

# Scientific Programming

## Practical 10

---

### Introduction

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

# Tutoring

Dear QCB students.

a tutoring service has been set up for you, it will be held by Gabriele Masina ( [gabriele.masina@studenti.unitn.it](mailto:gabriele.masina@studenti.unitn.it) ) on:

- **Mondays 16.30-18.30**

- **Thursdays 11.30-13.30.**

Starting from Monday, October 26th included.

These sessions will take place via Zoom, the details for the connection follow:

<https://unitn.zoom.us/j/86903894307?pwd=bGVqV1ExcDRPb1hqWWpPbkFXZlh1QT09>

**Meeting ID: 869 0389 4307**

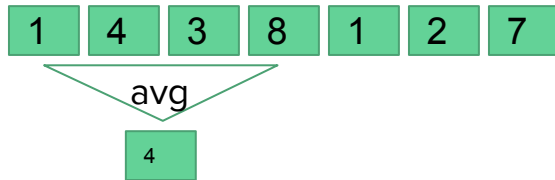
**Passcode: 076411**

Make the most of this additional chance to learn python!

# Ex 6

6. Implement a function `movingAvg(A, n)` where A is a numpy one dimensional array and n is the window size on which computing the average, that outputs the moving average over a numpy one dimensional array. Es. A = [1,2,3,4,5] `movingAvg(A,2) = [1.5, 2.5, 3.5, 4.5]`, while `movingAvg(A,3) = [2, 3, 4]` without using for loops. Hint: use cumsum and clever slicing. Assess the smoothing effect of the moving average by creating a numpy array containing a sinusoidal wave with some additional noise (i.e. use `np.sin` and `np.random.rand`) and testing several values of the window size n.

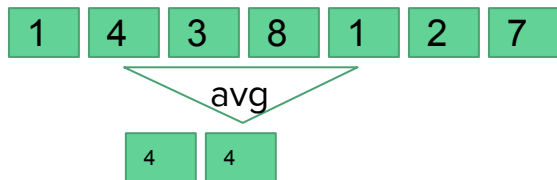
Example (win:4):



# Ex 6

6. Implement a function `movingAvg(A, n)` where A is a numpy one dimensional array and n is the window size on which computing the average, that outputs the moving average over a numpy one dimensional array. Es. A = [1,2,3,4,5] `movingAvg(A,2) = [1.5, 2.5, 3.5, 4.5]`, while `movingAvg(A,3) = [2, 3, 4]` without using for loops. Hint: use cumsum and clever slicing. Assess the smoothing effect of the moving average by creating a numpy array containing a sinusoidal wave with some additional noise (i.e. use `np.sin` and `np.random.rand`) and testing several values of the window size n.

Example (win:4):

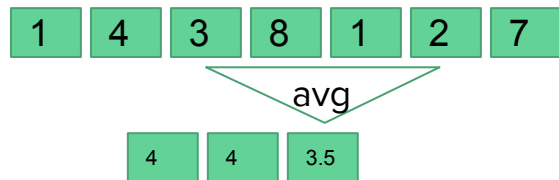


[4. 4. 3.5 4.5]

## Ex 6

6. Implement a function `movingAvg(A, n)` where A is a numpy one dimensional array and n is the window size on which computing the average, that outputs the moving average over a numpy one dimensional array. Es. A = [1,2,3,4,5] `movingAvg(A,2) = [1.5, 2.5, 3.5, 4.5]`, while `movingAvg(A,3) = [2, 3, 4]` without using for loops. Hint: use cumsum and clever slicing. Assess the smoothing effect of the moving average by creating a numpy array containing a sinusoidal wave with some additional noise (i.e. use `np.sin` and `np.random.rand`) and testing several values of the window size n.

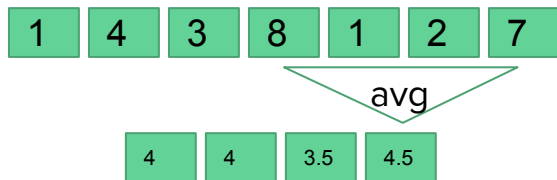
Example (win:4):



# Ex 6

6. Implement a function `movingAvg(A, n)` where A is a numpy one dimensional array and n is the window size on which computing the average, that outputs the moving average over a numpy one dimensional array. Es.  $A = [1, 2, 3, 4, 5]$  `movingAvg(A, 2) = [1.5, 2.5, 3.5, 4.5]`, while `movingAvg(A, 3) = [2, 3, 4]` without using for loops. Hint: use `cumsum` and clever slicing. Assess the smoothing effect of the moving average by creating a numpy array containing a sinusoidal wave with some additional noise (i.e. use `np.sin` and `np.random.rand`) and testing several values of the window size n.

Example (win:4):



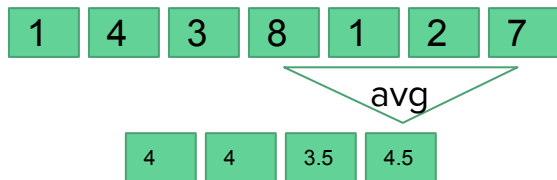
Cumulative sum:



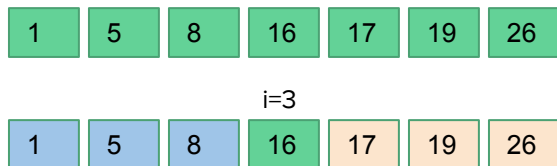
# Ex 6

6. Implement a function `movingAvg(A, n)` where A is a numpy one dimensional array and n is the window size on which computing the average, that outputs the moving average over a numpy one dimensional array. Es.  $A = [1, 2, 3, 4, 5]$  `movingAvg(A, 2) = [1.5, 2.5, 3.5, 4.5]`, while `movingAvg(A, 3) = [2, 3, 4]` without using for loops. Hint: use cumsum and clever slicing. Assess the smoothing effect of the moving average by creating a numpy array containing a sinusoidal wave with some additional noise (i.e. use `np.sin` and `np.random.rand`) and testing several values of the window size n.

Example (win:4):



Cumulative sum:

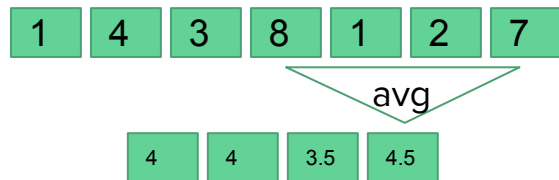


**Clever bit:** when we move to the right with the window we need to disregard (i.e. subtract) the blue elements one after the other

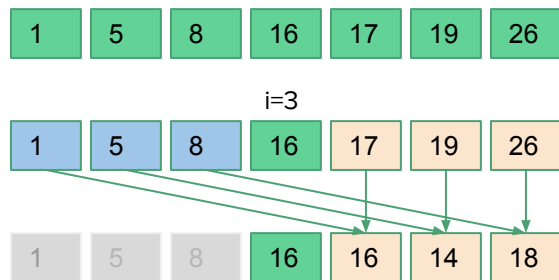
# Ex 6

6. Implement a function `movingAvg(A, n)` where A is a numpy one dimensional array and n is the window size on which computing the average, that outputs the moving average over a numpy one dimensional array. Es. A = [1,2,3,4,5] `movingAvg(A,2) = [1.5, 2.5, 3.5, 4.5]`, while `movingAvg(A,3) = [2, 3, 4]` without using for loops. Hint: use cumsum and clever slicing. Assess the smoothing effect of the moving average by creating a numpy array containing a sinusoidal wave with some additional noise (i.e. use `np.sin` and `np.random.rand`) and testing several values of the window size n.

Example (win:4):



Cumulative sum:



**Clever bit:** when we move to the right with the window we need to disregard (i.e. subtract) the blue elements one after the other

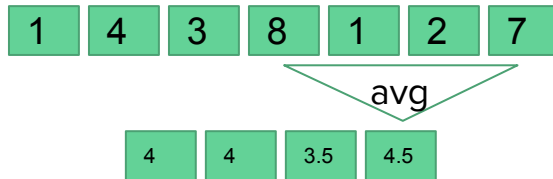
Let's subtract them after the starting point (i=3)



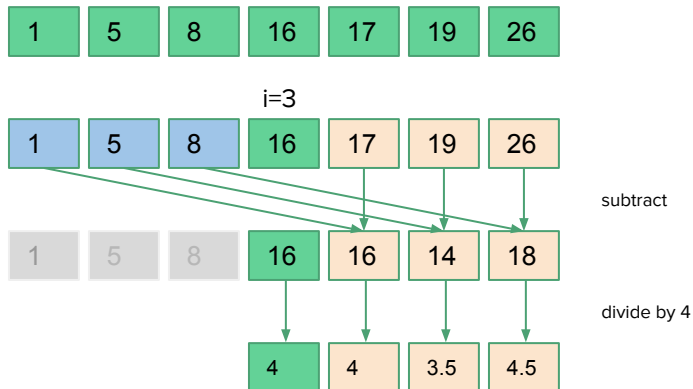
# Ex 6

6. Implement a function `movingAvg(A, n)` where A is a numpy one dimensional array and n is the window size on which computing the average, that outputs the moving average over a numpy one dimensional array. Es. A = [1,2,3,4,5] `movingAvg(A,2) = [1.5, 2.5, 3.5, 4.5]`, while `movingAvg(A,3) = [2, 3, 4]` without using for loops. Hint: use cumsum and clever slicing. Assess the smoothing effect of the moving average by creating a numpy array containing a sinusoidal wave with some additional noise (i.e. use `np.sin` and `np.random.rand`) and testing several values of the window size n.

Example (win:4):



Cumulative sum:



**Clever bit:** when we move to the right with the window we need to disregard (i.e. subtract) the blue elements one after the other

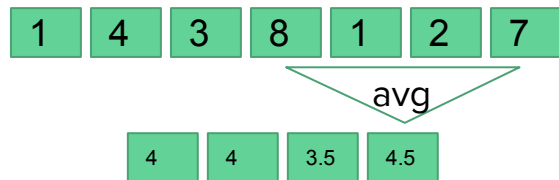
Let's subtract them after the starting point ( $i=3$ )

Finally, let's compute the mean value (i.e. divide by 4)

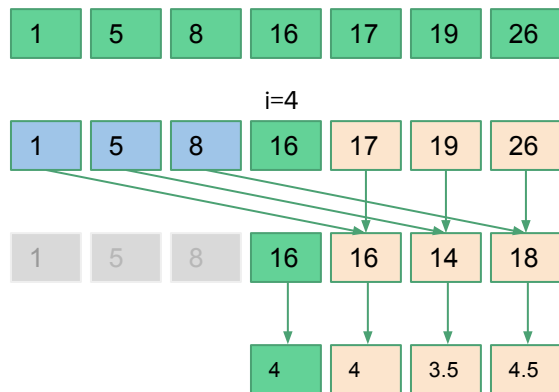
# Ex 6

6. Implement a function `movingAvg(A, n)` where A is a numpy one dimensional array and n is the window size on which computing the average, that outputs the moving average over a numpy one dimensional array. Es. A = [1,2,3,4,5] | `movingAvg(A,2)` = [1.5, 2.5, 3.5, 4.5], while `movingAvg(A,3)` = [2, 3, 4] without using for loops. Hint: use `cumsum` and clever slicing. Assess the smoothing effect of the moving average by creating a numpy array containing a sinusoidal wave with some additional noise (i.e. use `np.sin` and `np.random.rand`) and testing several values of the window size n.

Example:



Cumulative sum:



```
def movingAvg(v, n):  
    """computes the moving average of n values in the array v"""  
    cs = np.cumsum(v)  
    #print(cs)  
    #print("\t", cs[n:])  
    #print("\t", cs[:-n])  
    cs[n:] = cs[n:] - cs[:-n]  
    #print("CS:", cs)  
    return cs[n - 1:] / n  
  
X = np.arange(0, 5, 0.005)  
#B = np.random(100)  
B = np.sin(2*np.pi*X)  
  
#Let's add random numbers uniformly distributed in [0,1)  
B += np.random.random_sample(1000)  
#B += np.random.rand(1000)  
C = movingAvg(B,5)  
D = movingAvg(B,10)  
E = movingAvg(B,50)  
#X = np.arange(0,B.shape[0])  
  
plt.plot(X,B)  
plt.title("No moving average")  
plt.show()  
plt.close()  
plt.plot(np.arange(0,C.shape[0]),C)  
plt.title("Window size: 5")  
plt.show()  
plt.close()  
plt.title("Window size: 10")  
plt.plot(np.arange(0,D.shape[0]),D)  
plt.show()  
plt.close()  
plt.title("Window size: 50")  
plt.plot(np.arange(0,E.shape[0]),E)  
plt.show()
```

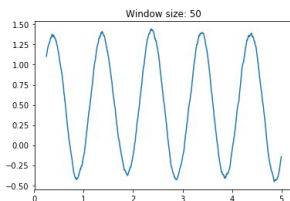
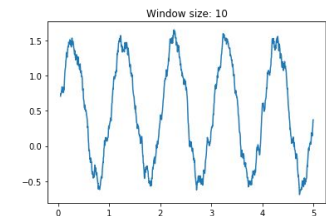
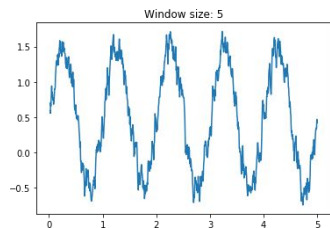
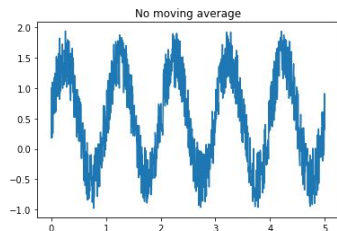
# Ex 6

6. Implement a function `movingAvg(A,n)` where A is a numpy one dimensional array and n is the window size on which computing the average, that outputs the moving average over a numpy one dimensional array. Es.  $A = [1,2,3,4,5]$  | `movingAvg(A,2) = [1.5, 2.5, 3.5, 4.5]`, while `movingAvg(A,3) = [2, 3, 4]` without using for loops. Hint: use `cumsum` and clever slicing. Assess the smoothing effect of the moving average by creating a numpy array containing a sinusoidal wave with some additional noise (i.e. use `np.sin` and `np.random.rand`) and testing several values of the window size n.

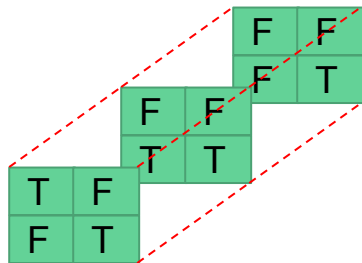
```
def movingAvg(v, n):  
    """computes the moving average of n values in the array v"""  
    cs = np.cumsum(v)  
    #print(cs)  
    #print("\t",cs[n:])  
    #print("\t", cs[:-n])  
    cs[n:] = cs[n:] - cs[:-n]  
    #print("CS:", cs)  
    return cs[n - 1:] / n
```

```
X = np.arange(0, 5, 0.005)  
#B = np.random(100)  
B = np.sin(2*np.pi*X)  
  
#Let's add random numbers uniformly distributed in [0,1)  
B += np.random.random_sample(1000)  
#B += np.random.rand(1000)  
C = movingAvg(B,5)  
D = movingAvg(B,10)  
E = movingAvg(B,50)  
#X = np.arange(0,B.shape[0])
```

```
plt.plot(X,B)  
plt.title("No moving average")  
plt.show()  
plt.close()  
plt.plot(X[4:],C) # X has 1000- 5 -1 elements  
plt.title("Window size: 5")  
plt.show()  
plt.close()  
plt.title("Window size: 10")  
plt.plot(X[9:],D) # X has 1000- 9 -1 elements  
plt.show()  
plt.close()  
plt.title("Window size: 50")  
plt.plot(X[49:],E) # X has 1000- 49 -1 elements  
plt.show()
```



# Axis on 3D arrays



```
vals:  
[[[ True False]  
  [False  True]]  
  
 [[False False]  
  [ True  True]]  
  
 [[False False]  
  [False  True]]]
```

The matrix is 3D

First matrix:

```
[[ True False]  
 [False  True]]
```

First row, all matrices:

```
[[ True False]  
 [False False]  
 [False False]]
```

Second column, all matrices:

```
[[False  True]  
 [False  True]  
 [False  True]]
```

```
import numpy as np
```

```
v1 = [[True, False],[False, True]]  
v2 = [[False, False],[True, True]]  
v3 = [[False, False], [False, True]]  
print("vals:")  
vals = np.array([v1,v2,v3])  
print(vals)  
print("\nThe matrix is {}".format(vals.ndim))  
print("First matrix:")  
print(vals[0,:,:])  
print("\nFirst row, all matrices:")  
print(vals[:,0,:])  
print("\nSecond column, all matrices:")  
print(vals[:, :,1])
```

```
print("\nAXIS=0")  
print("ANY:")  
print(np.any(vals, axis=0))  
print("ALL ")  
print(np.all(vals, axis=0))
```

```
print("\nAXIS=1")  
print("ANY:")  
print(np.any(vals, axis=1))  
print("ALL:")  
print(np.all(vals, axis=1))
```

```
print("\nAXIS=2")  
print("ANY:")  
print(np.any(vals, axis=2))  
print("ALL:")  
print(np.all(vals, axis=2))
```

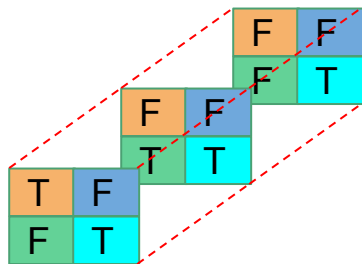
`np.ndarray[M, R, C]`

↑  
matrix (can be : for all)

↑  
row (can be : for all)

↑  
column (can be : for all)

# Axis on 3D arrays



axis = 0

```
AXIS=0
ANY:
[[ True False]
 [ True  True]]
ALL
[[False False]
 [False  True]]
```



```
import numpy as np

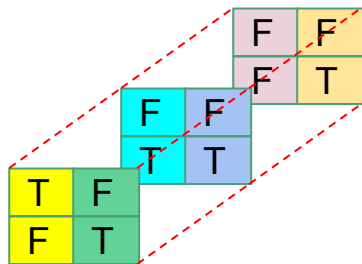
v1 = [[True, False],[False, True]]
v2 = [[False, False],[True, True]]
v3 = [[False, False], [False, True]]
print("vals:")
vals = np.array([v1,v2,v3])
print(vals)
print("\nThe matrix is {}D".format(vals.ndim))
print("First matrix:")
print(vals[0,:,:])
print("\nFirst row, all matrices:")
print(vals[:,0,:])
print("\nSecond column, all matrices:")
print(vals[:, :,1])

print("\n\nAXIS=0")
print("ANY:")
print(np.any(vals, axis=0))
print("ALL ")
print(np.all(vals, axis=0))

print("\n\nAXIS=1")
print("ANY:")
print(np.any(vals, axis=1))
print("ALL:")
print(np.all(vals, axis=1))

print("\n\nAXIS=2")
print("ANY:")
print(np.any(vals, axis=2))
print("ALL:")
print(np.all(vals, axis=2))
```

# Axis on 3D arrays



AXIS=1  
ANY:  
[[ True True]  
 [ True True]  
[False True]]  
ALL:  
[[False False]  
[False False]  
[False False]]



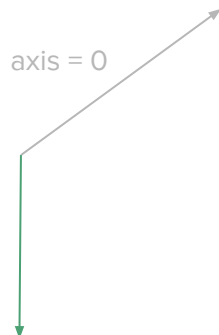
```
import numpy as np

v1 = [[True, False],[False, True]]
v2 = [[False, False],[True, True]]
v3 = [[False, False], [False, True]]
print("vals:")
vals = np.array([v1,v2,v3])
print(vals)
print("\nThe matrix is {}".format(vals.ndim))
print("First matrix:")
print(vals[0,:,:])
print("\nFirst row, all matrices:")
print(vals[:,0,:])
print("\nSecond column, all matrices:")
print(vals[:, :,1])

print("\n\nAXIS=0")
print("ANY:")
print(np.any(vals, axis=0))
print("ALL ")
print(np.all(vals, axis=0))

print("\n\nAXIS=1")
print("ANY:")
print(np.any(vals, axis=1))
print("ALL:")
print(np.all(vals, axis=1))

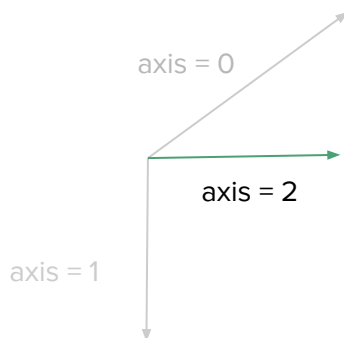
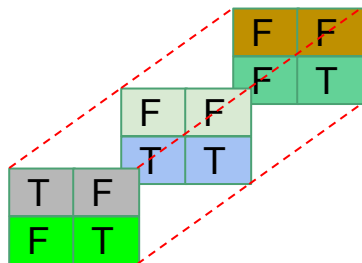
print("\n\nAXIS=2")
print("ANY:")
print(np.any(vals, axis=2))
print("ALL:")
print(np.all(vals, axis=2))
```



axis = 1



# Axis on 3D arrays



**AXIS=2**  
**ANY:**  
[[ True True]  
 [False True]  
 [False True]]  
**ALL:**  
[[False False]  
 [False True]  
 [False False]]



```
import numpy as np

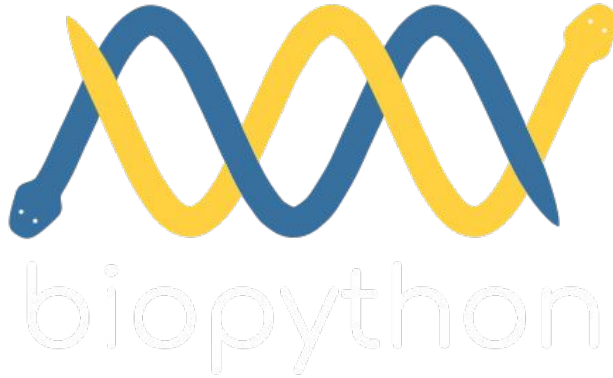
v1 = [[True, False],[False, True]]
v2 = [[False, False],[True, True]]
v3 = [[False, False], [False, True]]
print("vals:")
vals = np.array([v1,v2,v3])
print(vals)
print("\nThe matrix is {}".format(vals.ndim))
print("First matrix:")
print(vals[0,:,:])
print("\nFirst row, all matrices:")
print(vals[:,0,:])
print("\nSecond column, all matrices:")
print(vals[:, :,1])

print("\n\nAXIS=0")
print("ANY:")
print(np.any(vals, axis=0))
print("ALL ")
print(np.all(vals, axis=0))

print("\n\nAXIS=1")
print("ANY:")
print(np.any(vals, axis=1))
print("ALL:")
print(np.all(vals, axis=1))

print("\n\nAXIS=2")
print("ANY:")
print(np.any(vals, axis=2))
print("ALL:")
print(np.all(vals, axis=2))
```

# Biopython

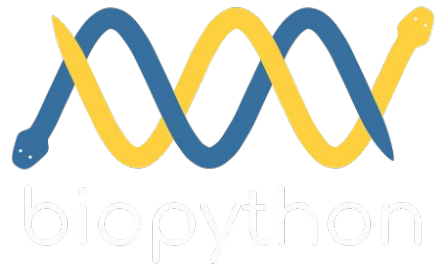


The Biopython Project is an international association of developers of freely available **Python tools for computational molecular biology**.

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



# Biopython



## Biopython:

1. Provides tools to **parse several common bioinformatics formats** (e.g. FASTA, FASTQ, BLAST, PDB, Clustalw, Genbank,..).
2. Provides an **interface towards biological data repositories** (e.g. NCBI, Expasy, Swiss-Prot, .. )
3. Provides an **interface towards some bioinformatic tools** (e.g. clustalw, MUSCLE, BLAST,...)
4. **Implements some tools** like pairwise alignment **and data structures** to deal with biological data.

More material at:

<http://biopython.org/DIST/docs/tutorial/Tutorial.pdf>

# Seq objects

Seq objects are more powerful than strings to deal with sequences and are defined in the module **Bio.Seq**.

They are **immutable objects**. The mutable version is **MutableSeq**.

```
from Bio.Seq import Seq

s = Seq("GATTACATAATA")
dna_seq = Seq("GATTATACGTAC")
print("S:", s)

print("dna_seq:", dna_seq)

my_prot = Seq("MGNAAAACGGSEQE")
print("my_prot:", my_prot)
```

```
S: GATTACATAATA
dna_seq: GATTATACGTAC
my_prot: MGNAAAACGGSEQE
```

# Seq objects

## Seq objects behave like strings.

In the latest release the description of the Alphabet associated to the sequence has been dropped therefore there is no consistency check...

```
from Bio.Seq import Seq

dna_seq = Seq("GATTATACGTAC")
my_prot = Seq("MGNAAAAKKGSEQE")

#Does it really make sense though?!?
print(dna_seq + my_prot)
```

GATTATACGTACMGNAAAAKKGSEQE

# Seq objects

**Seq objects behave like strings**,  
but the consistency of the alphabet is  
checked too.

We can loop through the elements of  
the sequence and perform slicing...

```
from Bio.Seq import Seq

dna_seq = Seq("GATTATACGTACGGCTA")

for base in dna_seq:
    print(base, end = " ")

print("")

sub_seq = dna_seq[4:10]
print(sub_seq)

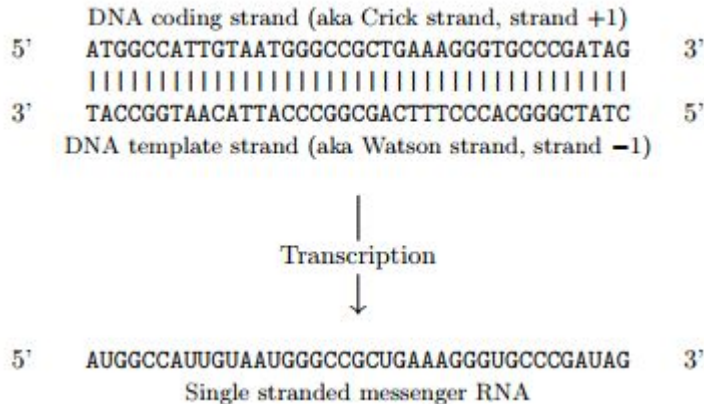
#Let's reverse the string:

print("Reversed: ", dna_seq[::-1])
#from Seq to string:
dna_str = str(dna_seq)
print("As string:", dna_str)
print(type(dna_str))
```

```
G A T T A T A C G T A C G G C T A
ATACGT
Reversed:  ATCGGCATGCATATTAG
As string: GATTATACGTACGGCTA
<class 'str'>
```

# Seq objects

Biopython provides several  
methods working on Seq  
objects  
(remember Seq are immutable!)



General methods (return **int** and **Seq** objects):

`Seq.count(s)` : counts the number of times s appears in the sequence;

`Seq.upper()` : makes the sequence of the object Seq in upper case

`Seq.lower()` : makes the sequence of the object Seq in lower case

Only for DNA/RNA (return **Seq** objects):

`Seq.complement()` to complement the sequence

`Seq.reverse_complement()` to reverse complement the sequence.

`Seq.transcribe()` transcribes the DNA into mRNA

`Seq.back_transcribe()` back transcribes mRNA into DNA

`Seq.translate()` translates mRNA or DNA into proteins

Other functions are in **SeqUtils**

(**ex. use** from Bio.SeqUtils import molecular\_weight):

`SeqUtils.GC(Seq)` computes GC content

`SeqUtils.molecular_weight(Seq)` computes the molecular weight of the seq

....

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

# Seq objects

Biopython provides several methods working on Seq objects (remember Seq are immutable!)

```
from Bio.Seq import Seq

my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")

print("Original sequence:\t{}".format(my_seq) )
comp = my_seq.complement()
print("")
print("Complement:\t\t{}".format(comp))
print("")
revcomp = my_seq.reverse_complement()
print("Reverse complement:\t{}".format(revcomp))
```

Original sequence:	GATCGATGGGCCTATATAGGATCGAAAATCGC
Complement:	CTAGCTACCCGGATATATCCTAGCTTTTAGCG
Reverse complement:	GCGATTTTCGATCCTATATAGGCCCATCGATC

	DNA coding strand (aka Crick strand, strand +1)	
5'	ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG	3'
3'	TACCGGTAACATTACCCGGCGACTTTCCACGGGCTATC	5'
	DNA template strand (aka Watson strand, strand -1)	

↓  
Transcription  
↓

5'	AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG	3'
	Single stranded messenger RNA	

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

# Seq objects

Biopython provides several methods working on Seq objects  
(remember Seq are immutable!)

```
from Bio.Seq import Seq

coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG")
print(coding_dna)

mrna = coding_dna.transcribe()
print(mrna)
print("")
print("... and back")
print(mrna.back_transcribe())
print("")
print("Translation to protein:")
prot = mrna.translate()
print(prot)
print("")
print("Up to first stop:")
print(mrna.translate(to_stop = True))
print("")
print("Mitochondrial translation: (TGA is W!)")
mit_prot = mrna.translate(table=2)
print(mit_prot)
#The following produces a translation error!
#print("RE-Translated protein: {}".format(prot.translate()))
```

```
ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG
AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG
```

```
... and back
ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG
```

```
Translation to protein:
MAIVMGR*KGAR*
```

```
Up to first stop:
MAIVMGR
```

```
Mitochondrial translation: (TGA is W!)
MAIVMGRWKGAR*
```

# Sequence annotations

The **SeqRecord** object is used to store annotations associated to sequences. They might provide:

1. `SeqRecord.seq` : the sequence (the Seq object)
2. `SeqRecord.id` : the identifier of the sequence, typically an accession number
3. `SeqRecord.name` : a "common" name or identifier sometimes identical to the accession number
4. `SeqRecord.description` : a human readable description of the sequence
5. `SeqRecord.letter_annotations` : a per letter annotation using a restricted dictionary (e.g. quality)
6. `SeqRecord.annotations` : a dictionary of unstructured annotation (e.g. organism, publications,...)
7. `SeqRecord.features` : a list of SeqFeature objects with more structured information (e.g. genes pos).
8. `SeqRecord.dbxrefs` : a list of database cross references.



# Sequence annotations

Read a fasta file [NC005816.fna](#) containing the whole sequence for *Yersinia pestis* biovar *Microtus* str. 91001 plasmid pPCP1 and retrieve some information about the sequence.

```
>gi|45478711|ref|NC_005816.1| Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence
```

```
TGTAACGAACGGTGCAATAGTGATCCACACCAACGCCTGAAATCAGATCCAGGGGTAATCTGCTCTCC  
TGATTCAGGAGAGTTTATGGTCACTTTTGAGACAGTTATGAAATTAATCCTGCACAAGCAGGGAATG  
AGTAGCCGGGCGATTGCCAGAGAAGCTGGGGATCTCCGCAATACCGTTAAACGTTATTTGCAGGCAAAAT  
CTGAGCCGCCAAAATATACGCCGCGACCTGCTGTTGCTTCACTCCTGGATGAATACCGGGATTATATTG  
TCAACGCATCGCCGATGCTCATCTTACAAAATCCCGGCAACGGTAATCGCTCGCGAGATCAGAGACCAG  
GGATATCGTGGCGGAATGACCATTCTCAGGGCATTCACTCGTTCTCTCGGTTCTCAGGAGCAGGAGC  
CTGCCGTTTCGGTTTCAAACTGAACCCGGACGACAGATGCAGTTGACTGGGGCACTATGCGTAATGGTCG  
CTCACCGCTTCACGTGTTGCTTCTCGGATACAGCCGAATGCTGTACATCGAATCACTGACAAT  
ATGCGTTATGACACGCTGGAGACCTGCCATCGTAATGCGTTCGCTTCTTTGGTGGTGTGCCGCGGAAG  
TGTTGTATGACAATATGAAAACGTGG....|
```

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

← → ↻ [https://www.ncbi.nlm.nih.gov/nuccore/NC\\_005816](https://www.ncbi.nlm.nih.gov/nuccore/NC_005816)

NCBI Resources How To

Nucleotide  NC005816  
Advanced

[Learn more](#) about upcoming changes to the Nucleotide, EST, and GSS databases.

GenBank Send to: ▼

### Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence

NCBI Reference Sequence: NC\_005816.1

[FASTA](#) [Graphics](#)

[Go to: ▼](#)

LOCUS	NC_005816	9609 bp	DNA	circular	CON 11-JAN-2018
DEFINITION	Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.				
ACCESSION	NC_005816				
VERSION	NC_005816.1				
DBLINK	BioProject: <a href="#">PRJNA224116</a> BioSample: <a href="#">SAMN02602970</a> Assembly: <a href="#">GCF_000007885.1</a>				
KEYWORDS	RefSeq.				
SOURCE	Yersinia pestis biovar Microtus str. 91001				
ORGANISM	<a href="#">Yersinia pestis biovar Microtus str. 91001</a> Bacteria; Proteobacteria; Gammaproteobacteria; Enterobacterales; Yersiniaceae; Yersinia.				
REFERENCE	1 (bases 1 to 9609)				
AUTHORS	Zhou,D., Tong,Z., Song,Y., Han,Y., Pei,D., Pang,X., Zhai,J., Li,M., Cui,B., Qi,Z., Jin,L., Dai,R., Du,Z., Wang,J., Guo,Z., Wang,J., Huang,P. and Yang,R.				
TITLE	Genetics of metabolic variations between Yersinia pestis biovars				

# Sequence annotations

Read a fasta file [NC005816.fna](#) containing the whole sequence for *Yersinia pestis* biovar *Microtus* str. 91001 plasmid pPCP1 and retrieve some information about the sequence.

```
ID: gi|45478711|ref|NC_005816.1|
Name: gi|45478711|ref|NC_005816.1|
Description: gi|45478711|ref|NC_005816.1| Yersinia
pestis biovar Microtus str. 91001 plasmid pPCP1,
complete sequence
Number of features: 0
Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGAT
CCAGG...CTG', SingleLetterAlphabet())
```

```
Sequence [first 30 bases]:
TGTAACGAACGGTGCAATAGTGATCCACAC
```

```
The id:
gi|45478711|ref|NC_005816.1|
```

```
The description:
gi|45478711|ref|NC_005816.1| Yersinia pestis biovar
Microtus str. 91001 plasmid pPCP1, complete sequence
```

```
The record is a: <class 'Bio.SeqRecord.SeqRecord'>
```

```
from Bio import SeqIO

record =
SeqIO.read("file_samples/NC_005816.fna",
"fasta")

print(record)
print("")
print("Sequence [first 30 bases]:")
print(record.seq[0:30])
print("")
print("The id:")
print(record.id)
print("")
print("The description:")
print(record.description)
print("")
print("The record is a: ", type(record))
```

# SeqIO.parse

The `Bio.SeqIO` module aims to provide a simple way to work with several different sequence file formats

The method `Bio.SeqIO.parse` is used to parse some sequence data into a `SeqRecord` iterator. In particular, the basic syntax is:

```
SeqRecordIterator = Bio.SeqIO.parse(filename, file_format)
```

where `filename` is typically an open handle to a file and `file_format` is a lower case string describing the file format. Possible options include **fasta**, **fastq-illumina**, **abi**, **ace**, **clustal**... all the

Note that `Bio.SeqIO.parse` returns an iterator, therefore it is possible to manually fetch one `SeqRecord` after the other with the `next(iterator)` method.

Formats available:

<https://biopython.org/wiki/SeqIO>

**WARNING:** When dealing with very large FASTA or FASTQ files, the overhead of working with all these objects can make scripts too slow. In this case `SimpleFastaParser` and `FastqGeneralIterator` parsers might be better as they return just a tuple of strings for each record.

# SeqIO

**Example:** Let's read the first 3 entries of the .fasta file [contigs82.fasta](#) printing off the length of the sequence and the first 50 bases of each sequence followed by "...".

SeqIO.parse  
returns an iterator,  
we can get the next  
element with  
`next(iterator)`

```
from Bio import SeqIO

seqIterator = SeqIO.parse("file_samples/contigs82.fasta", "fasta")

labels = ["1st", "2nd", "3rd"]
for l in labels:
    seqRec = next(seqIterator)
    print(l, "entry:")
    print(seqRec.id, " has size ", len(seqRec.seq))
    print(seqRec.seq[:50]+"...")
    print("")
```

```
1st entry:
MDC020656.85  has size  2802
GAGGGGTTTAGTTCTCCTACTCGCAAAGCAAAGATACATAAATTTAGAA...
```

```
2nd entry:
MDC001115.177  has size  3118
TGAATGGTGAAAATTAGCCAGAAGATCTTCTCCACATGACATATGCAT...
```

```
3rd entry:
MDC013284.379  has size  5173
TATCGTTTCCTCTGAGTAGAATATCGTTATAACAAGATTTTTTTTTTCT...
```

Do you remember  
all the “pain” to  
parse the header,  
concatenate the  
sequence etc... ?

# SeqIO

With  
SimpleFastaParser...

```
labels = ["1st", "2nd", "3rd"]

with open("file_samples/contigs82.fasta") as cont_handle:
    for l in labels:
        ID, seq = next(SimpleFastaParser(cont_handle))

        print(l, "entry:")
        print(ID, " has size ", len(seq))
        print(seq[:50]+"...")
        print("")
```

1st entry:  
MDC020656.85 has size 2802  
GAGGGGTTTAGTTCCTCATACTCGCAAAGCAAAGATACATAAATTTAGAA...

2nd entry:  
MDC013284.379 has size 5173  
TATCGTTTCCTCTGAGTAGAATATCGTTATAACAAGATTTTTTTTTTCT...

3rd entry:  
MDC018185.241 has size 23761  
AAAACGAGGAAAATCCATCTTGATGAACAGGAGATGCGGAGGAAAAAAT...



# SeqIO

The module `Bio.SeqIO` also has three different ways to allow random access to elements:

1. `Bio.SeqIO.to_dict(file_handle/iterator)` : builds a dictionary of all the SeqRecords keeping them in memory and allowing modifications to the records. **This potentially uses a lot of memory but is very fast;**
2. `Bio.SeqIO.index(filename, file_type)` : builds a sort of read-only dictionary, parses the elements into SeqRecords on demand (i.e. it returns an iterator!). **This method is slower, but more memory efficient;**
3. `Bio.SeqIO.index_db(indexName.idx, filenames, file_format)` : builds a read-only dictionary, but stores ids and offsets on a SQLite3 database. **It is slower but uses less memory.**

**Examples are given on the notes of the practical sheet**

# SeqIO.write

SeqRecords can be written out to files by using

```
N = Bio.SeqIO.write(records,out_filename, file_format)
```

where **records** is a list of the SeqRecords to write, **out\_filename** is the string with the filename to write and **file\_format** is the format of the file to write. **N** is the number of sequences written.

**WARNING:** If you write a file that is already present, `SeqIO.write` will just rewrite it without telling you.

The module `Bio.SeqIO` provides also a way to write sequence records to files in various formats (like fasta, fastq, genbank, pfam...)

**Examples are given on the notes of the practical sheet**

# Multiple sequence alignment

**Multiple Sequence Alignments** are a collection of **multiple sequences** which have been aligned together – usually with the insertion of gap characters, and addition of leading or trailing gaps – such that all the sequence strings have the same length.

```
Q5E940_BOVIN  -----MPREDRATWKSNYFLKIIQLDDDPKCFIVGADNVGSKOMQDIHMSLRGK-AVYLMGKNTMMRKAIRGHLNNH--PALE  76
RLA0_HUMAN   -----MPREDRATWKSNYFLKIIQLDDDPKCFIVGADNVGSKOMQDIHMSLRGK-AVYLMGKNTMMRKAIRGHLNNH--PALE  76
RLA0_MOUSE   -----MPREDRATWKSNYFLKIIQLDDDPKCFIVGADNVGSKOMQDIHMSLRGK-AVYLMGKNTMMRKAIRGHLNNH--PALE  76
RLA0_RAT      -----MPREDRATWKSNYFLKIIQLDDDPKCFIVGADNVGSKOMQDIHMSLRGK-AVYLMGKNTMMRKAIRGHLNNH--PALE  76
RLA0_CHICK    -----MPREDRATWKSNYFMKIIQLDDDPKCFIVGADNVGSKOMQDIHMSLRGK-AVYLMGKNTMMRKAIRGHLNNH--PALE  76
RLA0_RAMSY    -----MPREDRATWKSNYFLKIIQLDDDPKCFIVGADNVGSKOMQDIHMSLRGK-AVYLMGKNTMMRKAIRGHLNNH--PALE  76
Q7ZUG3_BRARE  -----MPREDRATWKSNYFLKIIQLDDDPKCFIVGADNVGSKOMQDIHMSLRGK-AVYLMGKNTMMRKAIRGHLNNH--PALE  76
RLA0_ICTPU    -----MPREDRATWKSNYFLKIIQLDDDPKCFIVGADNVGSKOMQDIHMSLRGK-AVYLMGKNTMMRKAIRGHLNNH--PALE  76
RLA0_DROME    -----HYRENKAAMKAQYFIKVVYLFDEFPPKCFIVGADNVGSKOMQDIHMSLRGK-AVYLMGKNTMMRKAIRGHLNNH--PALE  76
RLA0_DICDI    -----MSGAG-SKREKILFIEKATKLFITTDKMIYAEADVFVGSOLQIKRSIRGI-GAVLMGKNTMIRKVVIRDLADSK--PELD  75
Q54LP0_DICDI  -----MSGAG-SKRENVFIEKATKLFITTDKMIYAEADVFVGSOLQIKRSIRGI-GAVLMGKNTMIRKVVIRDLADSK--PELD  75
RLA0_PLAFB    -----MAKLSKQKKQMYIEKLSLIIQYYSKILIVHVDNYGNGMASVYKSLRGK-ATILMGKNTIRIALTKKNLQAV--DOIE  76
RLA0_SULAC    -----MIGLAVTTTKTAKWYDEVAELTSLKTKNTIIIANIIGFPADKLHEIRKKLRGK-ADIKVTKNLFPNIALKNAG---YDK  79
RLA0_SULTO    -----MRIMAVITQERKIAKWIEVKELEOKLRNHTIIIANIIGFPADKLHEIRKKLRGK-AEIKVTKNLFGIAKNAG---LDVS  80
RLA0_SULSO    -----MKRLALALQKRVASWKELEVKELETLKNSNTILIGNLEGFADKLHEIRKKLRGK-ATIKVTKNLFLKIAKNAG---IDIE  80
RLA0_AERPE    MSVYSIVGQMYKREKIPENKTLMLRELELFSKRVYFLADLTGTPFVVRVYKKEKWKKK-YHMMVAKRRIILBAMKAAGLE---LDDN  86
RLA0_PYRAE    HMLAIGKRRYVTRQIPARKYKIVSEATLLQKQYVYFLFDLHLISIRILHEVYRLARY-GVIRIIRKDLFKIAFTKYVGG---IPAE  85
RLA0_METAC    -----MAEERNHTEHIPQWKDEIENIKELIQSHKVFCHVRIEGILATKIQIRKDLKDV-AVLKVSNTLTERALNQLG---ETIP  78
RLA0_METMA    -----MAEERNHTEHIPQWKDEIENIKELIQSHKVFCHVRIEGILATKIQIRKDLKDV-AVLKVSNTLTERALNQLG---ESIP  78
RLA0_ARCFU    -----MAAVRG-----PPEYVYRAVEEIKRMISSEYVYVIVSFRNVFAGOMKIEREFGK-AEIKVVKNTLLERLADALG---GDYL  75
RLA0_METKA    MAYKAKGQPPSQYEPKVAEMREVEKELKLMDEYENGLDGIAPALQGEIRAKLERDIIIMRBRKTLMLRAIEEKLDER--PELE  88
RLA0_METTH    -----MAHVAEWKKKEVEELAKLISKYPVIALVDVSSMPAYPLSQMRRLIRENGGLLVSRNTLIELAIKKAAKELGKPELE  77
RLA0_METTL    -----MITASEHKIAPWKIEEVNKLKELLKNGQIYALVDMMEYPAQLQGEIRDKIR-GTMTLKMRNTLIEHAIEVAETGNPEFA  82
RLA0_METYA    -----MIDAKSEHKIAPWKIEEVNKLKELLKSNVIALDMMEYPAQLQGEIRDKIR-DQMTLKMRNTLIEHAIEVAETGNPEFA  82
RLA0_METJA    -----METKVAHVAPWKIEEVNKLKELLKSNVIALDMMEYPAQLQGEIRDKIR-DKVKLMRNTLIEHAIEVAETGNPEFA  81
RLA0_PYRAB    -----MAHVAEWKKKEVEELAKLISKYPVIALVDVSSMPAYPLSQMRRLIRENGGLLVSRNTLIELAIKKAAKELGKPELE  77
RLA0_PYRHO    -----MAHVAEWKKKEVEELAKLISKYPVIALVDVSSMPAYPLSQMRRLIRENGGLLVSRNTLIELAIKKAAKELGKPELE  77
RLA0_PYRFU    -----MAHVAEWKKKEVEELAKLISKYPVIALVDVSSMPAYPLSQMRRLIRENGGLLVSRNTLIELAIKKAAKELGKPELE  77
RLA0_PYRKO    -----MAHVAEWKKKEVEELAKLISKYPVIALVDVAGVPAYPLSKMRDKLE-GKALLVSRNTLIELAIKRAAGELGKPELE  76
RLA0_HALMA    -----MSAESERKTETIPWKQEEVDIVMIESYSGVYVNIAGIPBOLQDMRDLHGT-AELVSRNTLIELALDDVDY---DGLE  79
RLA0_HALVO    -----MSESEVRQTEVIPQWKREEVDLVDIESYSGVYVNIAGIPBOLQDMRDLHGT-AELVSRNTLIELALDDVDY---DGLE  79
RLA0_HALSA    -----MSAEQRTEETEEVPWKQEEVDLVDIESYSGVYVNIAGIPBOLQDMRDLHGT-AELVSRNTLIELALDDVDY---DGLE  79
RLA0_THEAC    -----MKESYQQKKELVNIEITRIKASRSVAIVDTAGIRTRQIDIRGKNGK-INLVKIKKLLFPALNLEGD---EKLS  72
RLA0_THEVO    -----MRKINPKKEIVSELAQDITKSKAVAIVDIKGVRTRMODIRAKNHDK-VKIKVVKKLLFPALNLEGD---EKLT  72
RLA0_PICTO    -----MTEPAQWKIDFVKNLENEINSRKVAIVSISKELRNNTFKIKNSIDDK-ARIKVRARLRLALNLEGD---NHIV  72
ruler 1.....10.....20.....30.....40.....50.....60.....70.....80.....90
```

In Biopython, each row is a `SeqRecord` object and alignments are stored in an object `MultipleSeqAlignment`



# Parsing MSAs: AlignIO

The function `Bio.AlignIO.parse()` returns an iterator of `MultipleSeqAlignment` objects that is a collection of `SeqRecords`.

Each `SeqRecord` contains several information like the **ID**, **Name**, **Description**, **Number of features**, **start**, **end** and **sequence**.

In the frequent case that we have to deal with a **single multiple alignment** we will have to use the `Bio.AlignIO.read()` function.

The basic syntax of the two functions:

```
Bio.AlignIO.parse(file_handle, alignment_format)
Bio.AlignIO.read(file_handle, alignment_format)
```

where `file_handle` is the handler to the opened file, while the `alignment_format` is a lower case string with the alignment format (e.g. fasta, clustal, stockholm, mauve, phylip,...).

```
from Bio import AlignIO

alignments = AlignIO.read("file_samples/PF02171_seed.sth", "stockholm")
```

```
for align in alignments:
    start = align.annotations["start"]
    end = align.annotations["end"]
    seq = align.seq
    desc = align.description
    dbref = ",".join([x for x in align.dbxrefs])
    print("{} S:{} E:{}".format(desc, start, end))
    if (len(dbref) > 0):
        print(dbref)
    print("{}".format(seq))
    print("")
```

```
AG01 SCHPO/500-799 S:500 E:799
YLFFILDK-NSPEP-YGSIKRVCNTMLGVPSCAISKHILQS-----KPQYCANLGMKINVKVGGIN-CSLIPKSNP----L

AG06 ARATH/541-851 S:541 E:851
FILCILPERKTSDI-YGPWKKICLTEGIHTQCICPIKI-----SDQYLTNVLLKINSKLGGIN-SLLGIEYSYNIPLI

AG04 ARATH/577-885 S:577 E:885
FILCVLPDKKNSDL-YGPWKKNLTFEGIVTQCMAPTRQPN-----QYLTNLLKINAKLGGIN-SMLSVERTPAFTVI

TAG76_CAEEL/660-966 S:660 E:966
CIIIVLQS-KNSDI-YMTVKEQSDIVHGIMSQCVLMKNVSRP-----TPATCANIVLKLNMKMGGIN--SRIVADKITNKYL
```

# Writing and converting MSAs

Biopython provides a function `Bio.AlignIO.write()` to write alignments to file

and

`Bio.AlignIO.convert()` to convert one format into the other (provided that all information needed for the second format is available)

```
N = Bio.AlignIO.write(alignments, outfile, file_format)
```

where `alignments` are a `MultipleSeqAlignment` object with the alignments to write to the output file with name `outfile` that has format `file_format` (a low case string with the file format). `N` is the number of entries written to the file.

Ex.

```
my_alignments = [align1, align2, align3]
N = AlignIO.write(my_alignments, "file_samples/my_malign.phy", "phylip")
```

```
Bio.AlignIO.convert(input_file, input_file_format, output_file, output_file_format)
```

basically by passing the input file name and format and output file name and format.

Ex:

```
Bio.AlignIO.convert("PF05371_seed.sth", "stockholm", "PF05371_seed.aln", "clustal")
```

**Example:** Convert the seed alignment of the [Piwi \(PF02171\)](#) family stored in the pfam (stockholm) format [PF02171\\_seed.sth](#) into phylip format. Print some stats on the data.

N. of seq: 16  
 Len of seq: 395  
 1 multiple alignments converted to phylip

```
# STOCKHOLM 1.0
#=GS AG01_SCHPO/500-799 AC 074957.1
#=GS AG06_ARATH/541-851 AC 048771.2
#=GS AG04_ARATH/577-885 AC 092VDS.2
#=GS TAG76_CAEL/660-966 AC P34681.2
#=GS O16720_CAEL/566-867 AC 016720.2
#=GS 062275_CAEL/594-924 AC 062275.1
#=GS YQ53_CAEL/650-977 AC 009249.1
#=GS NRDE3_CAEL/673-1001 AC 021691.1
#=GS Q17567_CAEL/397-708 AC 017567.1
#=GS AUB_DROME/555-852 AC 076922.1
#=GS PIWI_DROME/538-829 AC 09VKM1.1
#=GS PIWI_HUMAN/555-847 AC 096394.1
#=GS PIWI_ARCFU/110-406 AC 028951.1
#=GS PIWI_ARCFU/110-406 DR PDB: 2W42 B; 110-406;
#=GS PIWI_ARCFU/110-406 DR PDB: 1YTU B; 110-406;
#=GS PIWI_ARCFU/110-406 DR PDB: 2BGG B; 110-406;
#=GS PIWI_ARCFU/110-406 DR PDB: 1W9H A; 110-406;
#=GS PIWI_ARCFU/110-406 DR PDB: 2BGG A; 110-406;
#=GS PIWI_ARCFU/110-406 DR PDB: 1YTU A; 110-406;
#=GS PIWI_ARCFU/110-406 DR PDB: 2W42 A; 110-406;
#=GS Y1321_METJA/426-699 AC 058717.1
#=GS 067434_AQUAE/419-694 AC 067434.1
#=GS 067434_AQUAE/419-694 DR PDB: 1YVU A; 419-694;
#=GS 067434_AQUAE/419-694 DR PDB: 2F8S A; 419-694;
#=GS 067434_AQUAE/419-694 DR PDB: 2F8T A; 419-694;
#=GS 067434_AQUAE/419-694 DR PDB: 2F8S B; 419-694;
#=GS 067434_AQUAE/419-694 DR PDB: 2NUB A; 419-694;
#=GS 067434_AQUAE/419-694 DR PDB: 2F8T B; 419-694;
#=GS AG010_ARATH/625-946 AC 09XGW1.1
AG01_SCHPO/500-799
YLFFILDK.NSPEP.YGSIKRVNNTMLGVPSQCAISKHILQS.....KPQYCANLGMKINVKVGGIN.CSLIPKSNP....LGNVPTL.....ILGGDVYHPGI
AG06_ARATH/541-851
FILCILPERKTSOI.YGPWKICLCTEEGIHTQCICPIKI.....SDQYLTNVLLKINSKLGGIN.SLLGIEVSYNIPLINKIPTL.....TLGMDVSHGPF
AG04_ARATH/577-885
FILCVLPDKKNSDL.YGPWKKNLTFEGIVTQCMAPTRQPN.....QYLTNLLKINAKLGLLN.SMLSVERTPAFTVISKVPTI.....ILGMDVSHGPF
TAG76_CAEL/660-966
CIIIVLQS.KNSDI.YMTVKEQSDIVHIGMSQCVLMKNVSRP.....TPATCANIVLKLNMKGGIN..SRVADKITNKYLVDPQTH.....VVGIDVTHTPT
O16720_CAEL/566-867
LIVVVLPG..KTP.I.YAEVKRVGDTVLGIATQCVQAKNAIRT.....IFNEPII.....FLGCDITHPA
062275_CAEL/594-924
TFVFIITD.DSITT.LHQRYMKIEKDTKMIQVQDMKLSKALSV..IN...AGKRLTENVINKTNVKGGSN..VVFVDQAKKL....DSHL.....IIGVGISAPP
```



```
from Bio import AlignIO

alignments = AlignIO.read("file_samples/PF02171_seed.sth", "stockholm")

out = AlignIO.convert("file_samples/PF02171_seed.sth",
                      "stockholm",
                      "file_samples/PF05371_seed.aln",
                      "clustal")

print("N. of seq: {}\nLen of seq: {}".format(
    len(alignments),
    len(alignments[0])))

print("{} multiple alignments converted to phylip".format(out))
```

CLUSTAL X (1.81) multiple sequence alignment

```
AG01_SCHPO/500-799
AG06_ARATH/541-851
AG04_ARATH/577-885
TAG76_CAEL/660-966
O16720_CAEL/566-867
062275_CAEL/594-924
YQ53_CAEL/650-977
NRDE3_CAEL/673-1001
Q17567_CAEL/397-708
AUB_DROME/555-852
PIWI_DROME/538-829
PIWI_HUMAN/555-847
PIWI_ARCFU/110-406
Y1321_METJA/426-699
067434_AQUAE/419-694
AG010_ARATH/625-946
```

```
YLFFILDK-NSPEP-YGSIKRVNNTMLGVPSQCAISKHILQS-----
FILCILPERKTSOI-YGPWKICLCTEEGIHTQCICPIKI-----
FILCVLPDKKNSDL-YGPWKKNLTFEGIVTQCMAPTRQPN-----
CIIIVLQS-KNSDI-YMTVKEQSDIVHIGMSQCVLMKNVSRP-----
LIVVVLPG--KTP.I-YAEVKRVGDTVLGIATQCVQAKNAIRT-----
TFVFIITD-DSITT-LHQRYMKIEKDTKMIQVQDMKLSKALSV--IN---A
DILVGIAR-EKKPD-VHDILKYFEESIGLQTIQLCQQTVDKMMGG---Q
TIVFGIIA-EKRPD-MHDILKYFEELKQQTIQISSSETADKFMRD---H
MLVVMLAD-DNKTR-YDSLKKYLCVCEPIPNQCVNLRRLAGKSKDGENK
IVMVVMRS-PNEEK-YSCIKKRTCVDRPVPSQVVTLLKVIAPRQKQP---T
LILCLVPN-DNAER-YSSIKKRGYVDRAPVTQVVTLLKTTKNRSL-----
IVVCLLSS-NRKDK-YDAIKKYLCTDCPTSPQCVVARTLGKQQT-----
GIMLVLPE-YNTPL-YYKLKSYLINS--IPSQFMRYDILSNRNL-----
CFALITGKEYKNDYYEILKKQLFDLKIISQNILWENWRKDDK-----
LVIVFLEEYPKVDP-YKSFLLYDFVKRELLKKMIPSVILNRTLKN---E
LLLAILPD-NGSL-YGDLKRICETELGLISQCLTKHVFKI-----
```

AG01\_SCHPO/500-799

-KPQYCANLGMKINVKVGGIN-CSLIPKSNP----LGNVPTL-----

# Manipulating/writing MSA

It is possible to slice alignments using the `[]` operator applied on a `SeqRecord`.

Think about it as a matrix

1. `SeqRecord[i, j]` returns the *j*th character of alignment *i* as a string;
2. `SeqRecord[:, j]` returns all the *j*th characters of the multiple alignment as a string;
3. `SeqRecord[:, i:j]` returns a `MultipleSeqAlignment` with the sub-alignments going for *i* to *j* (excluded)
4. `SeqRecord[a:b, i:j]` similar to 3. but for alignments going from *a* to *b* (excluded) only

```
YLFFILDK-NSPEP-YGSIKLVPPVYYAHLVSNLARYQDV
FILCILPERKTSDI-YGPWKIVAPVRYAHLAAAQVAQFTK
FILCVLPDKKNSDL-YGPWKVVAPICYAHLAAAQLGTFMK
CIIIVLQS-KNSDI-YMTVKIPTPVYYADLVATRARCHVK
LIVVVLPG--KTPI-YAEVKIPAPAYYAHLVAFRARYHLV
TFVFIITD-DSITT-LHQRYLPTPLYVANEYAKRGRNLWN
DILVGIAR-EKKPD-VHDILVPDVLAAENLAKRGRNNYK
TIVFGIIA-EKRPD-MHDILIPNVSYAAQNLAQRHNNYK
MLVVMLAD-DNKTR-YDSLKVPAPCQYAHKLAFLTAQSLH
IVMVVMSR-PNEEK-YSCIKVPAVCHYAHKLAFLVAESIN
LILCLVPN-DNAER-YSSIKVPAVCQYAKKLATLVGTNLH
IVVCLLSS-NRKDK-YDAIKVPAVCQYAHKLAFLVGQSIH
GIMLVLPE-YNTPL-YYKLKLPVTVNYPKLVAGIIANVNR
CFALIIGKEKYKDNDDYYEILIPAPIHYADKFVKALGKNWK
LVIVFLEEYPKVDP-YKSFLLPATVHYSKITKMLRGIE
LLLAILPD-NNGSL-YGDLKIVPPAYYAHLAAFRARFYLE
```

`align[0,0]` is Y  
`align[2,1]` is I  
`align[:,0]` is YFFCLTDTMILIGCLL

`align[:,0:3]` gets first 3 rows (SeqRecords)  
YLFFILDK-N...  
FILCILPERK...  
FILCVLPDK...

`align[0:3,0:3]` first 3 cols of first 3 rows (SeqRecords):  
YLF  
FIL  
FIL



# Pairwise alignment

Biopython has its own module to make pairwise alignment. It provides two algorithms: [Smith-Waterman](#) for **local alignment** and [Needleman-Wunsch](#) for **global alignment**. These methods are implemented in two Biopython functions of the `Bio.pairwise2` module:

```
pairwise2.align.globalxx()  
pairwise2.align.localxx()
```

Example:

```
alignments = pairwise2.align.globalxx("ACCGTTATATAGGCCA", "ACGTACTAGTATAGGCCA")  
for i in range(len(alignments)):  
    print(alignments[i])
```

```
('ACCGT--TA-TATAGGCCA', 'A-CGTACTAGTATAGGCCA', 15.0, 0, 19)  
( 'ACCGT--TA-TATAGGCCA', 'AC-GTACTAGTATAGGCCA', 15.0, 0, 19)
```

```
aligns = pairwise2.align.globalxx(seq1, seq2)  
aligns = pairwise2.align.localxx(seq1, seq2)
```

where `seq1` and `seq2` are two `str` objects. These methods return a list of alignments (at least one) that have the same **optimal score**. Each alignment is represented as tuples with the following 5 elements in order:

1. The alignment of the first sequence;
2. The alignment of the second sequence;
3. The alignment score;
4. The start of the alignment (for global alignments this is always 0);
5. The end of the alignment (for global alignments this is always the length of the alignment).

# Pairwise alignment

## OPTIONS FOR MATCHES/MISMATCHES AND GAP OPENS/EXTENSIONS

`pairwise2.align.globalxx`  
`pairwise2.align.globalmx`  
`pairwise2.align.globalms`  
`pairwise2.align.globalmd`  
`pairwise2.align.globalxd`  
`pairwise2.align.globalxs`  
`pairwise2.align.localxx`  
`pairwise2.align.localmx`  
`pairwise2.align.localms`  
`pairwise2.align.localmd`  
`pairwise2.align.localxd`  
`pairwise2.align.localxs`

### Match parameters can be:

- `x` : means that a match scores 1 a mismatch 0;
- `m` : the match and mismatch score are passed as additional params after the sequence (es. `aligns = pairwise2.align.globalmx(seq1,seq2, 1, -1)` to set 1 as match score and -1 as mismatch penalty).

### Gap parameters can be:

- `x` : gap penalty is 0;
- `s` : same gap open and gap extend penalties for the 2 sequences (passed as additional params after seqs).
- `d` : different gap open and gap extend penalties for the 2 seqs (additional params after the seqs).

The first letter is **the score for a match**  
the second letter is **the penalty for a gap**

# Pairwise alignment

**Example.** Let's perform the alignment of the two

sequences “ACCGTTATATAGGCCA” and

“ACGTACTAGTATAGGCCA”

```
('ACCGT--TA-TATAGGCCA', 'A-CGTACTAGTATAGGCCA', 15.0, 0, 19)
('ACCGT--TA-TATAGGCCA', 'AC-GTACTAGTATAGGCCA', 15.0, 0, 19)
```

## Looping through aligns

ACCGT--TA-TATAGGCCA  
A-CGTACTAGTATAGGCCA  
Score: 15.0, Start: 0, End: 19

ACCGT--TA-TATAGGCCA  
AC-GTACTAGTATAGGCCA  
Score: 15.0, Start: 0, End: 19


Match: 1, Mismatch: -1, Gap open: -0.5, Gap extend: -0.2

ACCGT--TA-TATAGGCCA  
A-CGTACTAGTATAGGCCA  
Score: 13.3, Start: 0, End: 19

ACCGT--TA-TATAGGCCA  
AC-GTACTAGTATAGGCCA  
Score: 13.3, Start: 0, End: 19

[illegible]

# http://biopython.org



Python Tools for  
Computational  
Molecular Biology

[Documentation](#)  
[Download](#)  
[Mailing lists](#)  
[News](#)  
[Biopython Contributors](#)  
[Scriptcentral](#)  
[Source Code](#)  
[GitHub project](#)

Biopython version 1.78  
© 2020. All rights  
reserved.

[Edit this page on GitHub](#)

## Biopython

See also our [News feed](#) and [Twitter](#).

### Introduction

Biopython is a set of freely available tools for biological computation written in [Python](#) by an international team of developers.

It is a distributed collaborative effort to develop Python libraries and applications which address the needs of current and future work in bioinformatics. The source code is made available under the [Biopython License](#), which is extremely liberal and compatible with almost every license in the world.

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

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

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

The latest release is [Biopython 1.78](#), released on 4 September 2020.



<https://biopython.org/docs/1.78/api/py-modindex.html>

Check:

Seq

SeqRecord

MultipleSeqAlignment

The image shows a side-by-side comparison of the Biopython website and its Python Module Index. On the left is the Biopython homepage, featuring a logo with a blue and yellow DNA double helix and the text 'biopython'. Below the logo is a search bar labeled 'Search docs'. Underneath the search bar, the 'API CONTENTS:' section lists 'Bio package' and 'BioSQL package'. At the bottom of the page, it says 'Biopython v: 1.78'. On the right is the 'Python Module Index' page, which has a header 'Python Module Index' and a sub-header 'b'. Below this, a list of modules is shown, including 'Bio.Affy', 'Bio.Affy.CellFile', 'Bio.Align', 'Bio.Align.AlignInfo', 'Bio.Align.Applications', 'Bio.Align.substitution\_matrices', 'Bio.AlignIO', 'Bio.AlignIO.ClustalIO', 'Bio.AlignIO.EmbossIO', 'Bio.AlignIO.FastaIO', 'Bio.AlignIO.Interfaces', 'Bio.AlignIO.MafIO', 'Bio.AlignIO.MauveIO', 'Bio.AlignIO.MsfIO', 'Bio.AlignIO.NexusIO', 'Bio.AlignIO.PhylipIO', 'Bio.AlignIO.StockholmIO', 'Bio.Alphabet', 'Bio.Application', 'Bio.bgzf', 'Bio.Blast', 'Bio.Blast.Applications', 'Bio.Blast.NCBIWWW', 'Bio.Blast.NCBIXML', 'Bio.Blast.ParseBlastTable', 'Bio.Blast.Record', 'Bio.CAPS', 'Bio.Cluster', 'Bio.codonalign', 'Bio.codonalign.chisq', 'Bio.codonalign.codonalignment', and 'Bio.codonalign.codonseq'.

biopython

Search docs

API CONTENTS:

Bio package

BioSQL package

Biopython v: 1.78

Python Module Index

b

Bio

- Bio.Affy
- Bio.Affy.CellFile
- Bio.Align
- Bio.Align.AlignInfo
- Bio.Align.Applications
- Bio.Align.substitution\_matrices
- Bio.AlignIO
- Bio.AlignIO.ClustalIO
- Bio.AlignIO.EmbossIO
- Bio.AlignIO.FastaIO
- Bio.AlignIO.Interfaces
- Bio.AlignIO.MafIO
- Bio.AlignIO.MauveIO
- Bio.AlignIO.MsfIO
- Bio.AlignIO.NexusIO
- Bio.AlignIO.PhylipIO
- Bio.AlignIO.StockholmIO
- Bio.Alphabet
- Bio.Application
- Bio.bgzf
- Bio.Blast
- Bio.Blast.Applications
- Bio.Blast.NCBIWWW
- Bio.Blast.NCBIXML
- Bio.Blast.ParseBlastTable
- Bio.Blast.Record
- Bio.CAPS
- Bio.Cluster
- Bio.codonalign
- Bio.codonalign.chisq
- Bio.codonalign.codonalignment
- Bio.codonalign.codonseq

# Installing biopython

```
import Bio
```

```
-----  
ImportError                                Traceback (most recent call last)  
<ipython-input-1-f227b1b7f7f3> in <module>()  
----> 1 import Bio  
  
ImportError: No module named 'Bio'
```

In windows installing Biopython should be as easy as opening the command prompt as administrator (typing `cmd` and then right clicking on the link choosing run as administrator) and then `pip3 install biopython`.

In linux `sudo pip3 install biopython` will install biopython for python3 up to python3.5. On python 3.6, the command is: `python3.6 -m pip install biopython`.

## Exercises

1. Write a python function that reads a genebank file given in input and prints off the following information:
  1. Identifier, name and description;
  2. The first 100 characters of the sequence;
  3. Number of external references (dbxrefs) and ids of the external refs.
  4. The name of the organism (hint: check the annotations dictionary at the key "organism")
  5. Retrieve and print all (if any) associated publications (hint: annotation dictionary, key: "references")
  6. Retrieve and print all the locations of "CDS" features of the sequence (hint: check the features )

Hint: go back and check the details of the `SeqRecord` object.

Test the program downloading some files from genebank like [this](#)

Show/Hide Solution

2. Write a python program that loads a pfam file (stockholm format .sth) and reports for each record of the alignment:
  1. the id of the entry
  2. the start and end points
  3. the number of gaps and the % of gaps on the total length of the alignment
  4. the number of external database references (dbxrefs), and the first 3 external references comma separated (hint: use join).

Print these information to the screen. Finally, write this information in a tab separated file (.tsv) having the following format: `#ID\tstart\tend\tnum_gaps\tpercentage_gaps\tbdbxrefs`.