Homo sapiens insulin (INS), RefSeqGene on chromosome 11
...
```

More functions

- We cover here very basic functions
- To get more details use

```
>>> import Bio.Blast.Record
>>> help(Bio.Blast.Record)
```

Accessing NCBI’s Entrez Databases

Bio.Entrez

- Module for programmatic access to Entrez
- Example: search PubMed or download GenBank records from within a Python script
- Makes use of the Entrez Programming Utilities http://www.ncbi.nlm.nih.gov/entrez/utils/
- Makes sure that the correct URL is used for the queries, and that not more than one request is made every three seconds, as required by NCBI
- Note! If the NCBI finds you are abusing their systems, they can and will ban your access!

ESearch example

```
>>> from Bio import Entrez
>>> handle = Entrez.esearch(db="nucleotide",term="Homo sapiens[Orgn] AND INS[Gene]")
/home/pieterdb/anaconda3/lib/python3.7/site-packages/Bio/Entrez/__init__.py:606: UserWarning:
Email address is not specified.

To make use of NCBI's E-utilities, NCBI requires you to specify your
email address with each request.  As an example, if your email address
is A.N.Other@example.com, you can specify it as follows:
   from Bio import Entrez
   Entrez.email = 'A.N.Other@example.com'
In case of excessive usage of the E-utilities, NCBI will attempt to contact
a user at the email address provided before blocking access to the
E-utilities.
  E-utilities.""", UserWarning)
>>> record = Entrez.read(handle)
>>> print (record["IdList"])
['1028630736', '161086962', '1677529882', '568815587', '1677531262', '1677499294', '297374822',
'333826818', '123999447', '1036032746', '1036031581', '823670711', '823670709', '823670707',
'823670705', '74273671', '71514639', '13528923', '42655577', '389620190']
```

Explanation

- Entrez.read
  - Transforms the actual results (retrieved as XML) to a usable object of type Bio.Entrez.Parser.DictionaryElement

```
>>> record
{'Count': '27', 'RetMax': '20', 'RetStart': '0', 'IdList': ['1028630736', '161086962',
'1677529882', '568815587', '1677531262', '1677499294', '297374822', '333826818',
'123999447', '1036032746', '1036031581', '823670711', '823670709', '823670707',
'823670705', '74273671', '71514639', '13528923', '42655577', '389620190'],
'TranslationSet': [{'From': 'Homo sapiens[Orgn]', 'To': '"Homo sapiens"[Organism]'}],
'TranslationStack': [{'Term': '"Homo sapiens"[Organism]', 'Field': 'Organism',
'Count': '27557489', 'Explode': 'Y'}, {'Term': 'INS[Gene]', 'Field': 'Gene', 'Count':
'678', 'Explode': 'N'}, 'AND'], 'QueryTranslation': '"Homo sapiens"[Organism] AND
INS[Gene]'}
```

Database options

'pubmed', 'protein', 'nucleotide', 'nuccore', 'nucgss', 'nucest', 'structure', 'genome', 'books', 'cancerchromosomes', 'cdd', 'gap', 'domains', 'gene', 'genomeprj', 'gensat', 'geo', 'gds', 'homologene', 'journals', 'mesh', 'ncbisearch', 'nlmcatalog', 'omia', 'omim', 'pmc', 'popset', 'probe', 'proteinclusters', 'pcassay', 'pccompound', 'pcsubstance', 'snp', 'taxonomy', 'toolkit', 'unigene', 'unists'

Download a full record

```
>>> from Bio import Entrez
>>> Entrez.email = "pieterdb@irc.vib-ugent.be"
>>> handle = Entrez.efetch(db="nucleotide", id="333826818", rettype="gb", retmode="text")
>>> print(handle.read())
LOCUS       JF909299                 285 bp    mRNA    linear   PRI 25-JUL-2016
DEFINITION  Homo sapiens insulin (INS) mRNA, partial cds.
ACCESSION   JF909299
VERSION     JF909299.1
KEYWORDS    .
…
…
     misc_feature    223..>285
                     /gene="INS"
                     /note="insulin A chain"
ORIGIN
        1 ctggggacct gacccagccg cagcctttgt gaaccaacac ctgtgcggct cacacctggt
       61 ggaagctctc tacctagtgt gcggggaacg aggcttcttc tacacaccca agacccgccg
      121 ggaggcagag gacctgcagg tggggcaggt ggagctgggc gggggccctg gtgcaggcag
      181 cctgcagccc ttggccctgg aggggtccct gcagaagcgt ggcattgtgg aacaatgctg
      241 taccagcatc tgctccctct accagctgga gaactactgc aacta
//
```

Change ‘gb’ to ‘fasta’

```
>>> from Bio import Entrez
>>> Entrez.email = "pieterdb@irc.vib-ugent.be"
>>> handle = Entrez.efetch(db="nucleotide", id="333826818", rettype="fasta", retmode="text")
>>> print(handle.read())
>JF909299.1 Homo sapiens insulin (INS) mRNA, partial cds
CTGGGGACCTGACCCAGCCGCAGCCTTTGTGAACCAACACCTGTGCGGCTCACACCTGGTGGAAGCTCTC
TACCTAGTGTGCGGGGAACGAGGCTTCTTCTACACACCCAAGACCCGCCGGGAGGCAGAGGACCTGCAGG
TGGGGCAGGTGGAGCTGGGCGGGGGCCCTGGTGCAGGCAGCCTGCAGCCCTTGGCCCTGGAGGGGTCCCT
GCAGAAGCGTGGCATTGTGGAACAATGCTGTACCAGCATCTGCTCCCTCTACCAGCTGGAGAACTACTGC
AACTA
```

Read directly to Seq.IO object

```
>>> from Bio import Entrez, SeqIO
>>> handle = Entrez.efetch(db="nucleotide", id="333826818", rettype="gb", retmode="text")
>>> record = SeqIO.read(handle, "genbank")
>>> handle.close()
>>> print(record)
ID: JF909299.1
Name: JF909299
Description: Homo sapiens insulin (INS) mRNA, partial cds
Number of features: 7
/molecule_type=mRNA
/topology=linear
/data_file_division=PRI
/date=25-JUL-2016
/accessions=['JF909299']
/sequence_version=1
/keywords=['']
/source=Homo sapiens (human)
/organism=Homo sapiens
/taxonomy=['Eukaryota', 'Metazoa', 'Chordata', 'Craniata', 'Vertebrata', 'Euteleostomi', 'Mammalia',
'Eutheria', 'Euarchontoglires', 'Primates', 'Haplorrhini', 'Catarrhini', 'Hominidae', 'Homo']
/references=[Reference(title='Preparation of human insulin gene construct in order to use in diabetes
mellitus gene therapy', ...), Reference(title='Direct Submission', ...)]
Seq('CTGGGGACCTGACCCAGCCGCAGCCTTTGTGAACCAACACCTGTGCGGCTCACA...CTA', IUPACAmbiguousDNA())
```

Download directly from a URL

- Suppose we know how the database URLs look like
- Example: GEO (gene expression omnibus):

"`http://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE6609&format=file`"

Use the urlib.request module

```
>>> import urllib.request
>>> url='http://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE6609&format=file'
>>> with urllib.request.urlopen(url) as response, open('gse_6609_raw.tar', 'wb') as out_file:
...     data = response.read() # a `bytes` object
...     out_file.write(data)
...
14458880
```

More details

We covered only a few concepts
For more details on Biopython options, including dealing with specialized parsers, see:

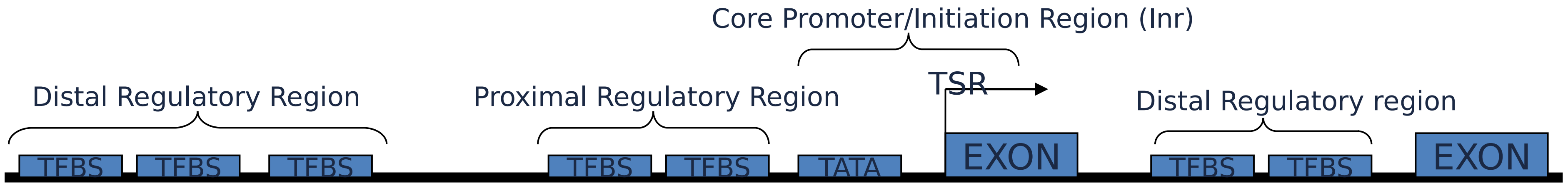
http://biopython.org/DIST/docs/tutorial/Tutorial.html#sec:parsing-blast

Look at the urllib.request manual:

https://docs.python.org/3/library/urllib.request.html

Sequence Motifs

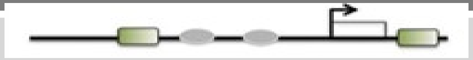
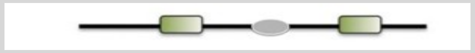
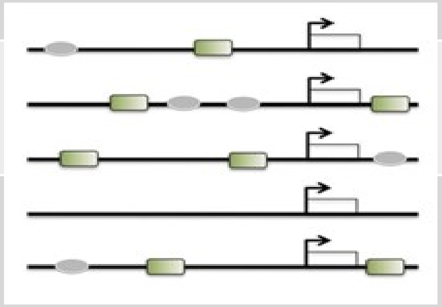
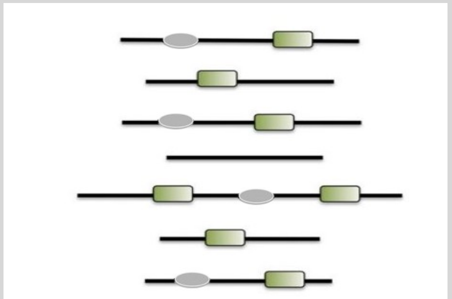
Anatomy of Transcriptional Regulation



- **Core Promoter** – Sufficient for initiation of transcription; orientation dependent (i.e. immediately upstream, functions in only one orientation)
  - **TSR** – transcription start region
    - Refers to a region rather than specific start site (**TSS**)
- **TFBS** – single transcription factor binding site
- **Regulatory Regions**
  - **Proximal/Distal** – vague reference to distance from TSR
  - May be **positive (enhancing)** or **negative (repressing)**
  - Orientation independent and location independent (generally)
  - **Modules** – Sets of TFBS within a region that function together

Ackn.: R. Bruskiewich and F. Brinkman, MBB with material from Wyeth W. Wasserman and Shannan Ho Sui

Different analyses types

Type of data	Biological questions (not limited)	
Single gene	1. I want to scan the promoter of this gene with one or more motif(s). 2. Which TFBSs are conserved in the promoters of orthologues?	 
Single sequence	1. I want to scan this sequence with one or more motif(s).	
List of gene names	1. Are there over-represented motifs in the promoters of these genes? 2. I want to scan the promoters of these genes with one or more motif(s). 3. Which TFBSs are conserved in the promoters of orthologues?	

Pattern Matching versus Pattern Discovery

Pattern Matching:

Finding known motifs

- Does protein X binds to the regulatory regions of my genes?
- Does it bind more than expected by chance?

Pattern Discovery:

Finding unknown motifs

- What motifs are present in the regulatory regions of my genes?
- What are these motifs?

Transcription Factor Binding Sites

- Regulatory **motifs** are often degenerate, variable but similar.
- Transcription factors are often pleiotropic, regulating several genes, but they may need to be expressed at different levels.
- A side effect of this degeneracy is spurious binding, where the protein has affinity at positions in the genome other than their functional sites.
- Non-specific binding competes for protein and requires more protein to be produced than would be required otherwise

Set of binding sites

```
AAGTTAATGA
CAGTTAATAA
GAGTTAAACA
CAGTTAATTA
GAGTTAATAA
CAGTTATTCA
GAGTTAATAA
CAGTTAATCA
AGATTAAAGA
AAGTTAACGA
AGGTTAACGA
ATGTTGATGA
AAGTTAATGA
AAGTTAACGA
AAATTAATGA
GAGTTAATGA
AAGTTAATCA
AAGTTGATGA
AAATTAATGA
ATGTTAATGA
AAGTAAATGA
AAGTTAATGA
AAGTTAATGA
AAATTAATGA
AAGTTAATGA
AAGTTAATGA
AAGTTAATGA
AAGTTAATGA
```

Transcription Factor Binding Sites

From alignment to motif

- Motif descriptors:
  - String-based
    - Strict consensus
    - Degenerate consensus
    - Regular expressions
  - Matrix-based
    - Position-specific scoring matrices (PSSMs)
    - Sequence Logos
    - Hidden Markov Models (HMM)
  - ML-based classifiers

```
  Set of
 binding
  sites
AAGTTAATGA
CAGTTAATAA
GAGTTAAACA
CAGTTAATTA
GAGTTAATAA
CAGTTATTCA
GAGTTAATAA
CAGTTAATCA
AGATTAAAGA
AAGTTAACGA
AGGTTAACGA
ATGTTGATGA
AAGTTAATGA
AAGTTAACGA
AAATTAATGA
GAGTTAATGA
AAGTTAATCA
AAGTTGATGA
AAATTAATGA
ATGTTAATGA
AAGTAAATGA
AAGTTAATGA
AAGTTAATGA
AAATTAATGA
AAGTTAATGA
AAGTTAATGA
AAGTTAATGA
AAGTTAATGA
```

Motif Descriptors – Consensus Sequence

A set of sites represented as a **consensus**

- derived from the collection of binding sites by taking the predominant letter at each positon of the motif:
  - **Strict consensus**: only ACGT alphabet
  - **Degenerate consensus**: IUPAC code for ambiguous nucleotides

- **IUPAC** (International Union of Pure and Applied Chemistry) nomenclature
  - A way to represent ambiguity when more than one value is possible at a particular position

```
   Set of
   binding
    sites
AAGTTAATGA
CAGTTAATAA
GAGTTAAACA
CAGTTAATTA
GAGTTAATAA
CAGTTATTCA
GAGTTAATAA
CAGTTAATCA
AGATTAAAGA
AAGTTAACGA
AGGTTAACGA
ATGTTGATGA
AAGTTAATGA
AAGTTAACGA
AAATTAATGA
GAGTTAATGA
AAGTTAATCA
AAGTTGATGA
AAATTAATGA
ATGTTAATGA
AAGTAAATGA
AAGTTAATGA
AAGTTAATGA
AAATTAATGA
AAGTTAATGA
AAGTTAATGA
AAGTTAATGA
AAGTTAATGA
```

Ackn.: R. Bruskiewich and F. Brinkman, MBB with material from Wyeth W. Wasserman and Shannan Ho Sui

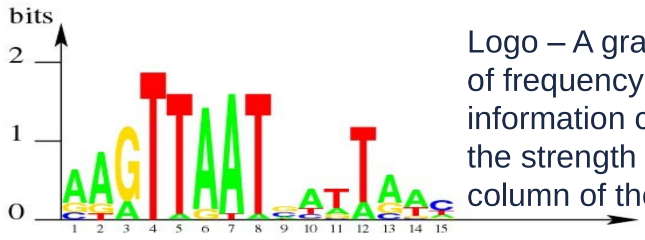
Consensus Sequence - Properties

- **Pro**: simple and synthetic representation
- **Contra**:
  - Strict consensus: loss of information about non-predominant letters
  - Degenerate consensus: loss of information about the most frequent letter

Motif Descriptors – The PSSM*

- *PSSM: Position-Specific Score Matrix
- A position-specific count matrix (PCM) describing a set of sites

A	14	16	4	0	1	19	20	1	4	13	4	4	13	12	3
C	3	0	0	0	0	0	0	0	7	3	1	0	3	1	12
G	4	3	17	0	0	2	0	0	9	1	3	0	5	2	2
T	0	2	0	21	20	0	1	20	1	4	13	17	0	6	4



Logo – A graphical representation of frequency matrix. Y-axis is information content , which reflects the strength of the pattern in each column of the matrix

Make your own logo at: http://weblogo.berkeley.edu/

Set of binding sites

```
AAGTTAATGA
CAGTTAATAA
GAGTTAAACA
CAGTTAATTA
GAGTTAATAA
CAGTTATTCA
GAGTTAATAA
CAGTTAATCA
AGATTAAAGA
AAGTTAACGA
AGGTTAACGA
ATGTTGATGA
AAGTTAATGA
AAGTTAACGA
AAATTAATGA
GAGTTAATGA
AAGTTAATCA
AAGTTGATGA
AAATTAATGA
ATGTTAATGA
AAGTAAATGA
AAGTTAATGA
AAGTTAATGA
AAATTAATGA
AAGTTAATGA
AAGTTAATGA
AAGTTAATGA
AAGTTAATGA
```

Ackn.: R. Bruskiewich and F. Brinkman, MBB with material from Wyeth W. Wasserman and Shannan Ho Sui

From a PCM to a PSSM

Add the following features to the matrix profile:

1. Correct for nucleotide frequencies in genome
2. Weight for the confidence (depth) in the pattern
3. Convert to log-scale probability for easy arithmetic

PCM

$$\begin{array}{c|ccccc} A & 5 & 0 & 1 & 0 & 0 \\ C & 0 & 2 & 2 & 4 & 0 \\ G & 0 & 3 & 1 & 0 & 4 \\ T & 0 & 0 & 1 & 1 & 1 \end{array}$$

$$\xrightarrow{\quad \text{Log}_2\left(\frac{f(b,i) + s(n)}{p(b)}\right) \quad}$$

PSSM

$$\begin{array}{c|ccccc} A & 1.6 & -1.7 & -0.2 & -1.7 & -1.7 \\ C & -1.7 & 0.5 & 0.5 & 1.3 & -1.7 \\ G & -1.7 & 1.0 & -0.2 & -1.7 & 1.3 \\ T & -1.7 & -1.7 & -0.2 & -0.2 & -0.2 \end{array}$$

Assigning a score to a motif: **TGCTG =-1.7+1.0+0.5-0.2+1.3= 0.9**

f(b,i) = frequency of base *b* at position *i*
s(n) = pseudocount correction (optionally applied to small samples of binding sites)
p(b) = background probability of base *b*

Ackn.: R. Bruskiewich and F. Brinkman, MBB with material from Wyeth W. Wasserman and Shannan Ho Sui

PSSM Scoring Scales

- Raw scores
  - Sum of values from indicated cells of the matrix
- Relative Scores (most common)
  - Normalize the scores to range of 0-1 or 0%-100%
- Empirical p-values
  - Based on distribution of scores for some DNA sequence, determine a p-value

Ackn.: R. Bruskiewich and F. Brinkman, MBB with material from Wyeth W. Wasserman and Shannan Ho Sui

Biopython motif objects

```
>>> from Bio import motifs
>>> from Bio.Seq import Seq
>>> instances =
[Seq("TACAA"),Seq("TACGC"),Seq("TACAC"),Seq("TACCC"),Seq("AACCC"),Seq("AATGC"),Seq("AATGC")]
>>> m = motifs.create(instances)
>>> print(m)
TACAA
TACGC
TACAC
TACCC
AACCC
AATGC
AATGC
```

Biopython motif objects

```
>>> print(m.counts)
        0      1      2      3      4
A:   3.00   7.00   0.00   2.00   1.00
C:   0.00   0.00   5.00   2.00   6.00
G:   0.00   0.00   0.00   3.00   0.00
T:   4.00   0.00   2.00   0.00   0.00
```

Biopython motif objects

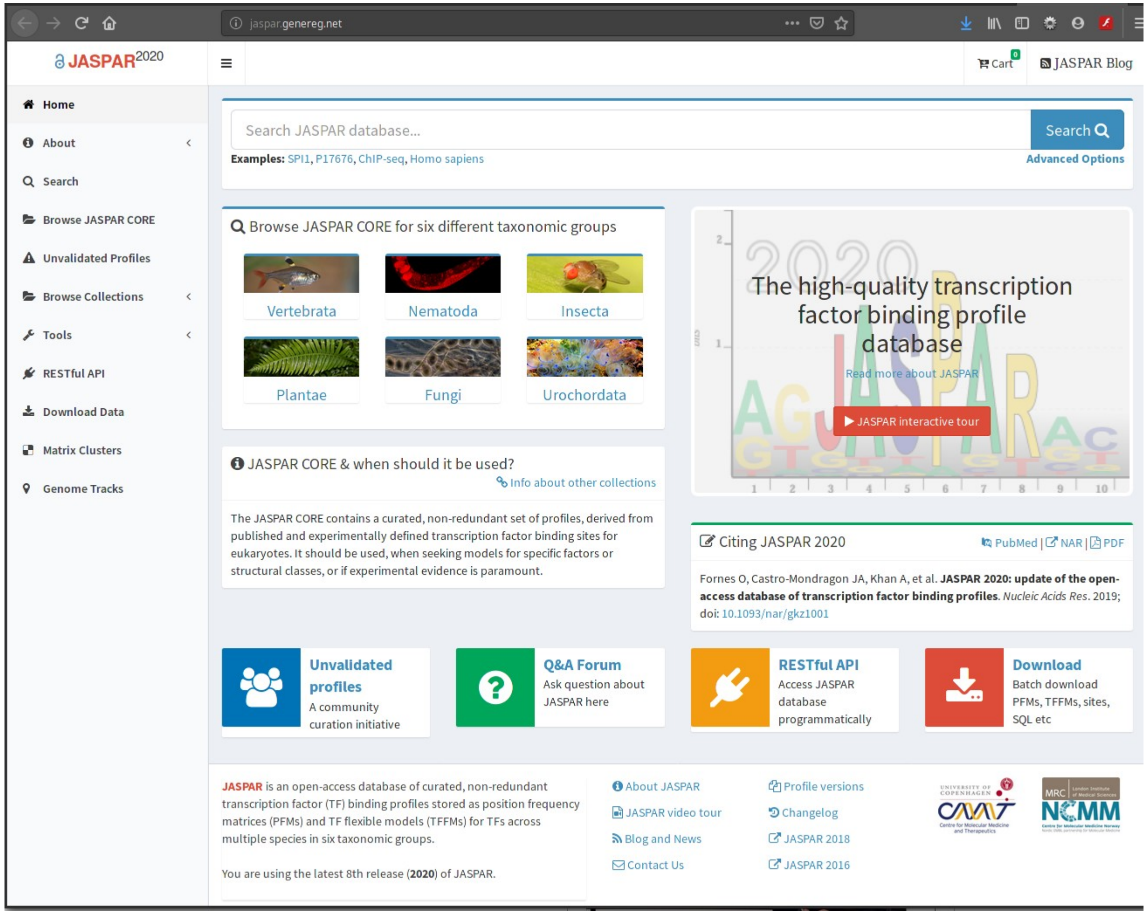
```
>>> m.consensus
Seq('TACGC')
```

#The anticonsensus sequence, corresponding to the smallest values in the columns of the .counts matrix:

```
>>> m.anticonsensus
Seq('CCATG')
```

Motif database

http://jaspar.genereg.net/



jaspar.genereg.net

JASPAR2020

Cart 0 | JASPAR Blog

- Home
- About
- Search
- Browse JASPAR CORE
- Unvalidated Profiles
- Browse Collections
- Tools
- RESTful API
- Download Data
- Matrix Clusters
- Genome Tracks

Search JASPAR database... Search

Examples: SPI1, P17676, ChIP-seq, Homo sapiens

Advanced Options

Browse JASPAR CORE for six different taxonomic groups

Vertebrata | Nematoda | Insecta | Plantae | Fungi | Urochordata

The high-quality transcription factor binding profile database

Read more about JASPAR

JASPAR interactive tour

JASPAR CORE & when should it be used?

Info about other collections

The JASPAR CORE contains a curated, non-redundant set of profiles, derived from published and experimentally defined transcription factor binding sites for eukaryotes. It should be used, when seeking models for specific factors or structural classes, or if experimental evidence is paramount.

Citing JASPAR 2020

PubMed | NAR | PDF

Fornes O, Castro-Mondragon JA, Khan A, et al. **JASPAR 2020: update of the open-access database of transcription factor binding profiles**. *Nucleic Acids Res*. 2019; doi: 10.1093/nar/gkz1001

Unvalidated profiles
A community curation initiative

Q&A Forum
Ask question about JASPAR here

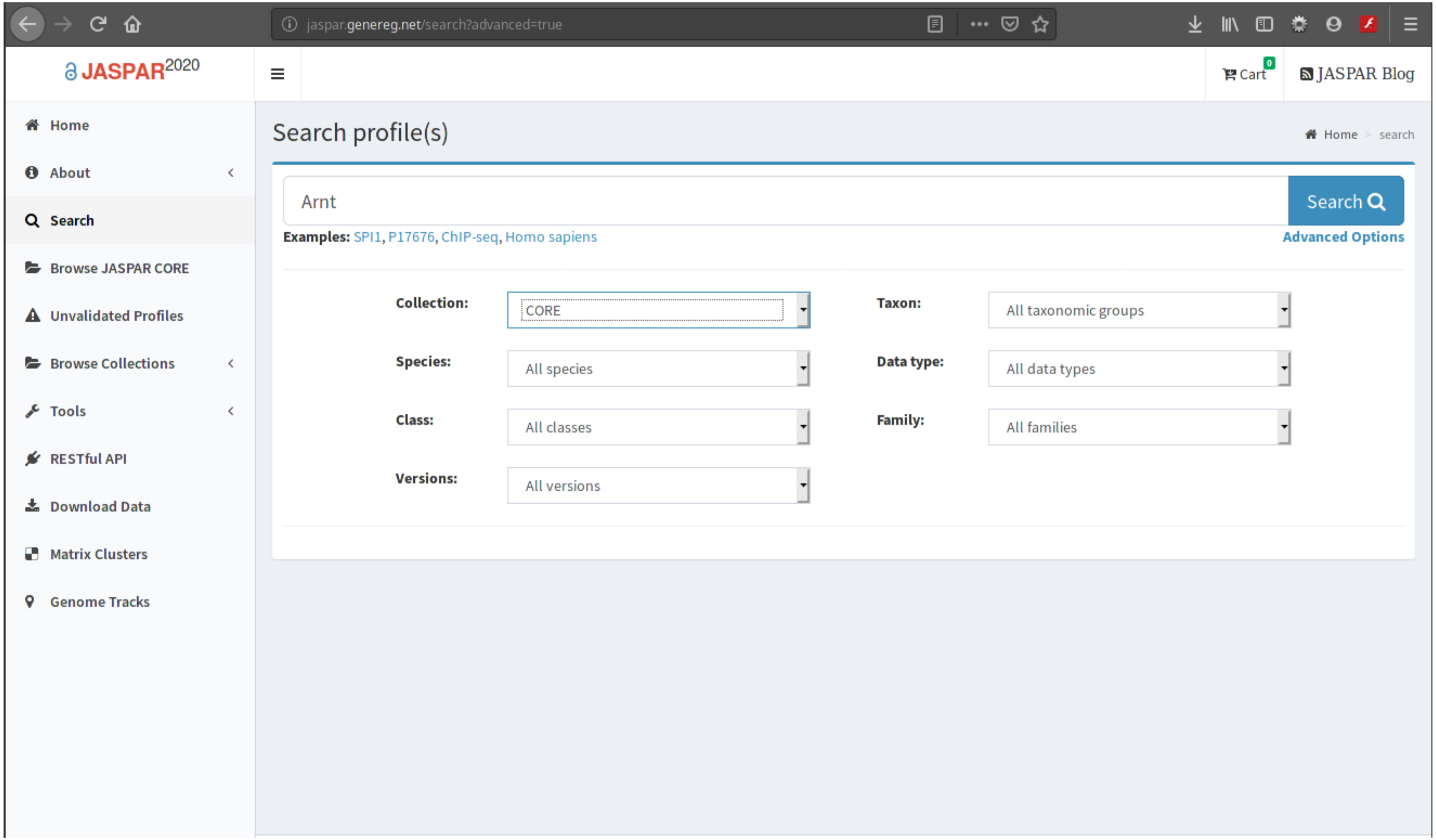
RESTful API
Access JASPAR database programmatically

Download
Batch download PFMs, TFFMs, sites, SQL etc

JASPAR is an open-access database of curated, non-redundant transcription factor (TF) binding profiles stored as position frequency matrices (PFMs) and TF flexible models (TFFMs) for TFs across multiple species in six taxonomic groups.

You are using the latest 8th release (**2020**) of JASPAR.

- About JASPAR
- JASPAR video tour
- Blog and News
- Contact Us
- Profile versions
- Changelog
- JASPAR 2018
- JASPAR 2016



jaspar.genereg.net/search?advanced=true

JASPAR2020

Cart 0 | JASPAR Blog

Home
About
Search
Browse JASPAR CORE
Unvalidated Profiles
Browse Collections
Tools
RESTful API
Download Data
Matrix Clusters
Genome Tracks

Search profile(s)

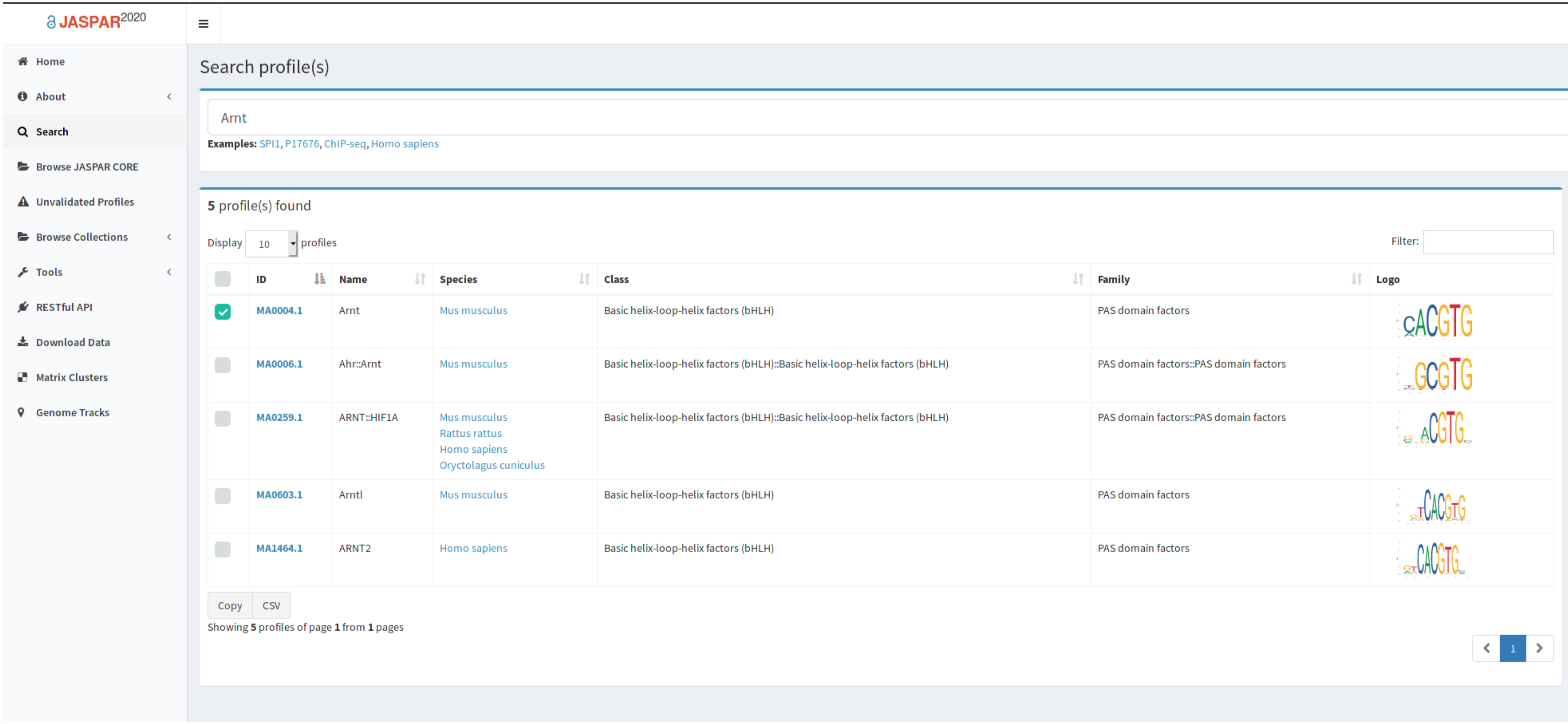
Home > search

Arnt | Search

Examples: SPI1, P17676, ChIP-seq, Homo sapiens

Advanced Options

Collection:	CORE	Taxon:	All taxonomic groups
Species:	All species	Data type:	All data types
Class:	All classes	Family:	All families
Versions:	All versions		


JASPAR2020

- Home
- About
- Search
- Browse JASPAR CORE
- Unvalidated Profiles
- Browse Collections
- Tools
- RESTful API
- Download Data
- Matrix Clusters
- Genome Tracks

Search profile(s)

Arnt

Examples: SPI1, P17676, ChIP-seq, Homo sapiens

5 profile(s) found

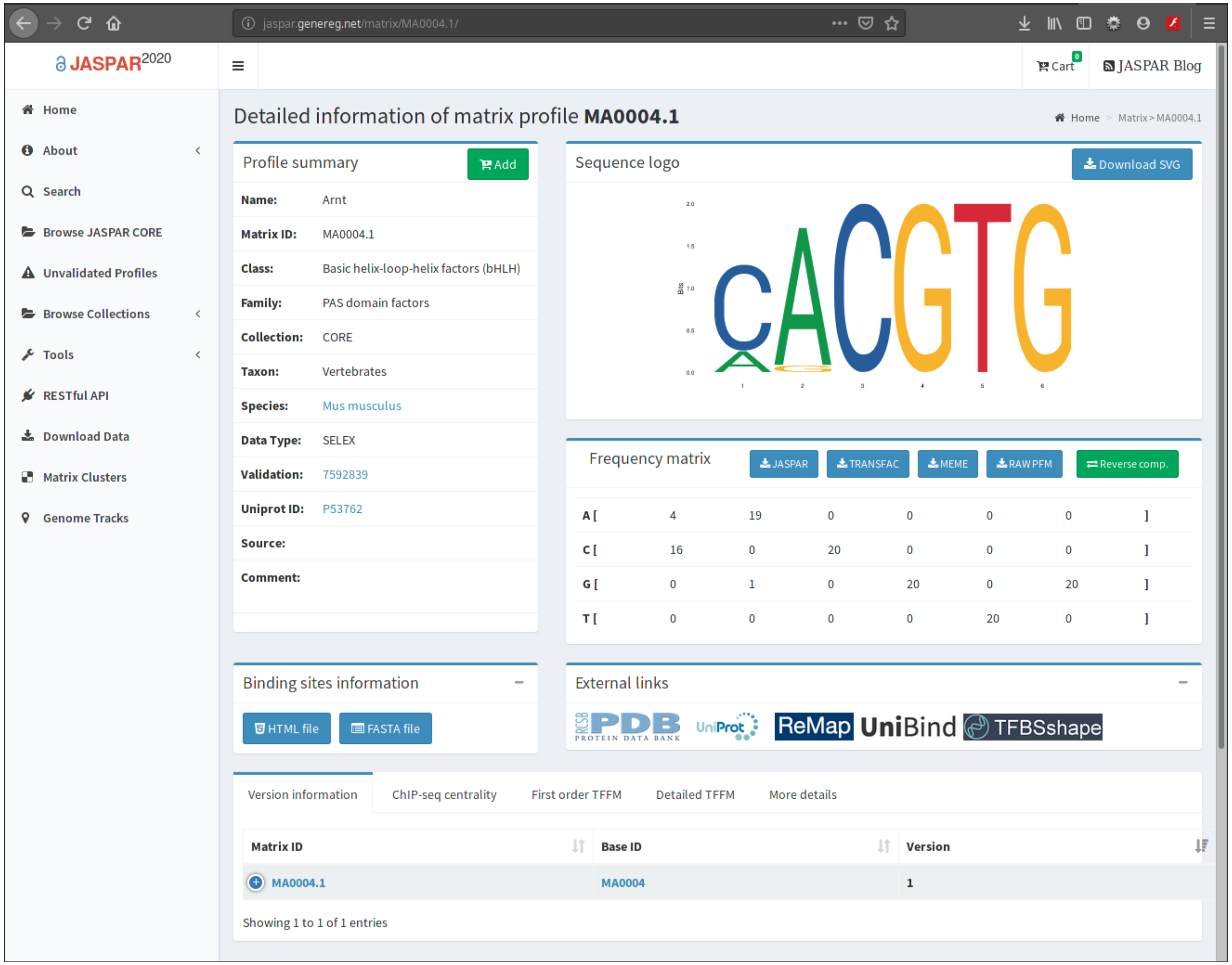
Display 10 profiles

Filter:

	ID	Name	Species	Class	Family	Logo
✓	MA0004.1	Arnt	Mus musculus	Basic helix-loop-helix factors (bHLH)	PAS domain factors	CACGTG
	MA0006.1	Ahr::Arnt	Mus musculus	Basic helix-loop-helix factors (bHLH)::Basic helix-loop-helix factors (bHLH)	PAS domain factors::PAS domain factors	GCGTG
	MA0259.1	ARNT::HIF1A	Mus musculus Rattus rattus Homo sapiens Oryctolagus cuniculus	Basic helix-loop-helix factors (bHLH)::Basic helix-loop-helix factors (bHLH)	PAS domain factors::PAS domain factors	ACGTG
	MA0603.1	Arntl	Mus musculus	Basic helix-loop-helix factors (bHLH)	PAS domain factors	TCACGTG
	MA1464.1	ARNT2	Homo sapiens	Basic helix-loop-helix factors (bHLH)	PAS domain factors	CACGTG

Copy CSV

Showing **5** profiles of page **1** from **1** pages



Detailed information of matrix profile **MA0004.1**

Profile summary

Name:	Arnt
Matrix ID:	MA0004.1
Class:	Basic helix-loop-helix factors (bHLH)
Family:	PAS domain factors
Collection:	CORE
Taxon:	Vertebrates
Species:	Mus musculus
Data Type:	SELEX
Validation:	7592839
Uniprot ID:	P53762
Source:	
Comment:	

Sequence logo

Frequency matrix

A [	4	19	0	0	0	0	]
C [	16	0	20	0	0	0	]
G [	0	1	0	20	0	20	]
T [	0	0	0	0	20	0	]

Binding sites information

External links

Version information | ChIP-seq centrality | First order TFFM | Detailed TFFM | More details

Matrix ID	Base ID	Version
MA0004.1	MA0004	1

Showing 1 to 1 of 1 entries

MA0004.1.sites

```
>MA0004 ARNT    1
CACGTGatgtcctc
>MA0004 ARNT    2
CACGTGggaggtac
>MA0004 ARNT    3
CACGTGccgcgcgc
>MA0004 ARNT    4
CACGTGaagttgtc
>MA0004 ARNT    5
taaatgcCACGTG
>MA0004 ARNT    6
aggtataCACGTG
>MA0004 ARNT    7
agtCACGTGttcc
>MA0004 ARNT    8
gggatCACGTGgt
>MA0004 ARNT    9
gggtCACGTGttc
>MA0004 ARNT    10
catgtCACGTGcc
>MA0004 ARNT    11
agttcgCACGTGc
>MA0004 ARNT    12
taagCACGTGgtc
>MA0004 ARNT    13
tgaatacCACGTG
>MA0004 ARNT    14
tgaCACGTGtccg
>MA0004 ARNT    15
attgtgCACGTGg
>MA0004 ARNT    16
AACGTGacttcgtacc
>MA0004 ARNT    17
AACGTGcgtgatgtcc
>MA0004 ARNT    18
AACGTGacagccctcc
>MA0004 ARNT    19
AACGTGcacatcgtcc
>MA0004 ARNT    20
aggaatCGCGTGc
```

Read records

We can create a Motif object from these instances as follows:

```
>>> from Bio import motifs
>>> arnt = motifs.read(open("MA0004.1.sites"), "sites")
>>> print(arnt.counts)
        0      1      2      3      4      5
A:   4.00  19.00   0.00   0.00   0.00   0.00
C:  16.00   0.00  20.00   0.00   0.00   0.00
G:   0.00   1.00   0.00  20.00   0.00  20.00
T:   0.00   0.00   0.00   0.00  20.00   0.00
```

Motif instances

The instances from which this motif was created is stored in the `.instances` property:

```
>>> print(arnt.instances[:3])
[Seq('CACGTG'), Seq('CACGTG'), Seq('CACGTG')]
>>> for instance in arnt.instances:
...     print(instance)
...
CACGTG
CACGTG
CACGTG
...
...
AACGTG
AACGTG
AACGTG
AACGTG
CGCGTG
>>> print(arnt.counts)
        0      1      2      3      4      5
A:   4.00  19.00   0.00   0.00   0.00   0.00
C:  16.00   0.00  20.00   0.00   0.00   0.00
G:   0.00   1.00   0.00  20.00   0.00  20.00
T:   0.00   0.00   0.00   0.00  20.00   0.00
```

The JASPAR pfm format

SRF.pfm

```
 2  9  0  1 32  3 46  1 43 15  2  2
 1 33 45 45  1  1  0  0  0  1  0  1
39  2  1  0  0  0  0  0  0  0 44 43
 4  2  0  0 13 42  0 45  3 30  0  0
```

```
>>> with open("SRF.pfm") as handle:
...     srf = motifs.read(handle, "pfm")
...
>>> print(srf.counts)
        0      1      2      3      4      5      6      7      8      9     10     11
A:   2.00   9.00   0.00   1.00  32.00   3.00  46.00   1.00  43.00  15.00   2.00   2.00
C:   1.00  33.00  45.00  45.00   1.00   1.00   0.00   0.00   0.00   1.00   0.00   1.00
G:  39.00   2.00   1.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00  44.00  43.00
T:   4.00   2.00   0.00   0.00  13.00  42.00   0.00  45.00   3.00  30.00   0.00   0.00


>>> print(srf.instances)
None
>>> print(arnt.counts.consensus)
CACGTG
>>> print(srf.counts.consensus)
GCCCATATATGG
```

Position-Specific Scoring Matrices

Using the background distribution and PWM with pseudo-counts added, it's easy to compute the log-odds ratios, telling us what are the log odds of a particular symbol to be coming from a motif against the background. We can use the .log_odds() method on the position-weight matrix:

```
>>> print(srf.pwm)
        0      1      2      3      4      5      6      7      8      9     10     11
A:   0.04   0.20   0.00   0.02   0.70   0.07   1.00   0.02   0.93   0.33   0.04   0.04
C:   0.02   0.72   0.98   0.98   0.02   0.02   0.00   0.00   0.00   0.02   0.00   0.02
G:   0.85   0.04   0.02   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.96   0.93
T:   0.09   0.04   0.00   0.00   0.28   0.91   0.00   0.98   0.07   0.65   0.00   0.00


>>> print(srf.pssm)
        0      1      2      3      4      5      6      7      8      9     10     11
A:  -2.52  -0.35   -inf  -3.52   1.48  -1.94   2.00  -3.52   1.90   0.38  -2.52  -2.52
C:  -3.52   1.52   1.97   1.97  -3.52  -3.52   -inf   -inf   -inf  -3.52   -inf  -3.52
G:   1.76  -2.52  -3.52   -inf   -inf   -inf   -inf   -inf   -inf   -inf   1.94   1.90
T:  -1.52  -2.52   -inf   -inf   0.18   1.87   -inf   1.97  -1.94   1.38   -inf   -inf


>>> print(srf.pwm.log_odds())
        0      1      2      3      4      5      6      7      8      9     10     11
A:  -2.52  -0.35   -inf  -3.52   1.48  -1.94   2.00  -3.52   1.90   0.38  -2.52  -2.52
C:  -3.52   1.52   1.97   1.97  -3.52  -3.52   -inf   -inf   -inf  -3.52   -inf  -3.52
G:   1.76  -2.52  -3.52   -inf   -inf   -inf   -inf   -inf   -inf   -inf   1.94   1.90
T:  -1.52  -2.52   -inf   -inf   0.18   1.87   -inf   1.97  -1.94   1.38   -inf   -inf
```

What next?

1) Searching for instances

2) Searching for exact matches

3) Searching for matches using the PSSM score

4) Selecting a score threshold

5) ...

MEME

- MEME is a tool for discovering motifs in a group of related DNA or protein sequences.
- It takes as input a group of DNA or protein sequences and outputs as many motifs as requested.
- Therefore, in contrast to JASPAR files, MEME output files typically contain multiple motifs.

Ab initio motif finding: Expectation Maximization

- Popular algorithm for motif discovery
- Motif model: Position Weight Matrix
- Local search algorithm
  - Move from current choice of motif to a new similar motif, so as to improve the score
  - Keep doing this until no more improvement is obtained: converge to local optima

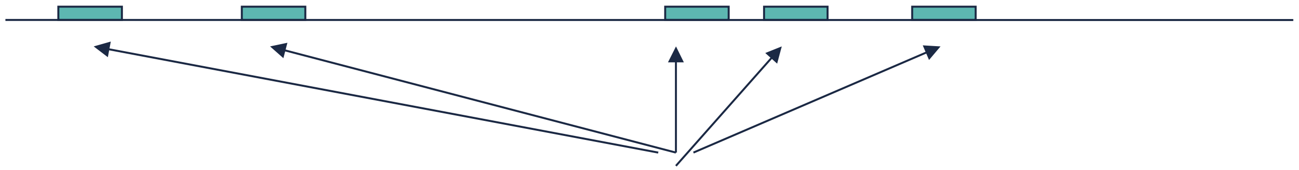
Ackn.: R. Bruskiewich and F. Brinkman, MBB with material from Wyeth W. Wasserman and Shannan Ho Sui

Basic idea of iteration

1.

Current motif

2. Scan sequence for good matches to the current motif.



3. Build a new PWM out of these matches, and make it the new motif

The basic idea can be formalized in the language of probability

Ackn.: R. Bruskiewich and F. Brinkman, MBB with material from Wyeth W. Wasserman and Shannan Ho Sui

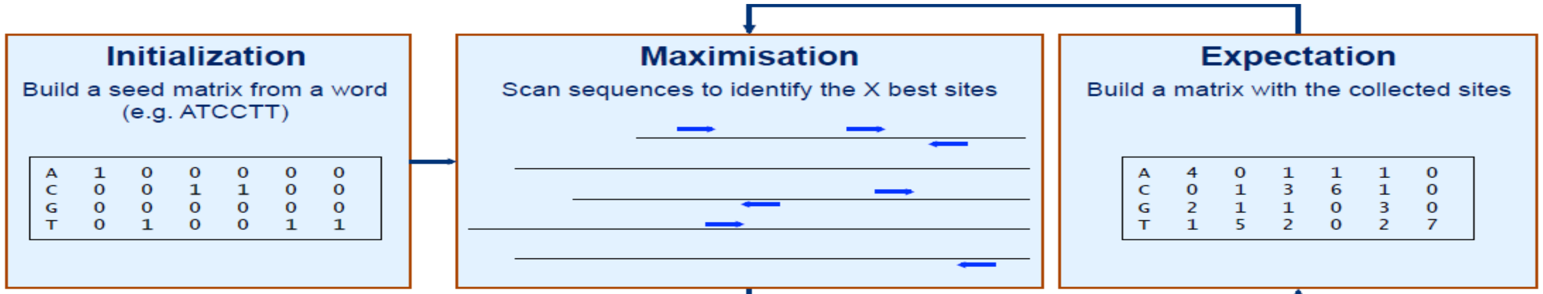
MEME

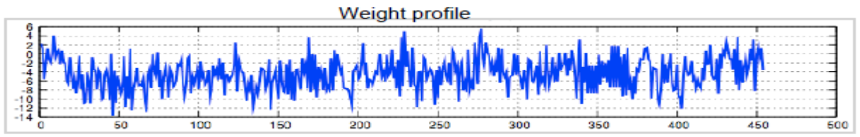
EM

- Instantiate a “seed” motif
- Iterate N times
  - Maximization: select the X highest scoring sites
  - Expectation: build a new matrix from the collected sites

Multiple EM

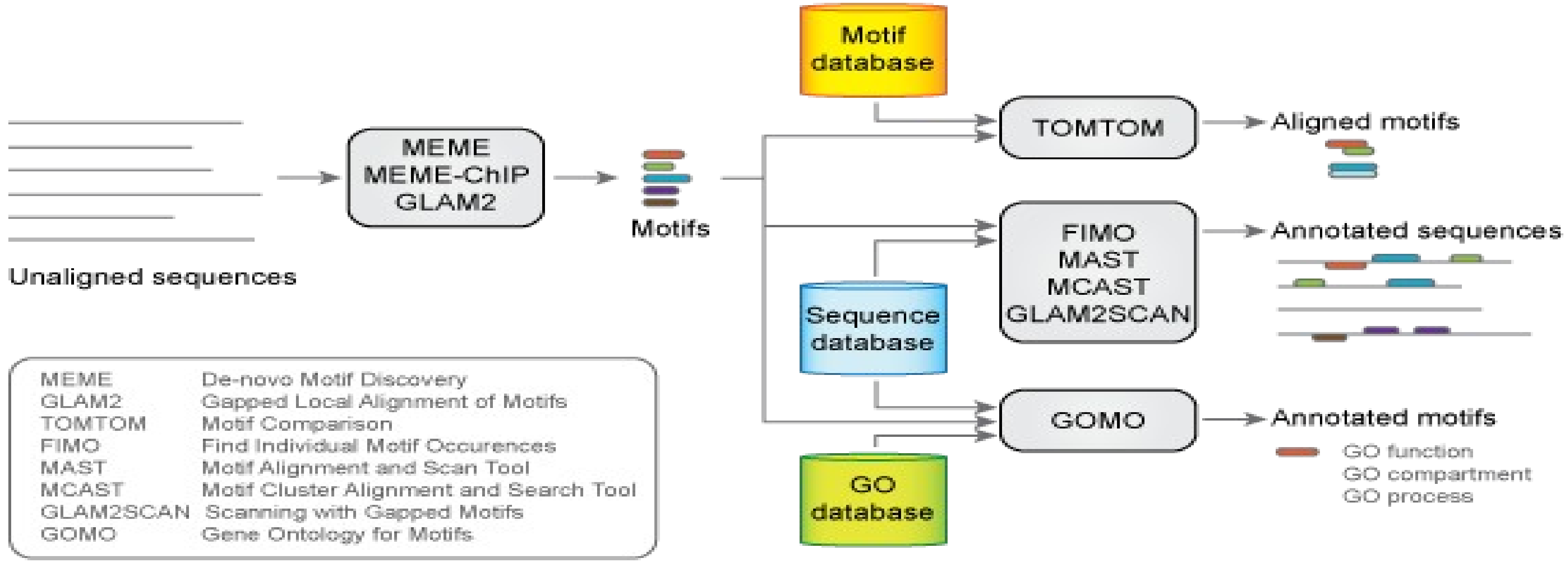
- Iterate over each k-mer found in the input set
  - building a matrix from the k-mer
  - Run an EM (expectation/maximisation) algorithm to optimize the matrix.
- Return the highest scoring matrices.





MEME

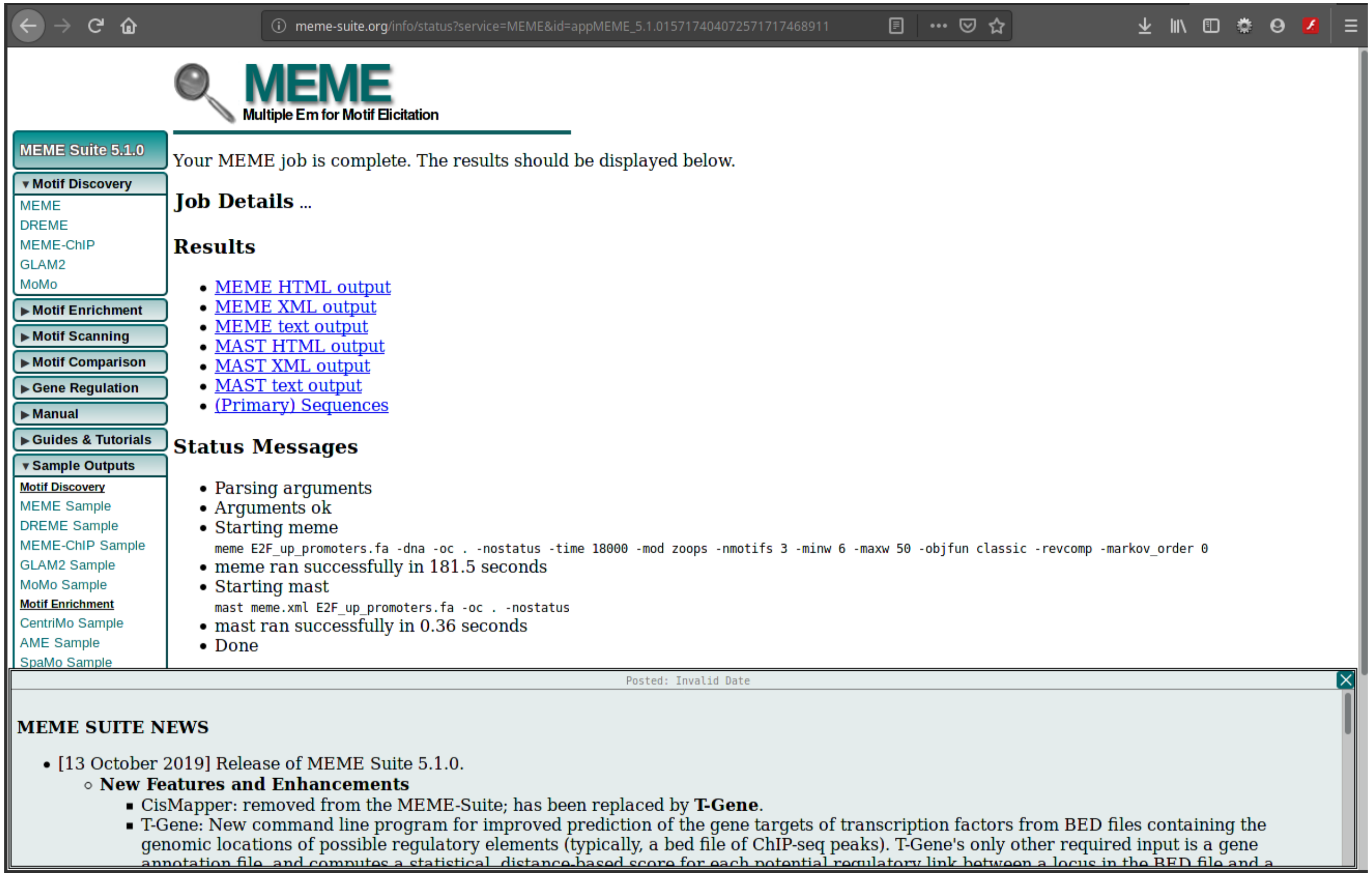
- Popular motif finding program that uses Expectation-Maximization
- http://meme-suite.org/



MEME	De-novo Motif Discovery
GLAM2	Gapped Local Alignment of Motifs
TOMTOM	Motif Comparison
FIMO	Find Individual Motif Occurences
MAST	Motif Alignment and Scan Tool
MCAST	Motif Cluster Alignment and Search Tool
GLAM2SCAN	Scanning with Gapped Motifs
GOMO	Gene Ontology for Motifs

Mod. from Stewart MacArthur - DNA Motif Finding

Run MEME with ‘E2F_up_promoters.fa’



Your MEME job is complete. The results should be displayed below.

Job Details ...

Results

- MEME HTML output
- MEME XML output
- MEME text output
- MAST HTML output
- MAST XML output
- MAST text output
- (Primary) Sequences

Status Messages

- Parsing arguments
- Arguments ok
- Starting meme

```
meme E2F_up_promoters.fa -dna -oc . -nostatus -time 18000 -mod zoops -nmotifs 3 -minw 6 -maxw 50 -objfun classic -revcomp -markov_order 0
```

- meme ran successfully in 181.5 seconds
- Starting mast

```
mast meme.xml E2F_up_promoters.fa -oc . -nostatus
```

- mast ran successfully in 0.36 seconds
- Done

Posted: Invalid Date

MEME SUITE NEWS

- [13 October 2019] Release of MEME Suite 5.1.0.
  - **New Features and Enhancements**
    - CisMapper: removed from the MEME-Suite; has been replaced by **T-Gene**.
    - T-Gene: New command line program for improved prediction of the gene targets of transcription factors from BED files containing the genomic locations of possible regulatory elements (typically, a bed file of ChIP-seq peaks). T-Gene's only other required input is a gene annotation file, and computes a statistical, distance-based score for each potential regulatory link between a locus in the BED file and a

Example result

```
********************************************************************************
COMMAND LINE SUMMARY
********************************************************************************
This information can also be useful in the event you wish to report a
problem with the MEME software.

command: meme E2F_up_promoters.fa -dna -oc . -nostatus -time 18000 -mod zoops -nmotifs 3 -minw 6 -maxw 50 -objfun classic -revcomp -markov_order 0


…

********************************************************************************
MOTIF GCCSGGSGCGGTGGCTCACGCCTSTAATCCCAGCACTTTGGGAGGCCGAG MEME-1    width =  50  sites =  12  llr = 569  E-value = 3.5e-109
********************************************************************************
--------------------------------------------------------------------------------
        Motif GCCSGGSGCGGTGGCTCACGCCTSTAATCCCAGCACTTTGGGAGGCCGAG MEME-1 Description
--------------------------------------------------------------------------------
Simplified        A  :::21::::2:22:2328:3::::28a2:2:8:191:1::2:a:2::2a:
pos.-specific     C  :894:1719::::17:8:8:8823:2::88a::9:81:1::::::a92::
probability       G  a1:49939:8a:89::13272::6::::::::a:::23:88a:a8::7:a
matrix            T  :11:::::1::8::28:::112818::82::2::118792::::::1:::


         bits    2.3                           *               *     *
                 2.0                           *               *     *
                 1.8 *         *               *   * * *   *  *** *  **
                 1.6 *         **          * ****  *** *   *  *** *  **
Relative         1.4 * * ** ** ** * * *    * ****  *****   *  *** ** **
Entropy          1.1 * * ** ******* * **  ** ************* ********* **
(68.5 bits)      0.9 *** ********** ****  ** *********************** **
                 0.7 *** ******************* **************************
                 0.5 **************************************************
                 0.2 **************************************************
                 0.0 --------------------------------------------------


Multilevel           GCCCGGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAG
consensus               G  G        A G A   C             G
sequence

--------------------------------------------------------------------------------
```

Parse results

```
>>> handle = open("meme.txt")
>>> record = motifs.parse(handle, "meme")
>>> handle.close()
>>> len(record)
3
>>> motif = record[0]
>>> print(motif.consensus)
CAAGAGCCTGCT
>>> print(motif.degenerate_consensus)
CAAGAGCCTGCT
```

Motif attributes

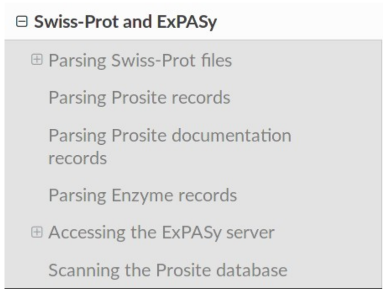
```
>>> motif.num_occurrences
51
>>> motif.length
12
>>> evalue = motif.evalue
>>> print("%3.1g" % evalue)
1e-161
>>> motif.name
'Motif 1
```

Where the motif was found

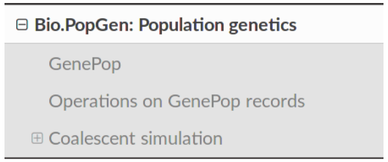
```
>>> motif = record['Motif 1']
>>> # Each motif has an attribute .instances with the sequence instances in which
the motif was found, providing some information on each instance
...
>>> len(motif.instances)
51
>>> motif.instances[0]
Instance('CAAGAGCCTGCT', 'ACGT')
>>> motif.instances[0].start
96
>>> motif.instances[0].strand
'+'
>>> motif.instances[0].length
12
>>> pvalue = motif.instances[0].pvalue
>>> print("%5.3g" % pvalue)
6.76e-08
```

Wait there is more!

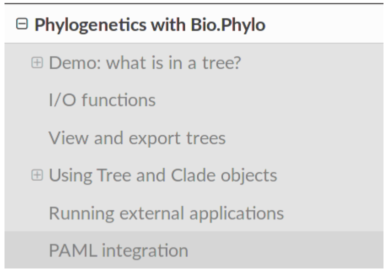
http://biopython.org/DIST/docs/tutorial/Tutorial.html



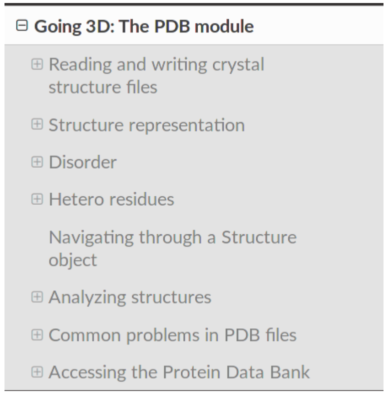
Swiss-Prot and ExPASy
- Parsing Swiss-Prot files
- Parsing Prosite records
- Parsing Prosite documentation records
- Parsing Enzyme records
- Accessing the ExPASy server
- Scanning the Prosite database



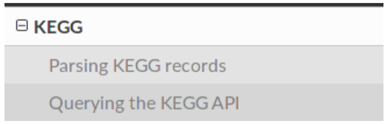
Bio.PopGen: Population genetics
- GenePop
- Operations on GenePop records
- Coalescent simulation



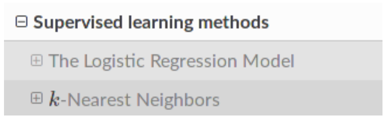
Phylogenetics with Bio.Phylo
- Demo: what is in a tree?
- I/O functions
- View and export trees
- Using Tree and Clade objects
- Running external applications
- PAML integration



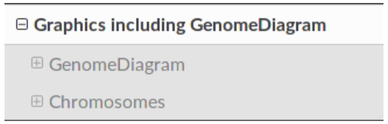
Going 3D: The PDB module
- Reading and writing crystal structure files
- Structure representation
- Disorder
- Hetero residues
- Navigating through a Structure object
- Analyzing structures
- Common problems in PDB files
- Accessing the Protein Data Bank



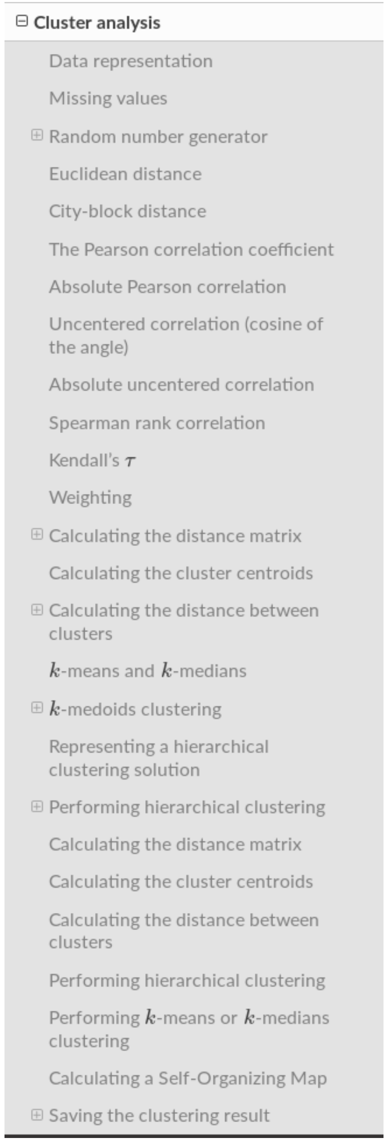
KEGG
- Parsing KEGG records
- Querying the KEGG API



Supervised learning methods
- The Logistic Regression Model
- k-Nearest Neighbors



Graphics including GenomeDiagram
- GenomeDiagram
- Chromosomes



Cluster analysis
- Data representation
- Missing values
- Random number generator
- Euclidean distance
- City-block distance
- The Pearson correlation coefficient
- Absolute Pearson correlation
- Uncentered correlation (cosine of the angle)
- Absolute uncentered correlation
- Spearman rank correlation
- Kendall's τ
- Weighting
- Calculating the distance matrix
- Calculating the cluster centroids
- Calculating the distance between clusters
- k-means and k-medians
- k-medoids clustering
- Representing a hierarchical clustering solution
- Performing hierarchical clustering
- Calculating the distance matrix
- Calculating the cluster centroids
- Calculating the distance between clusters
- Performing hierarchical clustering
- Performing k-means or k-medians clustering
- Calculating a Self-Organizing Map
- Saving the clustering result