# Biopython

Gustavo Isaza E, PhD
Profesor Titular. Investigador Senior
gustavo.isaza@ucaldas.edu.co
https://github.com/gisazae
https://www.researchgate.net/profile/Gustavo_Isaza

# Classes

```
class <classname>:
  statement_1
  .
  .
  statement_n
```

- Los métodos de una clase obtienen la instancia como primer parámetro

  tradicionalmente se termina como **`self`**

- El método **`__init__`** llama a la construcción de objetos (si está disponible)

# Classes

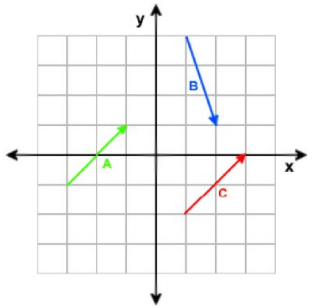
- ***type*** = **datos** (representación) + **comportamiento**.
- **Classes are user-defined types.**

```
class <classname>:
    statement_1
    .
    .
    statement_n
```

- Como un mini-programa:
  - Variables.
  - Definiciones de funciones.
  - Incluso comandos arbitrarios

- **Los Objetos** de una clase se conocen **instancias de clase**.

# Classes – Atributos y Métodos



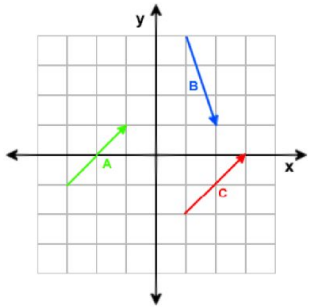

```
class Vector2D:

    def __init__(self, x, y):
        self.x, self.y = x, y

    def size(self):
        return (self.x ** 2 + self.y ** 2) ** 0.5
```

**Instance**

**Methods**

**Attributes**
**(each instance has *its own copy*)**

# Classes – Instantiate and Use




```
>>> v = Vector2D(3, 4) # Make instance.
>>> v
<__main__.Vector2D object at 0x00000000031A2828>

>>> v.size() # Call method on instance.
5.0
```

biopython

# Biopython

- Una asociación internacional de desarrolladores de herramientas Python (http://www.python.org) disponibles gratuitamente para biología molecular computacional
- Proporciona herramientas para
  - Análisis de archivos (fasta, clustalw, GenBank,...)
  - Interfaz a softwares comunes
  - Operaciones en secuencias
  - Aplicaciones sencillas de aprendizaje automático !!

# Instalando Biopython

- Ir a http://biopython.org/wiki/Download
- Windows
- Unix
- Seleccionar python 3.X
- Se requiere NumPy

# SeqIO

- La interfaz de entrada/salida de secuencia estándar para `BioPython`
- Trata con secuencias como objetos `SeqRecord`
- Hay una interfaz hermana, `Bio.AlignIO`, para trabajar directamente con archivos de alineamiento de secuencias como objetos de alineación.

# Analizando un archivo FASTA

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

```
gi|2765580|emb|Z78455.1|PJZ78455
Seq('CGTAACCAGGTTTCCGTAGGTGGACCTTCGGGAGGATCATTTTTGAGATCACAT...GCA', SingleLetterAlphabet())
745
gi|2765579|emb|Z78454.1|PFZ78454
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGATCACAT...AAC', SingleLetterAlphabet())
695
gi|2765578|emb|Z78453.1|PSZ78453
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGATCACAT...GCA', SingleLetterAlphabet())
745
gi|2765577|emb|Z78452.1|PBZ78452
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGATCACAT...GCA', SingleLetterAlphabet())
743
gi|2765576|emb|Z78451.1|PHZ78451
Seq('CGTAACAAGGTTTCCGTAGGTGTACCTCCGGAAGGATCATTGTTGAGATCACAT...AGC', SingleLetterAlphabet())
730
gi|2765575|emb|Z78450.1|PPZ78450
Seq('GGAAGGATCATTGCTGATATCACATAATAATTGATCGAGTTAAGCTGGAGGATC...GAG', SingleLetterAlphabet())
706
gi|2765574|emb|Z78449.1|PMZ78449
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGATCACAT...TGC', SingleLetterAlphabet())
744
gi|2765573|emb|Z78448.1|PAZ78448
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGATCACAT...AGG', SingleLetterAlphabet())
742
gi|2765572|emb|Z78447.1|PVZ78447
Seq('CGTAACAAGGATTCCGTAGGTGAACCTGCGGGAGGATCATTGTTGAGATCACAT...AGC', SingleLetterAlphabet())
694
gi|2765571|emb|Z78446.1|PAZ78446
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTCCGGAAGGATCATTGTTGAGATCACAT...CCC', SingleLetterAlphabet())
712
gi|2765570|emb|Z78445.1|PUZ78445
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGATCACAT...TGT', SingleLetterAlphabet())
715
gi|2765569|emb|Z78444.1|PAZ78444
Seq('CGTAACAAGGTTTCCGTAGGGTGAACTGCGGAAGGATCATTGTTGAGATCACAT...ATT', SingleLetterAlphabet())
688
```

# Archivos de GenBank

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

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

# Archivos de GenBank

```
from Bio import SeqIO
for seq_record in SeqIO.parse("ls_orchid.gbk", "genbank"):
    print(seq_record.id)
    print(repr(seq_record.seq))
    print(len(seq_record))
```

```
Z78447.1
Seq('CGTAACAAGGATTCCGTAGGTGAACCTGCGGGAGGATCATTGTTGAGATCACAT...AGC', IUPACAmbiguousDNA())
694
Z78446.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTCCGGAAGGATCATTGTTGAGATCACAT...CCC', IUPACAmbiguousDNA())
712
Z78445.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGATCACAT...TGT', IUPACAmbiguousDNA())
715
Z78444.1
Seq('CGTAACAAGGTTTCCGTAGGGTGAACTGCGGAAGGATCATTGTTGAGATCACAT...ATT', IUPACAmbiguousDNA())
688
```

# Objetos de Secuencias

- Admite métodos similares a las cadenas estándar
- Proporcionar métodos adicionales
- Traducción
- complemento inverso
- Admite diferentes alfabetos
  - AGTAGTTAAA puede ser
  - ADN
  - Proteína

# The `alphabet` attribute

- Alphabets are defined in the **`Bio.Alphabet`** module
- We will use the IUPAC alphabets (http://www.chem.qmw.ac.uk/iupac)
- **`Bio.Alphabet.IUPAC`** provides definitions for DNA, RNA and proteins + provides extension and customisation of basic definitions:
  - **`IUPACProtein`** **`(IUPAC standard AA)`**
  - **`ExtendedIUPACProtein (+ selenocysteine, X, etc)`**
  - **`IUPACUnambiguousDNA`** **`(basic GATC letters)`**
  - **`IUPACAmbiguousDNA (+ ambiguity letters)`**
  - **`ExtendedIUPACDNA (+ modified bases)`**
  - **`IUPACUnambiguousRNA`**
  - **`IUPACAmbiguousRNA`**

# Alfabeto genérico

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

Non-specific alphabet

# Secuencias específicas

```
>>> 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()

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

# Las secuencias se tratan como Strings

- Acceder a elementos

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

- Contar sin overlaps

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

# Calcular contenido de GC

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

# Desplazamientos y Cortes

- Slicing simple

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

- Empezar, parar, dar saltos

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

# Concatenar

- La adición simple de Python

```
>>> from Bio.Seq import Seq
>>> protein_seq = Seq("EVRNAK")
>>> dna_seq = Seq("ACGT")
>>> protein_seq + dna_seq
```

# Cambiando CAPS

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

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

# Complemento inverso

```
>>> from Bio.Seq import Seq
>>> my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")
>>> my_seq.complement()
Seq('CTAGCTACCCGGATATATCCTAGCTTTTAGCG')
>>> my_seq.reverse_complement()
Seq('GCGATTTTCGATCCTATATAGGCCCATCGATC')
```

# Transcripción

```
>>> coding_dna =
Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG)
>>> template_dna = coding_dna.reverse_complement()
>>> messenger_rna = coding_dna.transcribe()
>>> messenger_rna
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUA
G')
```

Todo lo que esto hace es cambiar T → U y ajustar el alfabeto.

# Traducción

## Ejemplo

```
>>> from Bio.Seq import Seq
>>> messenger_rna =
Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG”)
>>> messenger_rna
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG')
>>> messenger_rna.translate()
Seq('MAIVMGR*KGAR*')
```

Stop codon!

# Traducir hasta la primera parada en el cuadro

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

# Comparando Secuencias

Standard “==“ comparison is done by comparing the references (!):

```
>>> seq1 = Seq("ACGT")
>>> seq2 = Seq("ACGT")
>>> seq1==seq2
Warning (from warnings module):
… FutureWarning: In future comparing Seq objects will use string comparison (not object comparison). Incompatible alphabets will trigger a warning (not an exception)… please use str(seq1)==str(seq2) to make your code explicit and to avoid this warning.
True
>>> seq1==seq1
True
```

# Secuencias desconocidas

- En muchos casos biológicos tratamos con [illegible]nocidas

```
from Bio.Seq import Seq
from Bio.Seq import UnknownSeq
#unk_dna = UnknownSeq(20)
unk_dna = Seq("N"*20)
print(unk_dna)
my_seq = Seq("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA")
unk_dna + my_seq
#Seq('NNNNNNNNNNNNNNNNNNNNGCCATTGTAATGGGCCGCTGAAAGGGTGCCC
GA')
```

# Leer MSA (Alineamiento múltiple de secuencias

- Usar Bio.AlignIO.read(file, format)
- File – el path del archivo
- Formatos soportados:
  - “stockholm”
  - “fasta”
  - “clustal”
  - …
- Usar help(AlignIO)

# Ejemplo

- Queremos analizar este fichero (output) de PFAM

```
# STOCKHOLM 1.0
#=GS COATB_BPIKE/30-81  AC P03620.1
#=GS COATB_BPIKE/30-81  DR PDB; 1ifl ; 1-52;
#=GS Q9T0Q8_BPIKE/1-52  AC Q9T0Q8.1
#=GS COATB_BPI22/32-83  AC P15416.1
#=GS COATB_BPM13/24-72  AC P69541.1
#=GS COATB_BPM13/24-72  DR PDB; 2cpb ; 1-49;
#=GS COATB_BPM13/24-72  DR PDB; 2cps ; 1-49;
#=GS COATB_BPZJ2/1-49   AC P03618.1
#=GS Q9T0Q9_BPFD/1-49   AC Q9T0Q9.1
#=GS Q9T0Q9_BPFD/1-49   DR PDB; 1nh4 A; 1-49;
#=GS COATB_BPIF1/22-73  AC P03619.2
#=GS COATB_BPIF1/22-73  DR PDB; 1ifk ; 1-50;
COATB_BPIKE/30-81           AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA
#=GR COATB_BPIKE/30-81  SS  -HHHHHHHHHHHHHH--HHHHHHHH--HHHHHHHHHHHHHHHHHHHHH----
Q9T0Q8_BPIKE/1-52           AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA
COATB_BPI22/32-83           DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA
COATB_BPM13/24-72           AEGDDP...AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA
#=GR COATB_BPM13/24-72  SS  ---S-T...CHCHHHHCCCCTCCCTTCHHHHHHHHHHHHHHHHHHHHCTT--
COATB_BPZJ2/1-49            AEGDDP...AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA
Q9T0Q9_BPFD/1-49            AEGDDP...AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA
#=GR Q9T0Q9_BPFD/1-49   SS  ------...-HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH--
COATB_BPIF1/22-73            FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA
#=GR COATB_BPIF1/22-73   SS  XX-HHHH--HHHHHH--HHHHHHH--HHHHHHHHHHHHHHHHHHHHHH---
#=GC SS_cons                XHHHHHHHHHHHHHHHHCHHHHHHHHCHHHHHHHHHHHHHHHHHHHHHC--
#=GC seq_cons               AEssss...AptAhDSLpspAT-hIu.sWshVsslVsAsluIKLFKKFsSKA
```

# Ejemplo

```
from Bio import AlignIO
alignment = AlignIO.read("PF05371.sth", "stockholm")
print(alignment)
```

```
>>>
SingleLetterAlphabet() alignment with 7 rows and 52 columns
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRL...SKA COATB_BPIKE/30-81
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKL...SRA Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRL...SKA COATB_BPI22/32-83
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKL...SRA COATB_BPIF1/22-73
```

# Ejemplo de Alineamiento como objeto

```
>>> from Bio import AlignIO
>>> alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
>>> print(alignment[1])
ID: Q9T0Q8_BPIKE/1-52
Name: Q9T0Q8_BPIKE
Description: Q9T0Q8_BPIKE/1-52
Number of features: 0
/start=1
/end=52
/accession=Q9T0Q8.1
Seq('AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVI
KLFKKFVSRA', SingleLetterAlphabet())
```

# Ejemplo de Alineamiento como objeto

```
>>> print("Alignment length %i" % alignment.get_alignment_length())
Alignment length 52
>>> for record in alignment:
    print("%s - %s" % (record.seq, record.id))
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA
- COATB_BPIKE/30-81
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA
- Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA
- COATB_BPI22/32-83
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA -
COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA -
COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA -
Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVS
RA - COATB_BPIF1/22-73
```

# Escribir alineamiento a un archivo

```
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
3 9
Delta       GTCAGC-AG
Epislon     GACAGCTAG
Zeta        GTCAGCTAG
3 13
Eta         ACTAGTACAG CTG
Theta       ACTAGTACAG CT-
Iota        - CTACTACAG GTG
```

# Aplicaciones externas

- ¿Cómo llamamos a los algoritmos MSA en un conjunto de secuencias no alineado?
- Biopython proporciona wrappers
- La idea:
  - Cree un objeto de línea de comando con las opciones de algoritmo
  - Invocar el comando (Python usa subprocesos)
- Bio.Align.Applications module:
  - >>> import Bio.Align.Applications
  - >>> dir(Bio.Align.Applications)
  - ['ClustalwCommandline', 'DialignCommandline', 'MafftCommandline', 'MuscleCommandline', 'PrankCommandline', 'ProbconsCommandline', 'TCoffeeCommandline' ]

# Ejemplo ClustalW

- First step:descargar ftp://ftp.ebi.ac.uk/pub/software/clustalw2/2.1/
- Second step: instalar
- Third step: buscar por binarios y/o exe
- Lanzar desde Python

# Ejecutar ejemplo

```
>>> import os
>>> from Bio.Align.Applications import ClustalwCommandline
>>> clustalw_exe = r"/path/to/clustalw"
>>> clustalw_cline = ClustalwCommandline(clustalw_bin,
infile="opuntia.fasta")
>>> assert os.path.isfile(clustalw_bin), "Clustal W executable missing"
>>> stdout, stderr = clustalw_cline()
```

The command line is actually
a function we can run!

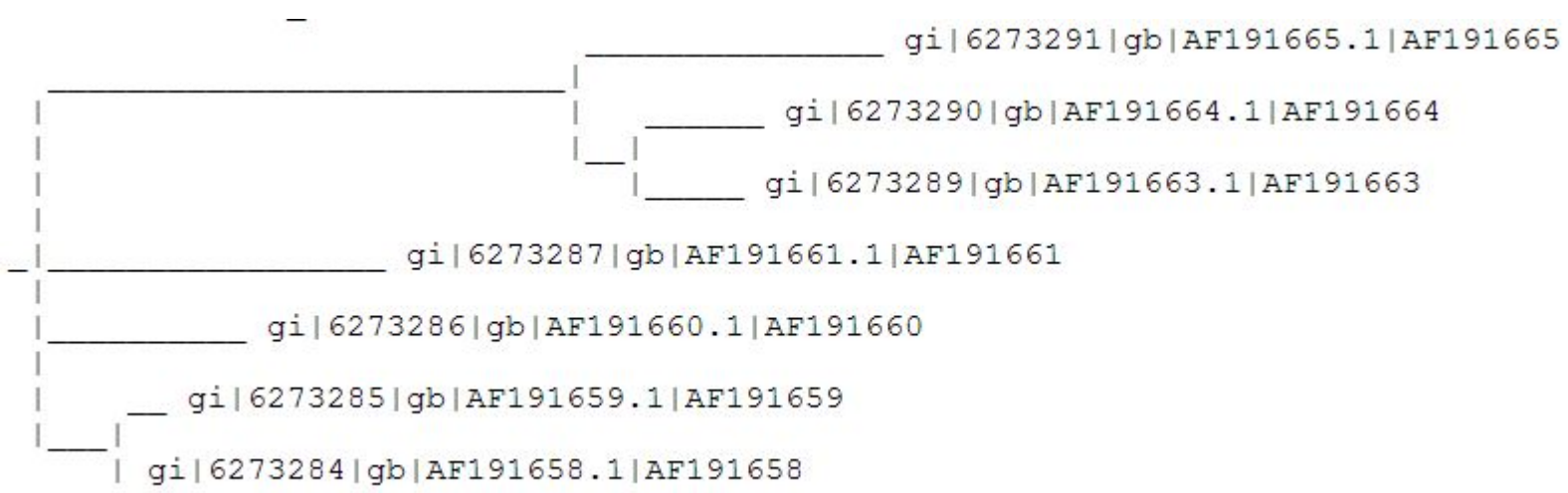
# ClustalW

```
>>> from Bio import AlignIO
>>> align = AlignIO.read("opuntia.aln", "clustal")
>>> print(align)
SingleLetterAlphabet() alignment with 7 rows and 906 columns
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAGA
gi|6273285|gb|AF191659.1|AF191
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAGA
gi|6273284|gb|AF191658.1|AF191
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAGA
gi|6273287|gb|AF191661.1|AF191
TATACATAAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAGAGA
gi|6273286|gb|AF191660.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAGAGA
gi|6273290|gb|AF191664.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAGAGA
gi|6273289|gb|AF191663.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAGAGA
gi|6273291|gb|AF191665.1|AF191
```

# ClustalW - tree

**En caso de que esté interesado, el archivo opuntia.dnd que crea ClustalW es solo un archivo de árbol estándar de Newick, y Bio.Phylo puede analizarlos:**

```
>>> from Bio import Phylo
>>> tree = Phylo.read("opuntia.dnd", "newick")
>>> Phylo.draw_ascii(tree)
```



# BLAST

# Ejecutar BLAST sobre internet

- Usamos la función qblast() en el módulo Bio.Blast.NCBIWWW. Esto tiene tres argumentos no opcionales:
  - El programa blast a utilizar para la búsqueda, como cadena en minúsculas: funciona con blastn, blastp, blastx, tblast y tblastx.
  - Las bases de datos en las que buscar. Las opciones para esto están disponibles en las páginas web del NCBI en http://www.ncbi.nlm.nih.gov/BLAST/blast_databases.shtml.
  - Una cadena que contiene su secuencia de consulta. Puede ser la secuencia en sí, la secuencia en formato fasta o un identificador como un número GI.

# qblast parámetros adicionales

- **format_type:** "HTML", "Text", "ASN.1", or "XML". El valor predeterminado es "XML", ya que ese es el formato esperado por el analizador (consulte los siguientes ejemplos)
- **expect** establece la expectativa o el umbral de valor

# Step 1: Llamar 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("m_cold.fasta").read()
>>> result_handle = NCBIWWW.qblast("blastn", "nt", fasta_string)

# option 3 – parse file to seq object
>>> record = SeqIO.read(open("m_cold.fasta"), format="fasta")
>>> result_handle = NCBIWWW.qblast("blastn", "nt", record.seq)
```

# Step2: Analizar los resultados

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

- ¡La lectura solo se puede usar una vez!
- El objeto `blast_record` mantiene los resultados reales.

# Consideraciones

- Básicamente, Biopython admite la lectura de resultados BLAST de HTML y archivos de texto.
- Estos métodos no son estables y a veces fallan porque los servidores cambian el formato.
- XML es estable
- Puede guardar archivos XML

# Save results as XML

```
save_file = open("my_blast.xml", "w") <<<
>>> save_file.write(result_handle.read())
>>> save_file.close()
()>>> result_handle.close
```

# BLAST records

- A BLAST Record contains everything you might ever want to extract from the BLAST output.
- Example:

```
E_VALUE_THRESH = 0.04 <<<
>>> for alignment in blast_record.alignments:
for hsp in alignment.hsps:
if hsp.expect < E_VALUE_THRESH:
print '****Alignment****'
print 'sequence:', alignment.title
print 'length:', alignment.length
print 'e value:', hsp.expect
print hsp.query[0:75] + "
print hsp.match[0:75] + "
" +              print hsp.sbjct[0:75]
```

# BLAST records

```
>>> ================================ RESTART ================================
>>>
****Alignment****
sequence: gi|224094601|ref|XM_002310151.1| Populus trichocarpa predicted protein, mRNA
length: 1089
e value: 8.0158e-88
AAAATGGGGAGAGAAATGAAGTACTTGGCCATGAAAACTGATCAATTGGCCGTGGCTAATATGATCGATTCCGAT
||||||||||    ||| |||  |||   ||||| |||||| |    | |   |||||| | || || ||||||
AAAATGGGGAGG---ATGGAGTTTTTGAAGATGAAGACTGATGATGAAGTCAGCGCTAATTTAATTGAGTCCGAT
****Alignment****
sequence: gi|470129329|ref|XM_004300526.1| PREDICTED: Fragaria vesca subsp. vesca cold-regulated 413 pl
e (LOC101313417), mRNA
length: 1099
e value: 5.05852e-84
ATGGGGAGAGAAATGAAGTACTTGGCCATGAAAACTGATCAATTGGCCGTGGCTAAT---ATGATCGATTCCGAT
|||||||| |   || | || ||||| |||||||||| | | | ||   ||| |||    ||||  ||||||||
ATGGGGAGGG---TGGACTATTTGGCTATGAAAACTGACCCAGTTGC---GGCCAATGAGTTGATGAATTCCGAT
****Alignment****
sequence: gi|359495761|ref|XM_002274845.2| PREDICTED: Vitis vinifera uncharacterized LOC100267774 (LOC1
length: 853
e value: 1.7656e-83
TGAACAGAAAATGGGGAGAGAAATGAAGTACTTGGCCATGAAAACTGATCAATTGGCCGTGGCTAATATGATCGA
||||  ||||||||||||    ||| ||||  |||| ||||||||||||    |  |    | ||| ||||  |
TGAAACGAAAATGGGGAGG---ATGGAGTATCTGGCTATGAAAACTGATCCC--GAACCAACCCAAT-TGATTAA
****Alignment****
sequence: gi|349709091|emb|FQ378501.1| Vitis vinifera clone SS0AEB13YG07
length: 834
e value: 1.7656e-83
TGAACAGAAAATGGGGAGAGAAATGAAGTACTTGGCCATGAAAACTGATCAATTGGCCGTGGCTAATATGATCGA
||||  ||||||||||||    ||| ||||  |||| ||||||||||||    |  |    | ||| ||||  |
TGAAACGAAAATGGGGAGG---ATGGAGTATCTGGCTATGAAAACTGATCCC--GAACCAACCCAAT-TGATTAA
****Alignment****
sequence: gi|255562758|ref|XM_002522339.1| Ricinus communis COR413-PM2, putative, mRNA
length: 835
```

# More functions

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

```
>>> import Bio.Blast.Record
>>> help(Bio.Blast.Record)
Help on module Bio.Blast.Record in Bio.Blast:

NAME
    Bio.Blast.Record - Record classes to hold BLAST output.

FILE
    d:\python27\lib\site-packages\bio\blast\record.py

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

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

# Parsing Swiss-Prot files and ENSEMBLE records

# Bio.SeqIO

- **Sequence I/O**
  - Parsing or Reading Sequences
  - Writing Sequence Files

A simple interface for working with assorted file formats in a uniform way

```
>>>from Bio import SeqIO
>>>help(SeqIO)
```

# `SeqIO.parse()`

Reads in sequence data as `SeqRecord` objects

**It expects two arguments:**

- An object (called **handle**) to read the data. It can be:
  - a file opened for reading
  - the output from a command line program
  - data downloaded from the internet

- A lower case string specifying the sequence format (see http://biopython.org/wiki/SeqIO for a full listing of supported formats).

**The object returned by `SeqIO.parse()` is an iterator which returns `SeqRecord` objects**

```
>>> from Bio import SeqIO
>>> handle = open("P05480.fasta")
>>> for seq_rec in SeqIO.parse(handle,
"fasta"):
...     print seq_rec.id
...     print repr(seq_rec.seq)
...     print len(seq_rec)
...
sp|P05480|SRC_MOUSE
Seq('MGSNKSKPKDASQRRRSLERGPSA...ENL',
SingleLetterAlphabet())
541
>>> handle.close()
```

```
from Bio import SeqIO

handle = open("1293613.gbk")

record =  SeqIO.parse(handle, "genbank")

for seq_rec in record:
    print seq_rec.id
    print str(seq_rec.seq)
    print len(seq_rec)

handle.close()
```

```
from Bio import SeqIO

handle = open("1293613.gbk")

for seq_rec in SeqIO.parse(handle, "genbank"):
    print seq_rec.id
    print repr(seq_rec.seq)
    print len(seq_rec)

handle.close()
```

## Iterating over the records in a multipe sequence file

```
>>> from Bio import SeqIO
>>> handle = open("SwissProt-Human.fasta")
>>> all_rec = SeqIO.parse(handle,"fasta")
>>> for rec in all_rec:
...     print rec.id
...
sp|P31946|1433B_HUMAN
sp|P62258|1433E_HUMAN
sp|Q04917|1433F_HUMAN
sp|P61981|1433G_HUMAN
sp|P31947|1433S_HUMAN
sp|P27348|1433T_HUMAN
sp|P63104|1433Z_HUMAN
>>> handle.close()
```

## Parsing sequences from the net

*Handles are not always from files*

```
import urllib2

handle = urllib2.urlopen("http://
www.uniprot.org/uniprot/B2TYV6.fasta")

F = handle.read()

print F
```

## Parsing sequences from the net

*Handles are not always from files*

Parsing SwissProt sequence from the Internet

```
from Bio import ExPASy
from Bio import SeqIO

handle = ExPASy.get_sprot_raw("P04637")

seq_record = SeqIO.read(handle,"swiss")

handle.close()

print seq_record.id
print seq_record.name
print seq_record.description:w
```

## Parsing sequences from the net

Parsing SwissProt sequences from the Internet

```
from Bio import ExPASy
from Bio import SeqIO

AC_list = ['P04637', 'P0CQ42', 'Q13671']
records = []
for ac in AC_list:
    handle = ExPASy.get_sprot_raw(ac)
    record = SeqIO.read(handle, "swiss")
    records.append(record)
out = open('myfile.fasta','w')
for rec in records:
    print rec.id
    fasta = Bio.SeqIO.write(rec, out, "fasta")
out.close()
```

## Parsing sequences from the net

Parsing GenBank records from the Internet

```
from Bio import Entrez
Entrez.email = "allegra.via@uniroma1.it"
from Bio import SeqIO

handle = Entrez.efetch(db =
"nucleotide",rettype = "fasta",id = "6273291")
seq_record = SeqIO.read(handle,"fasta")

handle.close()

print seq_record.description
print seq_record.id
```

# Indexing really large files

`Bio.SeqIO.index()` returns a dictionary without keeping everything in memory.

It works fine even for million of sequences

The main drawback is less flexibility: it is read-only

```
>>> from Bio import SeqIO
>>> recs_dict = SeqIO.index("ncbi_gene.fasta", "fasta")
>>> len(recs_dict)
34
>>> recs_dict.keys()
['M69013', 'M69012', 'AJ580952', 'J03005', 'J03004', 'L13858',
'L04510', 'M94539', 'M19650', 'A10421', 'AJ002990', 'A06663',
'A06662', 'S62035', 'M57424', 'M90035', 'A06280', 'X95521',
'X95520', 'M28269', 'S50017', 'L13857', 'AJ345013', 'M31328',
'AB038040', 'AB020593', 'M17219', 'DQ854814', 'M27543', 'X62025',
'M90043', 'L22075', 'X56614', 'M90027']
>>> print recs_dict['M57424']
ID: M57424
Name: M57424
Description: M57424 Human adenine nucleotide translocator-2 (ANT-2)
gene, complete cds. : Location:1..1000
Number of features: 0
Seq('gagctctggaatagaatacagtagaggcatcatgctcaaagagagtagcagatg...agc',
SingleLetterAlphabet())
```

# Writing sequence files

## `Bio.SeqIO.write()`

This function takes three arguments:

1. SeqRecord objects
2. a handle to write to
3. a sequence format

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

Rec1 = SeqRecord(Seq("ACCA"), id="1", description="")
Rec2 = SeqRecord(Seq("CDRFAA"), id="2", description="")
Rec3 = SeqRecord(Seq("GRKLM"), id="3", description="")

My_records = [Rec1, Rec2, Rec3]

handle = open("MySeqs.fas","w")

SeqIO.write(My_records, handle, "fasta")

handle.close()
```

## Converting between sequence file formats

You can do file conversion by combining `Bio.SeqIO.parse()` and `Bio.SeqIO.write()`

```
from Bio import SeqIO

In_handle = open ("1293613.gbk", "r")
Out_handle = open("1293613.fasta", "w")

records = SeqIO.parse(In_handle, "genbank")
SeqIO.write(records, Out_handle, "fasta")

In_handle.close()
Out_handle.close()
```

## Search the Entrez nucleotide database by keyword(s)

You can do it using a combination of `Entrez.esearch()` and `Entrez.efetch()`

```
from Bio import Entrez
from Bio import SeqIO

Entrez.email = "allegra.via@uniroma1.it"
# search entries by keywords
handle = Entrez.esearch(db="nucleotide", term="Homo
sapiens AND mRNA AND MapK")
records = Entrez.read(handle)
print records['Count']

FewRecords = records['IdList'][0:3]
# retrieve and parse entries from the database
handle = Entrez.efetch(db = "nucleotide",retmode="xml",id
= FewRecords)

records = Entrez.parse(handle)
for record in records:
    print record.keys()
    print record['GBSeq_primary-accession']
    print record['GBSeq_sequence']
    print len(records)
```

## Search the Entrez Pubmed database by keyword(s)

You can do it using a combination of `Entrez.esearch()` and `Entrez.efetch()`

```
from Bio import Entrez
from Bio import Medline

keyword = "PyCogent"
# search publications in PubMed
Entrez.email = "allegra.via@uniroma1.it"
handle = Entrez.esearch(db="pubmed", term=keyword)
record = Entrez.read(handle)
pmids = record['IdList']
print pmids

# retrieve Medline entries from PubMed
handle = Entrez.efetch(db="pubmed", id=pmids,
rettype="medline", retmode="text")
medline_records = Medline.parse(handle)
records = list(medline_records)

n = 1
for record in records:
    if keyword in record["TI"]: print n, ')', record["TI"]
```

## Search the Entrez protein database by keyword(s)

You can do it using a combination of `Entrez.esearch()` and `Entrez.efetch()`

```
from Bio import Entrez

Entrez.email = "allegra.via@gmail.com"

# search entries by keywords
handle = Entrez.esearch(db = "protein", term = "Human AND
cancer AND p21 AND Map kinase", rettype = "fasta")
records = Entrez.read(handle)

FewRecords = records['IdList'][0:3]
print FewRecords
ID_list = ",".join(FewRecords)

# retrieve and parse entries from the database
handle = Entrez.efetch(db = "protein", id = ID_list,
rettype="fasta", retmode="xml")

records= Entrez.read(handle)
rec = list(records)
print rec[0].keys()
print rec[0]['TSeq_defline']
```

# Running BLAST

# Running Blast

## Locally

From the UNIX shell

Blast command line

From a script

Using `os.system()`

Using `Biopython`

## Over the Internet

Using `Bio.Blast.NCBIWWW`

Using Web Programming

Using a Web Browser

# Running BLAST locally

```
blastProgram –query Query –out OutFile -db Database
```

**Using the UNIX shell command line**

```
$blastp -query P05480.fasta -out blast_output -db nr.00
```

# Running BLAST locally

## From a script

**Using os.system()**

```
import os
S = "blastp -query P05480.fasta -out \
blast_output -db nr.00"
os.system(S)
```

# Running BLAST locally

From a script **Using Biopython**

```
import os

from Bio.Blast.Applications import NcbiblastpCommandline
cline = NcbiblastpCommandline(query = "P05480.fasta",
                              db = "nr.00",
                              out = "Blast.out",
                              evalue = 0.001)
print cline
os.system(str(cline))
```

# Running BLAST locally

From a script **Using Biopython**

```
import os

# outfmt = 5 generates the output in XML format:
cline = NcbiblastpCommandline(query = "P05480.fasta",
                              db = "nr.00",
                              out = "Blast.xml",
                              evalue = 0.001,
                              outfmt = 5)
print cline
os.system(str(cline))
```

# Running BLAST locally

From a script **Using Biopython**

```
# Run BLAST+ with nucleotide sequences.In order to run the
# following command, you need to download or format
# a nucleotide database.
# my_gene.fasta must be in the directory where you
# run the script.

from Bio.Blast.Applications import NcbiblastnCommandline
cline = NcbiblastnCommandline(query = "my_gene.fasta",
                              db = "nt", strand = "plus",
                              evalue = 0.001,
                              out = "Blast.xml",
                              outfmt = 5)
print cline

os.system(str(cline))
```

## Running BLAST over the Internet

## and saving the output to a file

`Bio.Blast.NCBIWWW`

```
from Bio.Blast import NCBIWWW

result_handle = NCBIWWW.qblast("blastn","nr","8332116")

save_file = open("qblast_blastn.out", "w")
save_file.write(result_handle.read())
save_file.close()

result_handle.close()
```

**Some useful parameters**:

| | |
|---|---|
| **program** | blastn, blastp, blastx, tblastn, or tblastx (lower case) |
| **database** | Which database to search against (e.g. "nr"). |
| **sequence** | The sequence to search. |
| ncbi_gi | TRUE/FALSE whether to give 'gi' identifier. |
| descriptions | Number of descriptions to show. Def 500. |
| alignments | Number of alignments to show. Def 500. |
| expect | An expect value cutoff. Def 10.0. |
| matrix_name | Specify an alt. matrix (PAM30, PAM70, BLOSUM80, BLOSUM45). |
| filter | "none" turns off filtering. Default no filtering |
| **format_type** | "HTML", "Text", "ASN.1", or "XML". Def. "XML". |
| entrez_query | Entrez query to limit Blast search |
| hitlist_size | Number of hits to return. Default 50 |
| megablast | TRUE/FALSE whether to use MEga BLAST algorithm (blastn only) |
| service | plain, psi, phi, rpsblast, megablast (lower case) |

Save the output locally in xml format

```
from Bio.Blast import NCBIWWW

result_handle =
NCBIWWW.qblast("blastn","nr","8332116",
format_type="XML")

save_file = open("qblast_blastn.xml", "w")
save_file.write(result_handle.read())
save_file.close()

result_handle.close()
```

## Running BLAST and saving the output to a XML file

```
from Bio.Blast import NCBIWWW

result_handle = NCBIWWW.qblast("blastn","nr","8332116",
format_type = "XML")

save_file = open("qblast_blastn.out", "w")
save_file.write(result_handle.read())
save_file.close()

result_handle.close()
```

## Parsing the BLAST output

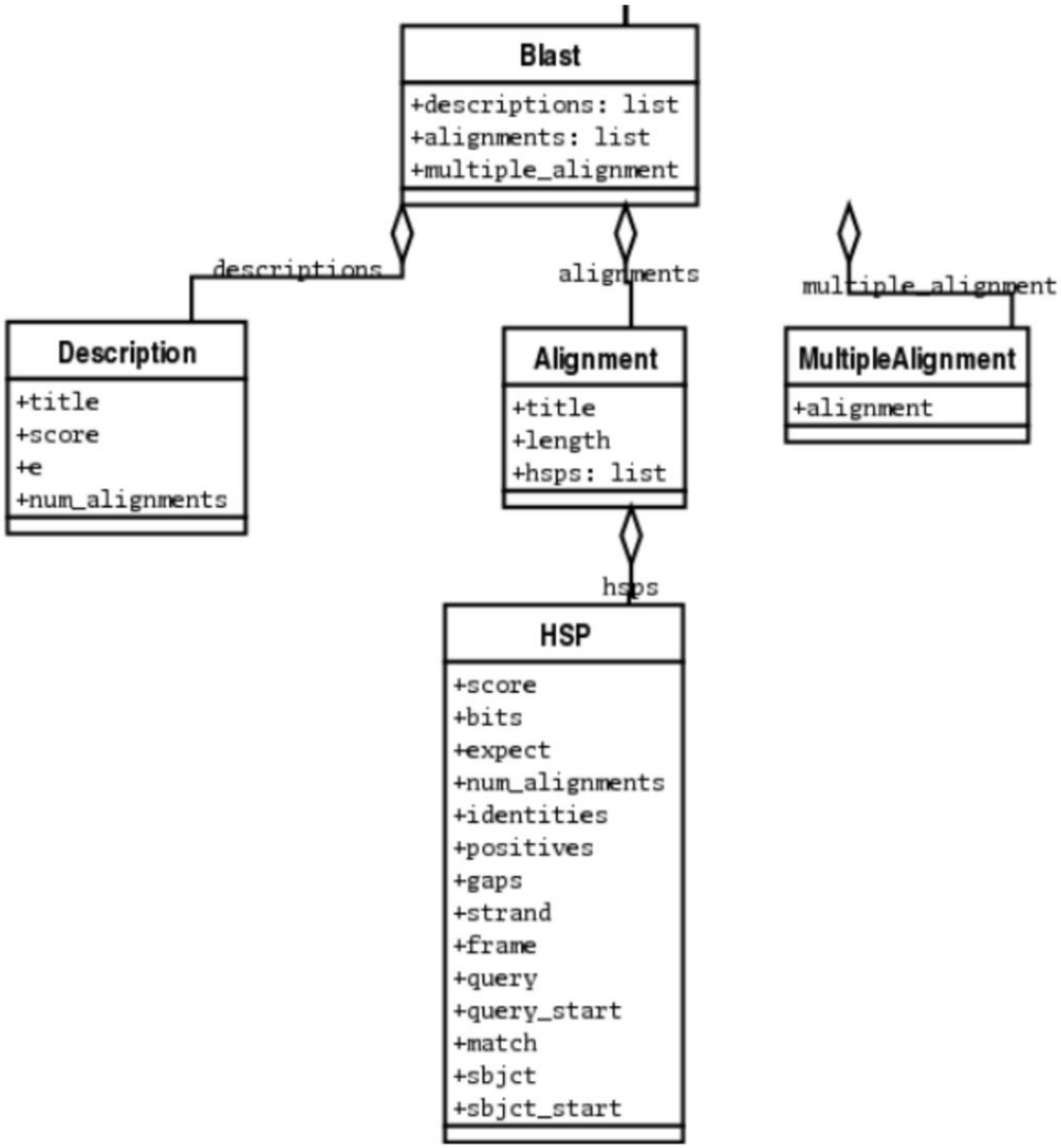
You can get BLAST output in XML in various ways: for the parser it does not matter how the output was generated as long as it is in XML format

- Use Biopython to run BLAST over the internet
- Use Biopython to run BLAST locally
- Do the BLAST search yourself on the NCBI site through your web browser, and then save the results (choose XML format for the output)
- Run BLAST locally without using Biopython, and save the output in a file (choose XML format for the output file)

# Parsing the BLAST output

## Bio.Blast.NCBIXML

```
from Bio.Blast import NCBIXML

result_handle = open("qblast_blastn.out")

#If you expect a single BLAST result
blast_record = NCBIXML.read(result_handle)

#If you have lots of results
blast_records = NCBIXML.parse(result_handle)
```


Blast
+descriptions: list
+alignments: list
+multiple_alignment
descriptions
alignments
multiple_alignment
Description
+title
+score
+e
+num_alignments
Alignment
+title
+length
+hsps: list
MultipleAlignment
+alignment
hsps
HSP
+score
+bits
+expect
+num_alignments
+identities
+positives
+gaps
+strand
+frame
+query
+query_start
+match
+sbjct
+sbjct_start

```
>>> from Bio.Blast import NCBIXML
>>> result_handle = open("qblast_blastn.out")
>>> blast_record = NCBIXML.read(result_handle)
>>> for alignment in blast_record.alignments:
...     for hsp in alignment.hsps:
...             print hsp.score
...
370.0
354.0
354.0
336.0
292.0
292.0
278.0
276.0
```

```
>>> from Bio.Blast import NCBIXML
>>> blast_results = open("qblast_blastn.out")
>>> blast_records = NCBIXML.parse(blast_results)
>>> for blast_rec in blast_records:
...   for alignment in blast_rec.alignments:
...         for hsp in alignment.hsps:
...               print hps.score, hps.expect
...
370.0 6.67609e-88
354.0 1.47051e-83
354.0 1.47051e-83
336.0 1.13052e-78
292.0 9.91696e-67
292.0 9.91696e-67
278.0 6.25828e-63
276.0 2.18435e-62
264.0 3.94941e-59
```

The Biopython module name is `Bio`

It must be downloaded and installed
(`http://biopython.org/wiki/Download`)

You need to install `numpy` first

```
>>>import Bio
```

# Phylo IO

- Phylo.read() reads a tree with exactly one tree
- If you have many trees use a loop over the returned object of Phylo.parse()
- Write to file using Phylo.write(treeObj,format)
  - Popular formats: “nwk”, “xml”
- Convert tree formats using Phylo.convert
  - Phylo.convert("tree1.xml", "phyloxml", "tree1.dnd", "newick")