

**Universidade Federal do Pará**  
**Instituto de Tecnologia**  
**Laboratório de Planejamento de Redes de**  
**Alto Desempenho**

**Introdução a Linguagem Python**  
- Aplicações em Bioinformática -

Diego Damasceno

[damasceno.diego@gmail.com](mailto:damasceno.diego@gmail.com)

<http://damascenodiego.wordpress.com>



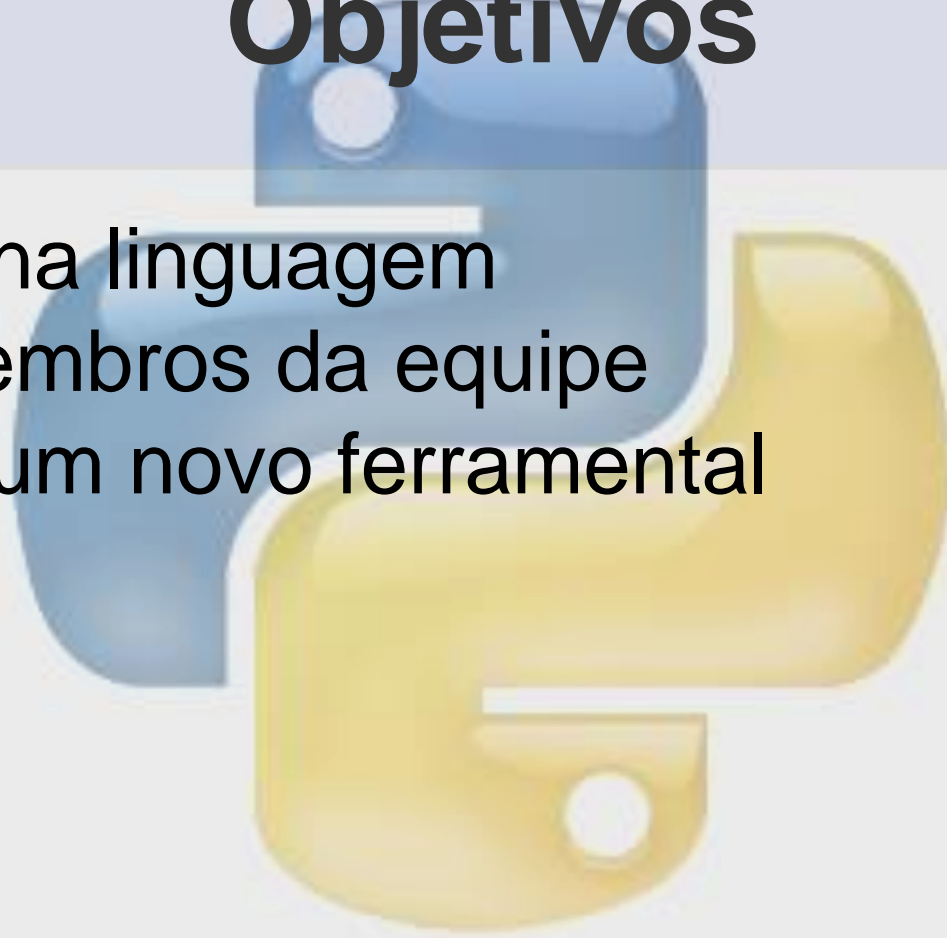
# Tópicos Abordados

- Objetivos
- Histórico
- Caraterísticas Gerais
- Aplicações
- Sintaxe
- Tipos de Dados
- Estruturas de Dados
- Estruturas de Controle
- Expressões Regulares
- BioPython
- Referências



# Objetivos

- Overview na linguagem
- Nivelar membros da equipe
- Introduzir um novo ferramental



# Histórico

- Criada em 1989
- Guido van Rossum



# Histórico

Há mais de seis anos, em dezembro de 1989, eu estava procurando por um projeto de programação como "hobby" que me mantivesse ocupado durante a semana próxima ao Natal. Meu escritório... estaria fechado, mas eu tinha um computador em casa, e não muito mais do que isso em mãos. Eu decidi escrever um interpretador para a **nova linguagem de scripting** sobre a qual eu vinha pensando ultimamente: uma descendente da ABC que agradaria a **hackers de Unix/C**. Eu escolhi **Python** como um título provisório para o projeto, sendo que eu estava num humor um pouco irreverente (e sendo também um grande fã do Monty Python's Flying Circus).

— Introdução de Programming Python, por Mark Lutz, O'Reilly

# Características Gerais

The Python logo, consisting of two interlocking snakes, one blue and one yellow, is positioned in the background of the slide.

- Multi-Paradigma
  - Imperativa, OO e Funcional
- Interpretada
- Case Sensitive
- Tipagem Dinâmica
- Propósito Geral
  - Web, Scripting, GUI
- Multiplataforma
  - Windows, Mac, Linux

# Características Gerais

- Legibilidade
  - Fácil leitura do código
  - “Human Readable”



# Características Gerais

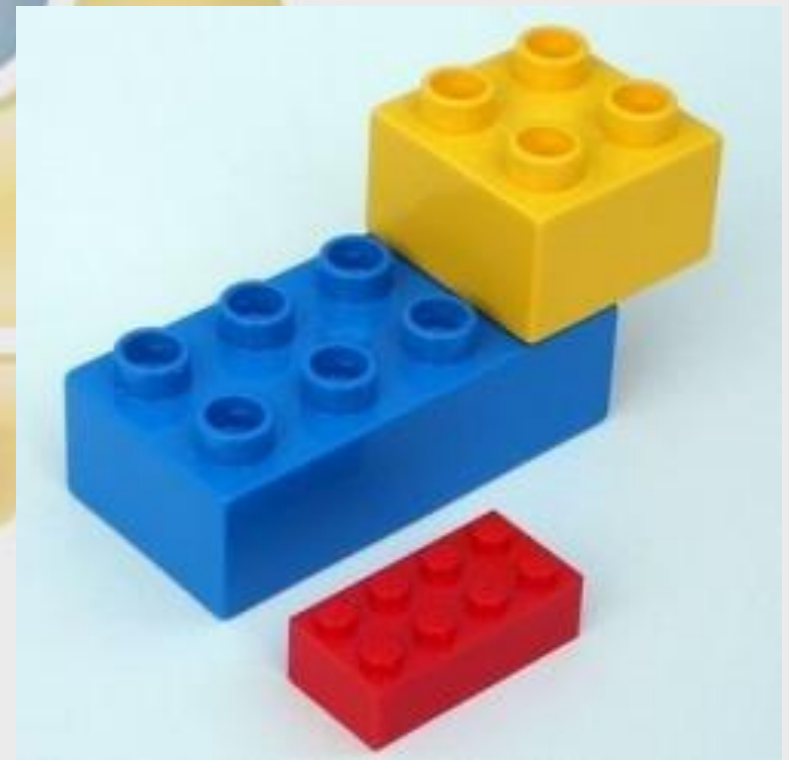
- Funcionalidades Embutidas
  - Leitura/Escrita de arquivos
  - Leitura/Escrita XML
  - Leitura/Escrita zip
  - Acesso a URLs





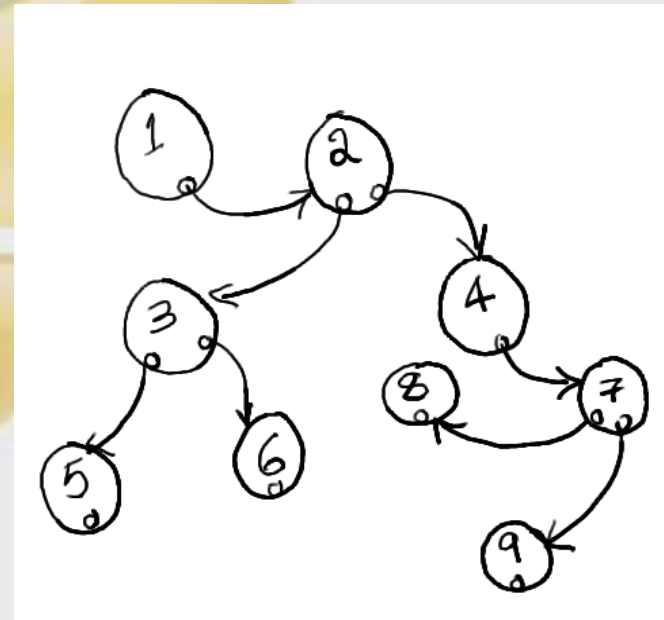
# Características Gerais

- Diversos módulos
  - Bioinformática
  - Gerador de PDF
  - Acesso a bancos de dados
  - Animações
  - Gráficos 2D/3D



# Características Gerais

- Estruturas de dados em alto nível
  - Dictionary
  - Set
  - Lists
  - Tuples



# Características Gerais

- Multiparadigma
  - Procedural
  - Orientado a Objeto



# Características Gerais

- Extensível
- Bindings
- Third Party Modules
- Bibliotecas



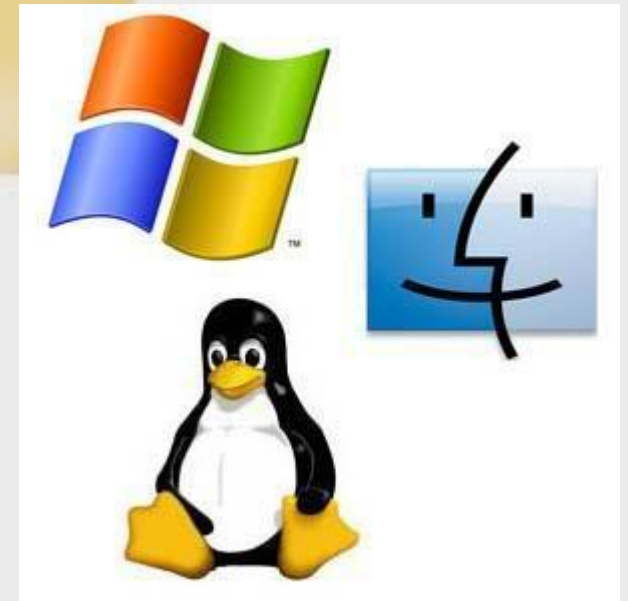
# Características Gerais

- Open Source
  - Pode ser usada livremente
  - Distribuída livremente



# Características Gerais

- Multiplataforma
  - Interpretador
  - Windows, Linux, Mac

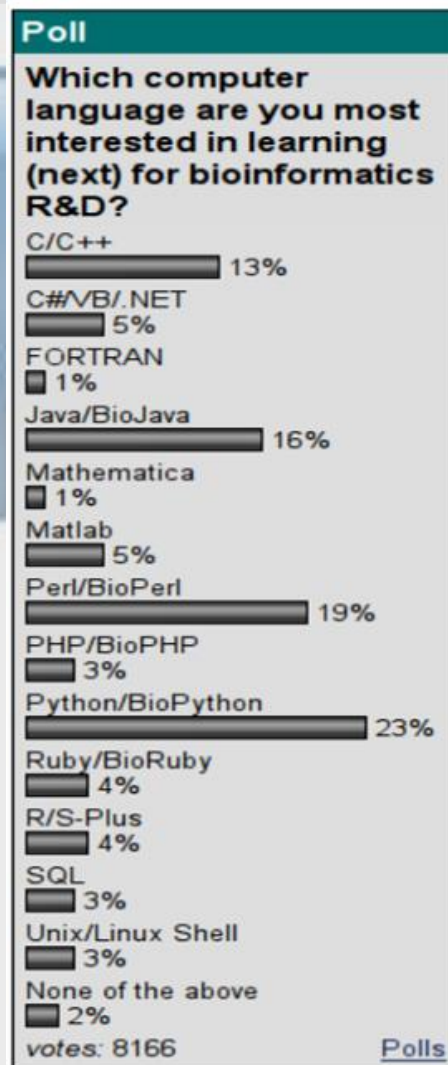


# Características Gerais

- Comunidade Próspera
- Comunidade científica
- Diversas bibliotecas
- Suporte



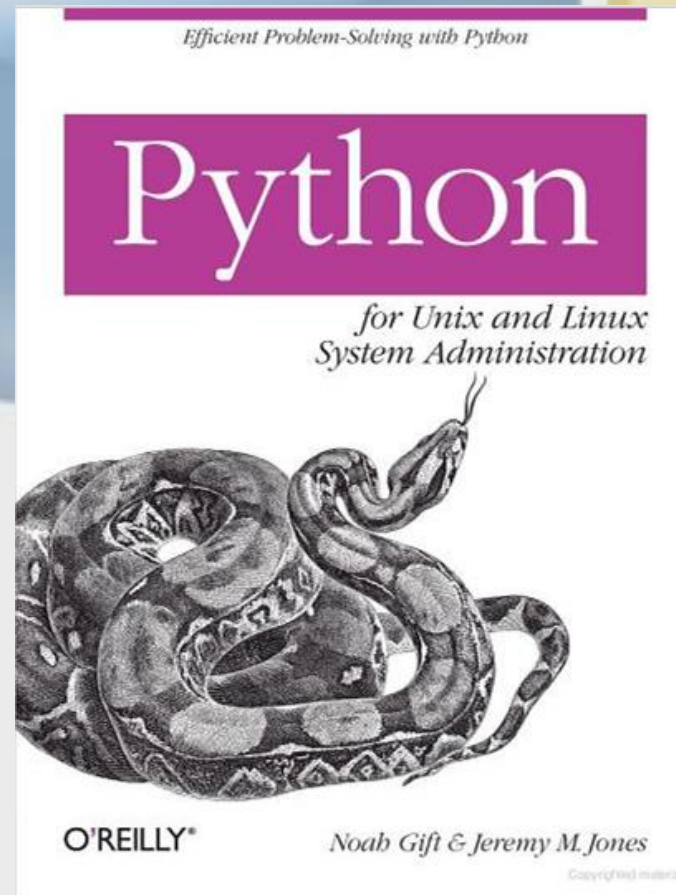
# Características Gerais





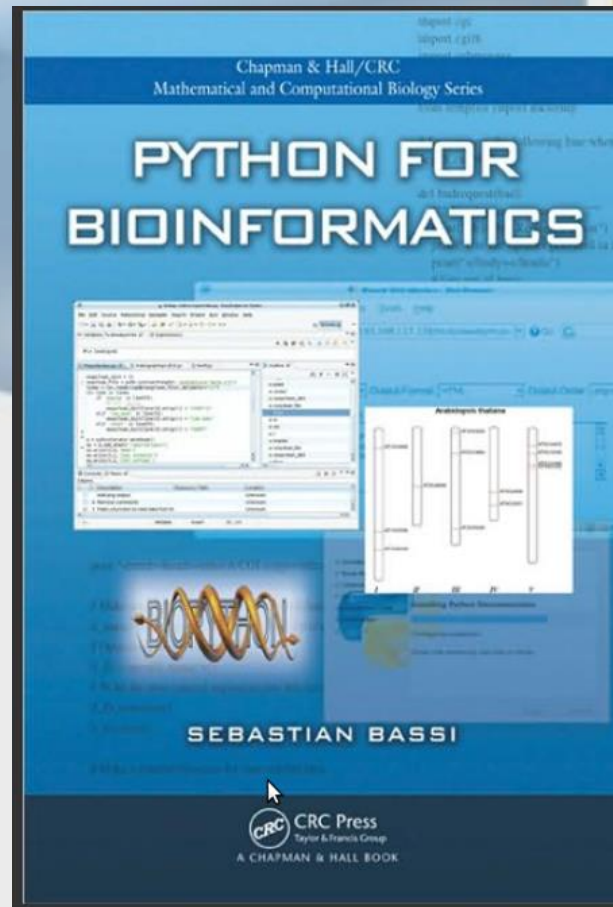
# Aplicações

## •Scripting



# Aplicações

- Bioinformática



# Aplicações

- Bioinformática



Biopython Tutorial and Cookbook

Jeff Chang, Brad Chapman, Iddo Friedberg, Thomas Hamelryck,  
Michiel de Hoon, Peter Cock, Tiago Antao, Eric Talevich, Bartek Wilczyński

Last Update – 18 August 2011 (Biopython 1.58)

# Aplicações

## • Aplicações Web - Django



### Tasks filed under "gtd"

This list belongs to J-School

#### Add a task:

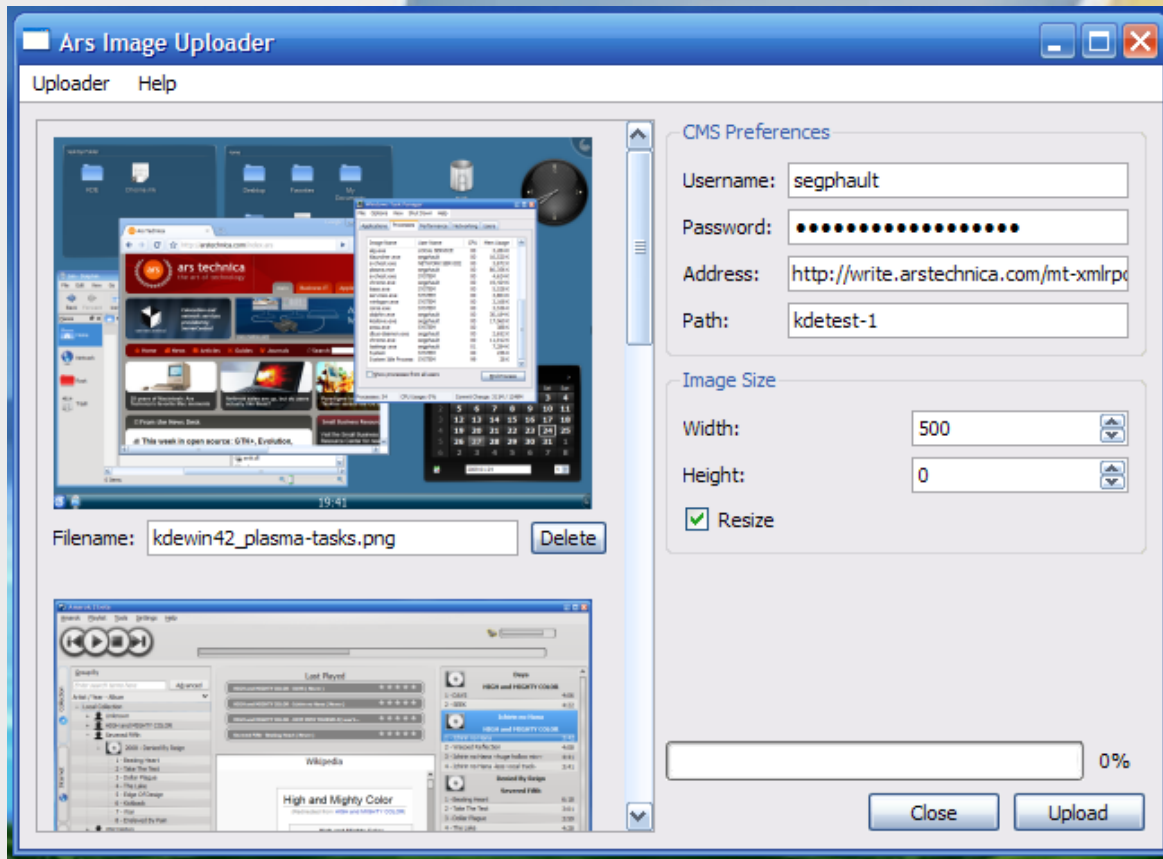
Task:  Due date:  Priority:  Assign to:

#### Incomplete tasks

Done	Priority	Task								Created	Due on
<input type="checkbox"/>	1	Priority cha	7	8	9	10	11	12	13	07/24/2008	
<input type="checkbox"/>	1	Date comp	14	15	16	17	18	19	20	09/12/2008	09/30/2008
<input type="checkbox"/>	3	Ajax updat	21	22	23	24	25	26	27	08/13/2008	
<input type="checkbox"/>	3	Allow long	28	29	30					09/12/2008	09/30/2008
<input type="checkbox"/>	3	Code review - clean?								09/12/2008	09/19/2008

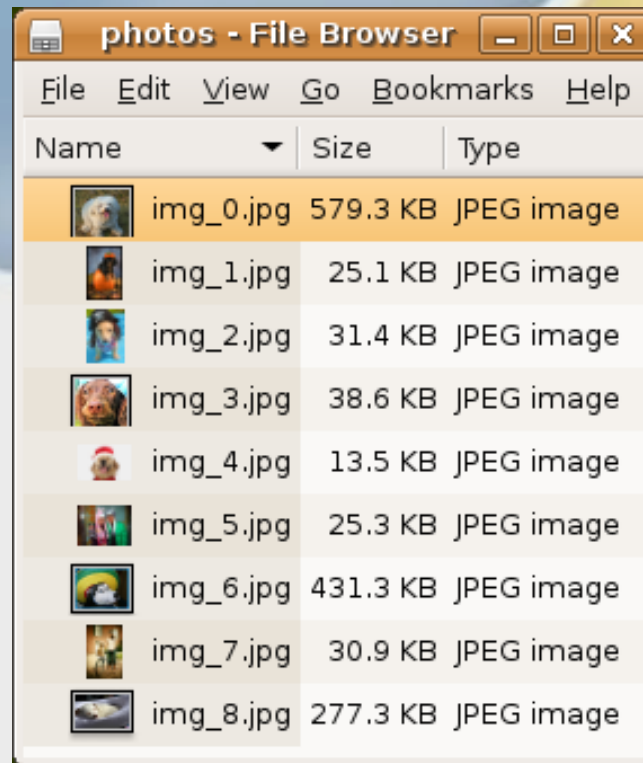
# Aplicações

## .GUI - PyQt



# Aplicações

- GUI - PyGTK



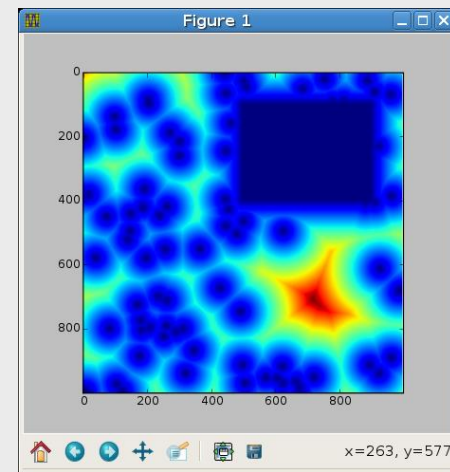
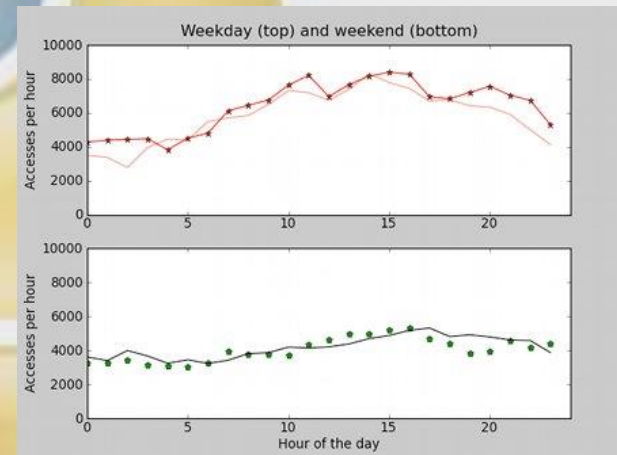
# Aplicações

- Jogos - PyGame



# Aplicações

## • Computação Científica – SciPy, NumPy

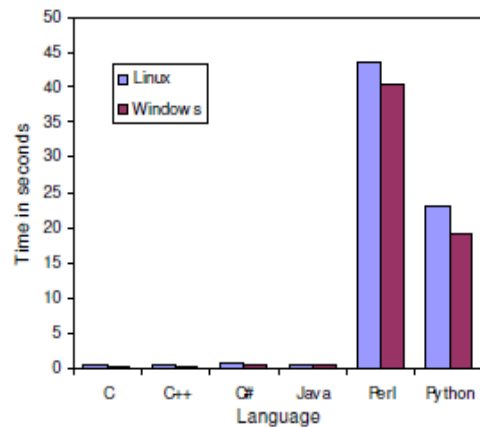




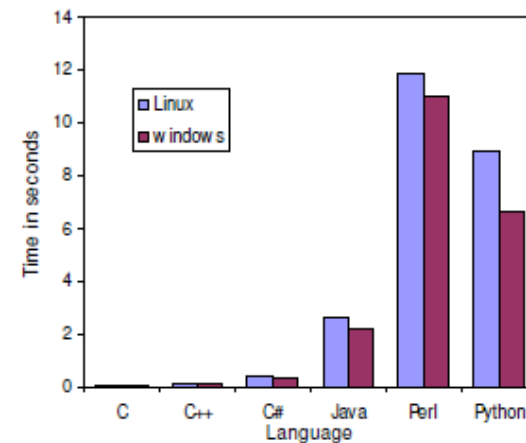
# Relevância para Bioinformática

- **Freely** available
- **Open source** tools
- Available for **all the major operating systems**
- Very **high-level** programming language
- **Easy to learn** syntax
- **Object Oriented** programming capabilities
- Wide array of **libraries**
- Can interface to **optimized code** written in C, C++ or even FORTRAN
- **Numerical Python** project numpy (Oliphant, 2006)
  - Scientific programming (Oliphant, 2007)
  - Molecular dynamics (Hinsen, 2000).
- **High-quality plotting** libraries such as matplotlib
  - (matplotlib.sourceforge.net)

# Relevância para Bioinformática

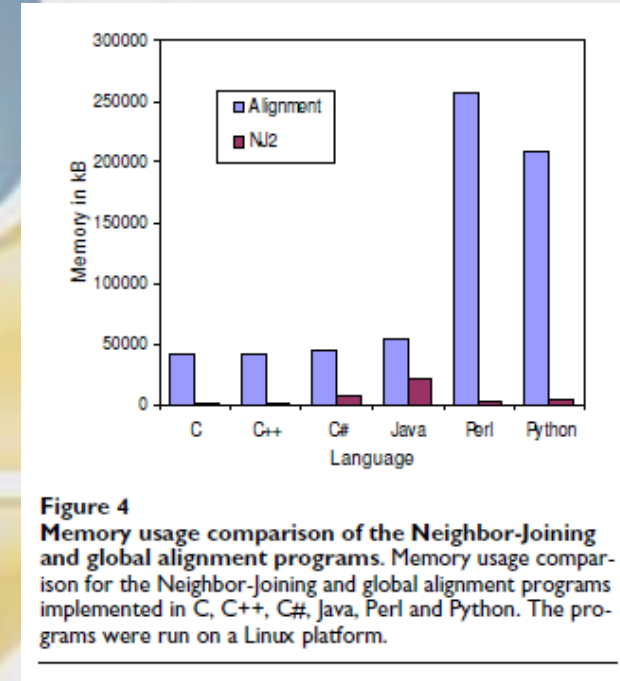
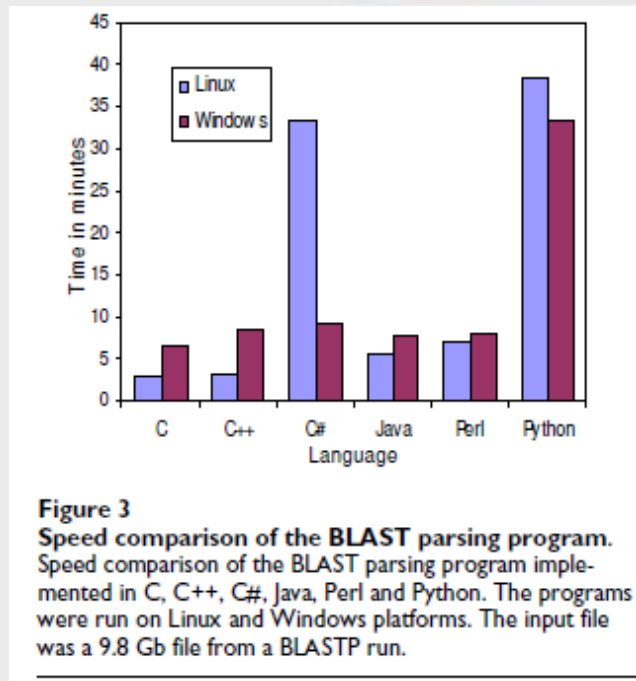


**Figure 1**  
Speed comparison of the global alignment program. Speed comparison of the global alignment algorithm using a gap penalty of 10 implemented in C, C++, C#, Java, Perl and Python. The programs were run on Linux and Windows platforms. Two DNA sequences of 3216 bp and 3217 bp were used.

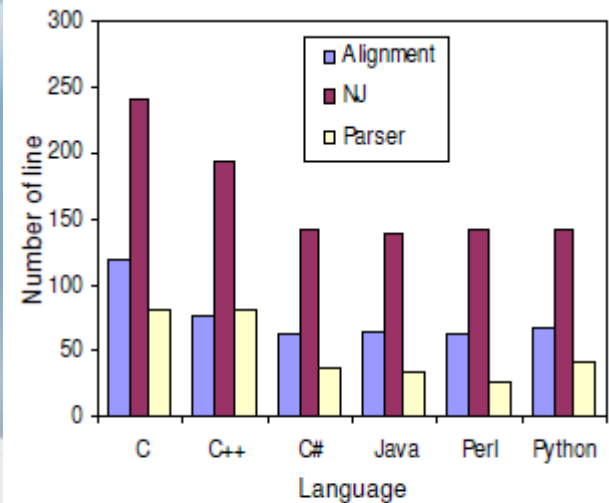


**Figure 2**  
Speed comparison of the Neighbor-Joining program. Speed comparison of the Neighbor-Joining algorithm using the Jukes-Cantor evolutionary model implemented in C, C++, C#, Java, Perl and Python. The programs were run on Linux and Windows platforms. The input file was an alignment of 76 DNA sequences.

# Relevância para Bioinformática



# Relevância para Bioinformática



**Figure 5**  
Number of lines for each program. Number of lines for the global alignment, BLAST parser and Neighbor-Joining programs implemented in C, C++, C#, Java, Perl and Python.

# Hands On Python !



# Abrir o Ambiente



# PyDev - Eclipse



<http://pydev.org/>

# PyDev - Eclipse

The screenshot shows the Eclipse IDE with the PyDev plugin. The main editor window displays the following Python code:

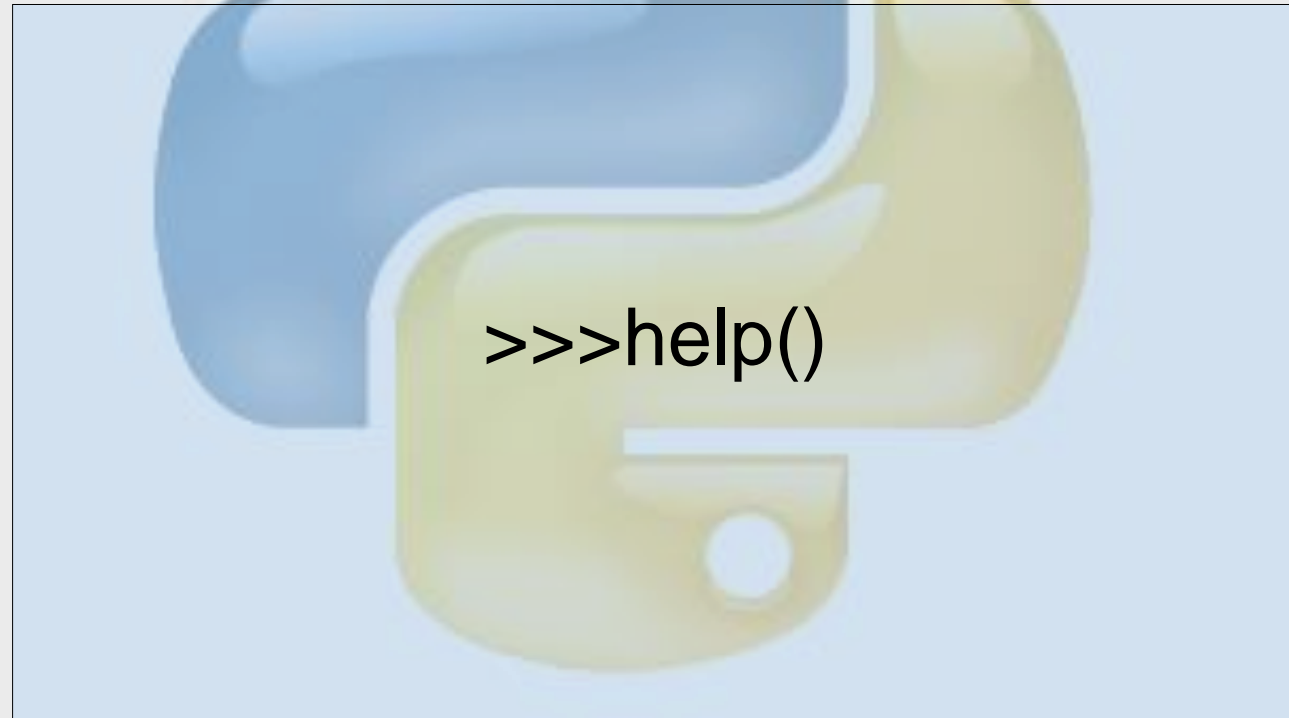
```
28 #check when cache is active
29 self.assertEqual(set([1, _E, _D, _C, _B, _A, object]),
30 self.assertEqual(set([_E, _D, _C, _B, _A, object]), Ge
31
32 self.assertEqual(set([_D, _C, _B, _A, object]), GetClas
33 self.assertEqual(set([_C, _B, object]), GetClassHierar
34
35 #check when cache is active
36 self.assertEqual(set([_A, object]), GetClassHierarchy(
37 self.assertEqual(set([_A, object]), GetClassHierarchy(
38
39
40 def testIsInstance(self):
41 '''
42 Check if IsInstance works with class name.
43 '''
44 self.assert_(IsInstance(_C(), '_B'))
45 ERROR_NOT_DEFINED_VARIABLE
46 self.assert_(IsInstance(_C(), ('_B',)))
47 self.assert_(not IsInstance(_C(), ('_A',)))
48 self.assert_(IsInstance(_C(), ('_A', '_B')))
49 self.assert_(not IsInstance(_C(), ('_A', '_D')))
50
51
52 def testIsSubclass(self):
```

The Problems window at the bottom shows the following error message:

```
<terminated> C:\temp\coilib50\source\python\coilib50\basic\klass_tests\test_class.py
-----
Traceback (most recent call last):
  File "C:\temp\coilib50\source\python\coilib50\basic\klass_tests\test class.py", line 2
    self.assertEqual(set([1, _E, _D, _C, _B, _A, object]), GetClassHierarchy(_E))
AssertionError: set([1, <class '__main__.E'>, <class '__main__.B'>, <class '__main__.D
```



# Duvidas ?



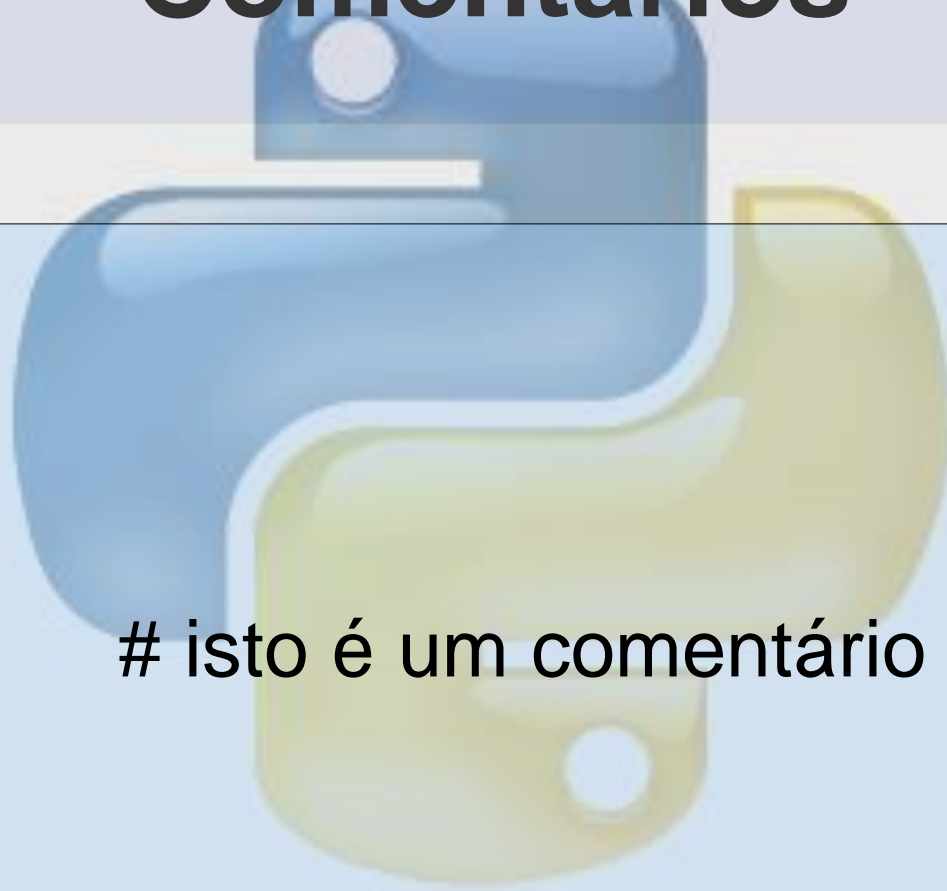
# Hello World!



```
>>>print("\nHello ACTG!")
```

\* python 2.x  
\*\* python 3.x

# Comentários



`# isto é um comentário`

# hashbang (#!)



```
#!/usr/bin/env python
```

```
#!/usr/bin/python
```

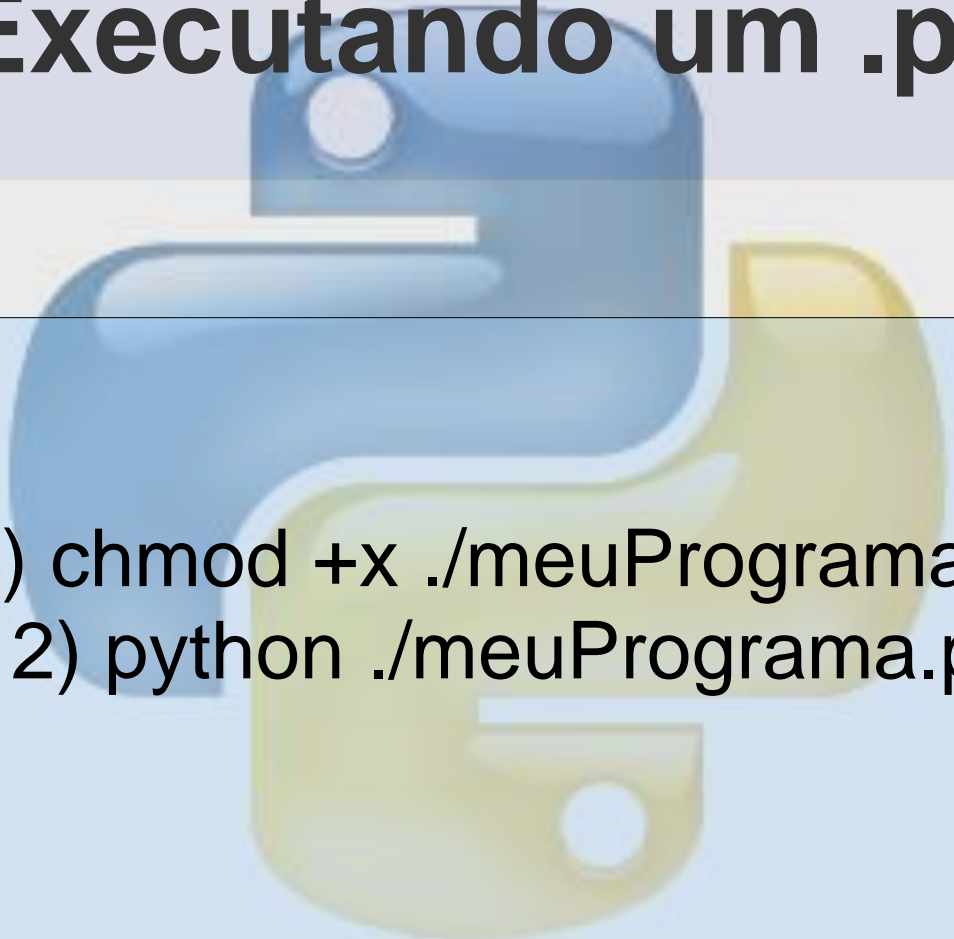
# Encoding

The Python logo, consisting of two interlocking snakes, one blue and one yellow, is centered in the background of the slide. It is partially obscured by a light blue rectangular box that contains text.

```
# -*- coding: ENCODING -*-
```

ascii, latin1, 8859-1, UTF-8 ...

# Executando um .py

- 
- The Python logo, consisting of two interlocking snakes, one blue and one yellow, is positioned in the background of the slide. It is partially obscured by a light blue rectangular box that contains the terminal commands.
- 1) `chmod +x ./meuPrograma.py`
  - 2) `python ./meuPrograma.py`

# Entrada de Datos Via Teclado

```
>>> name = raw_input("Enter your name: ")  
          (*)
```

```
Enter your name: Seba
```

```
>>> name  
'Seba'
```

```
>>> name = input("Enter your name: ")  
          (**)
```

```
Enter your name: Seba
```

```
>>> name  
'Seba'
```

\* python 2.x

\*\* python 3.x

# Tipagem Dinâmica

```
>>> X= 50
>>> type(X)
>>> X="meu nome"
>>> type(X)
>>> X= 1.2345
>>> type(X)
```



# Operações Básicas

```
>>>3+4 # Adição  
>>>10-9 # Subtração  
>>>23*10 # Multiplicação  
>>>100/2 # Divisão  
>>>3%2 # Resto
```

# Strings

## Criação

```
>>>"This is a string in Python"  
>>>'This is a string in Python'  
>>>"""This is a string in Python"""  
>>>""""This is a string in Python""""
```

python 2.x - ASCII  
python 3.x - UNICODE

# Strings

## Atribuição e Alteração da Caixa

```
>>> signal_peptide="MASKATLLLAFTLLFATCIA"  
>>> signal_peptide.lower()  
'maskatlllafatcia'  
>>> signal_peptide  
'MASKATLLLAFTLLFATCIA'  
>>> signal_peptide=signal_peptide.lower()  
>>> signal_peptide  
'maskatlllafatcia'
```

# Strings

## Substituição

```
>>> DNAseq="TTGCTAG"  
>>> mRNAseq=DNAseq.replace("T","U")  
>>> mRNAseq  
'UUGCUGAG'
```

# Strings

## Contagem e Localização de Caracteres

```
>>> c=DNAseq.count("C")
>>> g=DNAseq.count("G")
>>> float(c+g)/len(DNAseq)*100
48.387096774193552
>>> mRNAseq.find("UAG")
4
>>> mRNAseq.find("xxx")
-1
>>> mRNAseq.index("xxx")
ValueError
```

# Strings

## Divisão de Strings em Pontos Específicos

```
>>> "This string has words separated by spaces".split()
['This', 'string', 'has', 'words', 'separated', 'by', 'spaces']
>>> "Alex Doe,5555-2333,nobody@example.com".split(",")
['Alex Doe', '5555-2333', 'nobody@example.com']
```

# Strings

## Concatenação e Medição

```
>>> aseq="atggctaggc"  
>>> list(aseq)  
['a', 't', 'g', 'g', 'c', 't', 'a', 'g', 'g', 'c']  
>>> "".join(['a', 't', 'g', 'g', 'c', 't', 'a', 'g', 'g', 'c'])  
'atggctaggc'  
>>> DNAseq="ATGCTAGACGTCCTCAGATAGCCG"  
>>> TATAbox="TATAAA"  
>>> TATAbox+DNASeq  
'TATAAAATGCTAGACGTCCTCAGATAGCCG'  
>>> my_sequence="MRVLLVALALLALAASATS"  
>>> len(my_sequence)  
19
```

# Strings

## Substrings

```
>>> nome="ACTCGTCACGTAC"  
>>> print(nome[:10])  
ACTCGTCACG  
>>> print(nome[-3])  
T  
>>> print(nome[5:10])  
TCACG
```



# Strings

## Iteração

```
>>> nome= "mini curso do LPRAD"  
>>> for i in nome:  
>>>     print(i)
```

# Strings

Existencia de Elemento

```
letras="ACGTGCACGAT"
```

```
"u" in letras
```

```
"A" in letras
```

# Listas

## Criação, Inserção, Remoção, Inversão, Ordenação e Intervalos Numéricos

```
Lista=[]
Lista=list()
Lista=["yo"]
Lista.append("hehe")
Lista.pop()
Lista.reverse()
Lista.sort()
range(100) # (**)
 xrange(10000) # (*)
Lista1.append("item")
Lista2.append("coisa")
Lista2.extend(Lista1)
Lista1[1]="treco"
Lista1.insert(2, "outraCoisa")
Lista1+=["maisUmaCoisa"]
Lista1+="maisUmaCoisa"
del Lista2[1]
Lista2.remove("coisa")
Lista2.index("coisa")
```

\* python 2.x

\*\* python 3.x

# Tuplas



```
>>> point=tuple()  
>>> point=(23,56,11)  
>>> point[2]
```

\* imutável

# Dicionários

```
>>> IUPAC = {'A':'Ala','C':'Cys','E':'Glu'}
>>> print("C stands for the amino acid "+IUPAC['C'])
C stands for the amino acid Cys
>>> IUPAC['E']
'Glu'
>>> IUPAC.keys()
['A', 'C', 'E']
>>> IUPAC.values()
['Ala', 'Cys', 'Glu']
>>> IUPAC.items()
[('A', 'Ala'), ('C', 'Cys'), ('E', 'Glu')]
>>> IUPAC.get('A','No translation')
'Ala'
>>> IUPAC.get('Z','No translation')
'No translation'
>>> del IUPAC['A']
>>> IUPAC
[('C', 'Cys'), ('E', 'Glu'), ('F', 'Phe')]
```

# Conjuntos

```
>>> first_set = set(['CP0140.1','EF3613.1','EF3616.1'])
>>> first_set = set()
>>> first_set.add('CP0140.1')
>>> first_set.add('EF3613.1')
>>> first_set.add('EF3616.1')
>>> first_set
set(['CP0140.1','EF3613.1','EF3616.1'])
```

# Conjuntos

```
>>> first_set = set(['CP0140.1', 'EF3613.1', 'EF3616.1'])
>>> other_set = set(['CP0140.2', 'EF3613.1', 'EF3616.2'])
>>> common = first_set.intersection(other_set)
>>> common = first_set & other_set
>>> common
set(['EF3613.1'])
```

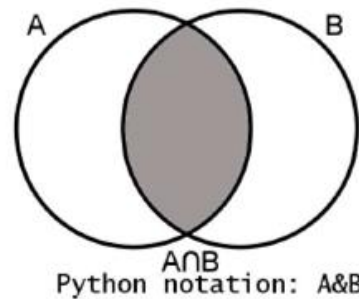


FIGURE 3.1: Intersection.

# Conjuntos

```
>>> first_set.union(other_set)
set(['EF3616.2', 'EF3613.1', 'EF3616.1', 'CP0140.1', 'CP0140.2'])
>>> first_set | other_set
set(['EF3616.2', 'EF3613.1', 'EF3616.1', 'CP0140.1', 'CP0140.2'])
```

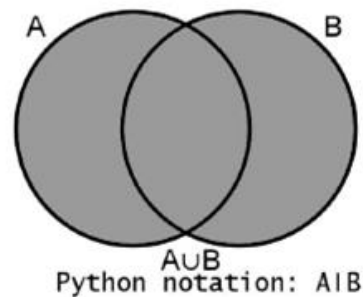
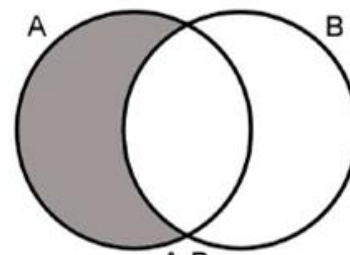


FIGURE 3.2: Union.



# Conjuntos

```
>>> first_set.difference(other_set)
set(['CP0140.1', 'EF3616.1'])
>>> first_set - other_set
set(['CP0140.1', 'EF3616.1'])
```



A-B  
Python notation: A-B

FIGURE 3.3: Difference.

# Conjuntos

```
>>> first_set.symmetric_difference(other_set)
set(['EF3616.2', 'CP0140.1', 'CP0140.2', 'EF3616.1'])
>>> first_set ^ other_set
set(['EF3616.2', 'CP0140.1', 'CP0140.2', 'EF3616.1'])
```

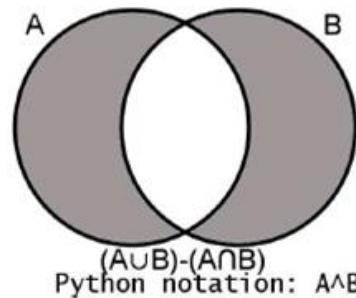


FIGURE 3.4: Symmetric difference.

# Indentação

#Espaços em branco fazem a indentação

```
if(x==10):  
    print("eh dez")  
else:  
    print("naum eh dez")
```

# Estrutura de Decisão - if

```
if <test>:  
    <bloco do TRUE>  
else:  
    <bloco do FALSE>
```

# Estrutura de Decisão - switch

Não há “switch”!

```
if <test>:  
    <bloco do TRUE>  
elif <test>:  
    <bloco do FALSE>  
elif <test>:  
    <bloco do FALSE>  
(...)  
else:  
    <bloco do FALSE>
```

# Estrutura de Repetição - for

```
for <indice> in <objeto>:  
    <bloco do for>  
    ...  
    ...  
    ...  
    if <test>: break  
    if <test>: continue  
else:  
    <bloco caso não ocorra break>
```

# Estrutura de Repetição - while

```
while <test>
    <bloco do for>
    ...
    ...
    ...
    if <test>: break
    if <test>: continue
else:
    <bloco caso não ocorra break>
```

# Pass - Não fazer nada!

```
if <test>:  
    pass  
else:  
    <bloco do true>
```



# Leitura/Escreita de Arquivos

```
fh = open('/home/sb/seqA.fas')
name = fh.readline()[1:-1]
sequence = ""
for line in fh:
    sequence += line.replace("\n", "")
print ("The name is: %s"%name)
print ("The sequence is: %s"%sequence)
fh.close()
```

# Leitura/Escreita de Arquivos

```
fh = open('/home/sb/bioinfo/seqA.fas')
myfile = fh.read() #myfile is a string
name = myfile.split('\n')[0][1:]
sequence = ''.join(myfile.split('\n')[1:])
print("The name is: %s"%name)
print("The sequence is: %s"%sequence)
fh.close()
```

# Leitura/Escreita de Arquivos

```
fh = open('/home/sb/seqA.fas')
FirstLine = fh.readline()
name = FirstLine[1:-1]
sequence = ""
while True:
    line = fh.readline()
    if line=="":
        break
    sequence += line.replace("\n","")
print("The name is: %s"%name)
print("The sequence is: %s"%sequence)
fh.close()
```

# Leitura/Escreita de Arquivos

```
import csv
tlen=0;n=0
lines = csv.reader(open('B1.csv'))
lines.next()
for line in lines:
    tlen += int(line[1])
    n += 1
print(tlen/float(n))
```

```
csv.reader(open("data.csv"), delimiter=',')
csv.reader(open("data.csv"),
dialect='excel')
csv.Sniffer().sniff(open('data.csv').read())
```

# Leitura/Escreva de Arquivos

```
fh = open('prot.fas')
fh.readline()
sequence = ""
for line in fh:
    sequence += line[:-1].upper()
fh.close()
charge = -0.002
AACharge={"C":-.045,"D":-.999,"E":-.998,"H":.091,
          "K":1,"R":1,"Y":-.001}
for aa in sequence:
    charge += AACharge.get(aa,0)
fhout = open('out.txt','w')
fhout.write(str(charge))
fhout.close()
```

# Outros Módulos

```
>>> import os
>>> os.getcwd()
>>> os.chdir('..')
>>> os.getcwd()
>>> os.listdir('/home/sb/bioinfo/seqs')
>>> os.path.isfile('/home/sb')
False
>>> os.path.isdir('/home/sb')
True
>>> os.remove('/home/sb/bioinfo/seqs/ms115.ab1')
>>> os.rename('/home/sb/seqs/readme.txt', '/home/sb/Readme')
>>> os.mkdir('/home/sb/processed-seqs')
>>> os.path.join(os.getcwd(), "images")
>>> os.path.exists(os.path.join(os.getcwd(), "images"))
False
>>> os.path.split('/home/sb/seqs/ms2333.ab1')
('/home/sb/seqs', 'ms2333.ab1')
>>> os.path.splitext('/home/sb/seqs/ms2333.ab1')
('/home/sb/seqs/ms2333', '.ab1')
```

# Outros Módulos

```
import os
mypath='/home/sb/bioinfo/test/'
AACharge = {"C":-0.045,"D":-0.999,"E":-0.998,
"H":0.091,"K":1,"R":1,"Y":-0.001}
for x in os.listdir(mypath):
    if os.path.splitext(x)[1]=='.fas':
        fh = open(os.path.join(mypath,x),'U')
        name = fh.readline()[1:-1]
        seq = ""
        for line in fh:
            seq = seq + line[:-1].upper()
        fh.close()
        charge = -0.002
        for aa in seq:
            charge += AACharge.get(aa,0)
        fh = open(os.path.join(mypath,'netvalues.txt'),'a')
        fh.write("%s,%s\n"%(name,charge))
        fh.close()
```

# Funções

The Python logo, consisting of two interlocking snakes, one blue and one yellow, is positioned in the background of the slide.

```
def FunctionName(argument1, argument2, ...):  
    """ Optional Function description (Docstring) """  
    ... FUNCTION CODE ...  
    return DATA
```



# Funções

```
def chargeandprop(AAseq):  
    """ Returns the net charge of a protein sequence  
    and proportion of charged amino acids """  
    protseq = AAseq.upper()  
    charge = -0.002  
    cp = 0  
    AACharge={"C":-0.045,"D":-0.999,"E":-0.998,"H":0.091,  
              "K":1,"R":1,"Y":-0.001}  
    for aa in protseq:  
        charge += AACharge.get(aa,0)  
        if aa in AACharge:  
            cp += 1  
    prop = 100.*cp/len(AAseq)  
    return (charge,prop)
```

```
>>> chargeandprop("QTALLVVLVLLAVALQATEAGPYGA")  
(-1.0009999999999999, 8.0)
```

```
>>> chargeandprop("EEARGPLRGKGDQKSAVSQKPRSRGILH")  
(4.0940000000000003, 39.285714285714285)
```

# Funções

```
def savelist(L, fname="temp.txt"):
    """ A list (L) is saved to a file (fname) """
    fh = open(fname, "w")
    for x in L:
        fh.write(str(x)+"\n")
    fh.close()
    return None
```

```
mylist = ['MS233', 'MS772', 'MS120', 'MS93', 'MS912']
>>> savelist(mylist)
```

# Funções

```
def commandline(name, **parameters):  
    line = ""  
    for parname, parvalue in parameters.iteritems():  
        line = line + " -" + parname + " " + parvalue  
    return name+line
```

```
>>> commandline("formatdb",t="Caseins",i="indata.fas")  
'formatdb -i indata.fas -t Caseins'  
>>> commandline("formatdb",t="Caseins",i="indata.fas",p="F")  
'formatdb -i indata.fas -p F -t Caseins'
```

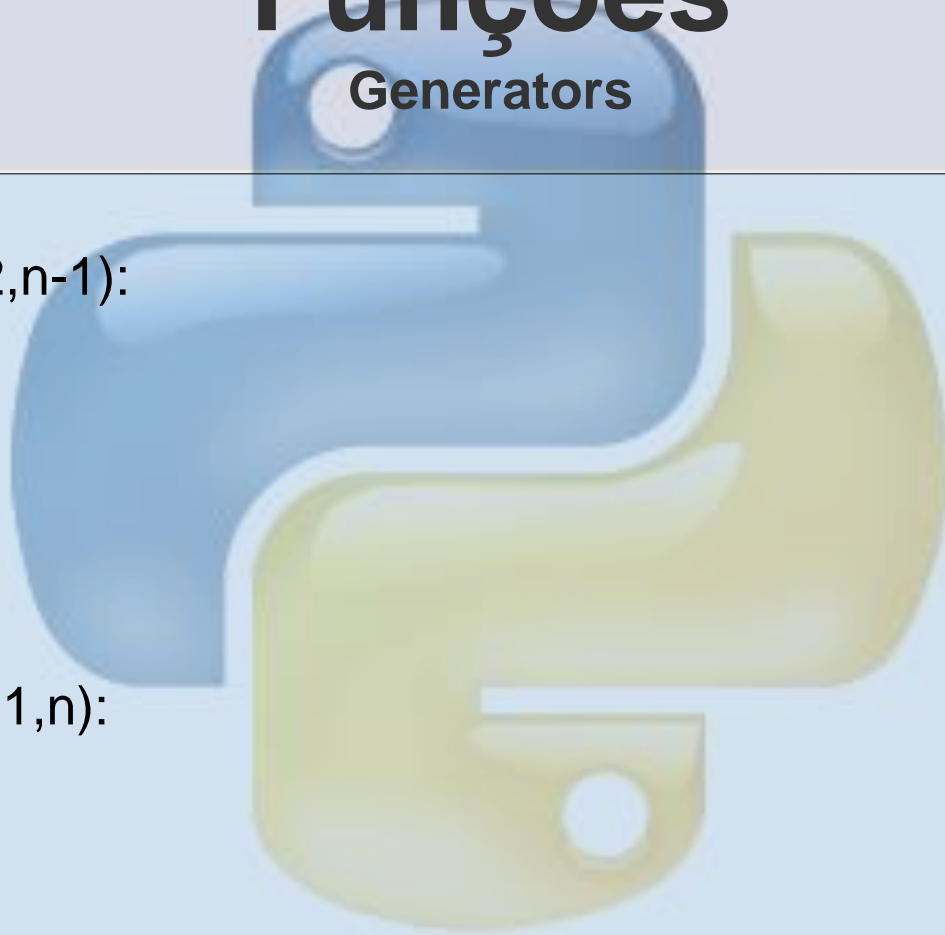
# Funções

## Generators

```
def isprime(n):  
    for i in range(2,n-1):  
        if n%i == 0:  
            return False  
    return True
```

```
def putn(n):  
    p = []  
    for i in xrange(1,n):  
        if isprime(i):  
            p.append(i)  
    return p
```

```
def gputn(n):  
    for i in xrange(1,n):  
        if isprime(i):  
            yield i
```



# Módulos



```
>>> import os  
>>> os.getcwd()
```

```
>>> from os import getcwd  
>>> getcwd()
```

```
>>> from os import *  
>>> getcwd()
```

```
>>> import xml.etree.ElementTree as ET  
>>> tree = ET.parse("/home/sb/bioinfo/smallUniprot.xml")
```

```
>>> import sys  
>>> sys.path  
['/home/sb', '/usr/local/bin']  
>>> sys.path.append("/home/sb/MyPyModules")
```

```
$ sudo easy_install <module_name>
```

# Tratamento de Excessão

```
import os
while True:
    try:
        iname = raw_input("Enter input filename: ")
        oname = raw_input("Enter output filename: ")
        fh = open(iname)
            line = fh.readline()
        fh.close()
        value = line.split('\t')[0]
        fw = open("/home/sb/"+oname,"w")
        fw.write(str(int(value)*.2))
        fw.close()
    except IOError, (errno,errmsg):
        if errno==13:
            print "Can't write to outfile."
        elif errno==2:
            print "File not exist"
    except ValueError, strerror:
        if "substring not found" in strerror.message:
            print "There is no tab"
        elif "invalid literal for int" in strerror.message:
            print "The value can't be converted to int"
    else:
        print "Thank you!. Everything went OK."
        break
```

# Tratamento de Excessão

```
def avg(numbers):  
    if not numbers:  
        raise ValueError("Please enter at least one element")  
    return sum(numbers)/len(numbers)
```

```
class NotDNAException(Exception):  
    """ A user-defined exception. """  
    def __init__(self, dna):  
        self.dna = dna  
    def __str__(self):  
        for nt in self.dna:  
            if nt not in 'atcg':  
                return nt
```

# Orientação a Objetos

```
class Sequence:
    TranscriptionTable = {"A":"U","T":"A","C":"G","G":"C"}
    def __init__(self, seqstring):
        self.seqstring=seqstring.upper()
    def transcription(self):
        tt = ""
        for x in self.seqstring:
            if x in 'ATCG':
                tt += self.TranscriptionTable[x]
        return tt
```



# Orientação a Objetos

```
>>> DangerousVirus=Sequence('atggagagccttggttcttggtgtcaa')
>>> DangerousVirus.seqstring
'ATGGAGAGCCTTGTTCTTGGTGTCAA'
>>> HarmlessVirus=Sequence('aatgctactactattagtagaattgatgcc')
>>> HarmlessVirus.seqstring
'AATGCTACTACTATTAGTAGAATTGATGCCA'
>>> DangerousVirus.transcription()
'GCUAAGAGCUCGCGUCCUCAGAGUUUAGGA'
```

# Orientação a Objetos

```
class TestClass:
    def a(self):
        pass
    def __b(self):
        # mangled to _TestClass__b
        pass
```

```
>>> MyObject = TestClass()
```

```
>>> MyObject.a()
```

```
>>> MyObject.__b()
```

```
Traceback (most recent call last):
```

```
File "<pyshell#14>", line 1, in <module>
```

```
    MyObject.__b()
```

```
AttributeError: TestClass instance has no attribute '__b'
```

# Expressões Regulares

```
>>> import re
>>> mo = re.search("hello","Hello world, hello Python!")
>>> mo.group()
'hello'
>>> mo.span()
(13, 18)
>>> re.findall("[Hh]ello","Hello world, hello Python,!")
['Hello', 'hello']
>>> rgx = re.compile("[Hh]ello")
>>> rgx.findall("Hello world, hello Python,!")
['Hello', 'hello']
>>> mos = re.finditer("[Hh]ello","Hello world, hello Python,!")
>>> for x in mos:
            print x.group()
            print x.span()
>>> mo = re.match("hello", "Hello world, hello Python!")
>>> print mo
None
```

# Expressões Regulares

```
import re
regex = re.compile(' \d|\n|\t')
seq = ""
for line in open('pMOSBlue.txt'):
    seq += regex.sub("",line)
print seq
```

```
ATGACCATGA TTACGCCAAG CTCTAATACG ACTCACTATA GGGAAAGCTT GCATGCCTGC
AGGTCGACTC TAGAGGATCT ACTAGTCATA TGGATATCGG TCCCCGGGT ACCGAGCTCG
AATTCACTGG CCGTCGTTTT
```

```
ATGACCATGATTACGCCAAGCTCTAATACGACTCACTATAGGGAAAGCTTGCATGCCTGC
AGGTCGACTCTAGAGGATCTACTAGTCATATGGATATCGGATCCCCGGGTACCGAGCTCG
AATTCACTGGCCGTCGTTTT
```

# BioPython



- Biopython (Agosto de 1999)
- Iniciado por Jeff Chang e Andrew Dalke
- Tarefas repetitivas no processo de análise
- Inspirado no BioPerl
- Open Bioinformatics Foundation
  - Bio\*
- Código disponível na internet
  - <http://www.biopython.org>
- Usuários podem contribuir com o projeto
  - Sugestões de funcionalidades
  - Propostas de código

# BioPython

## Componentes (Classes)

- 
- The BioPython logo is a stylized Python logo, consisting of two interlocking snakes. The left snake is blue and the right snake is yellow. The logo is positioned in the background of the slide, behind the list of components.
- Alphabet
  - Seq
  - MutableSeq
  - SeqRecord
  - Alignment
  - ClustalW
  - SeqIO
  - AlignIO
  - BLAST
  - Entrez
  - PDB
  - PROSITE
  - Restriction
  - SeqUtils
  - Sequencing
  - SwissProt

# BioPython

## Alphabet

```
>>> import Bio.Alphabet
>>> Bio.Alphabet.ThreeLetterProtein.letters
['Ala', 'Asx', 'Cys', 'Asp', 'Glu', 'Phe', 'Gly', 'His', '<=
'Ile', 'Lys', 'Leu', 'Met', 'Asn', 'Pro', 'Gln', 'Arg', '<=
'Ser', 'Thr', 'Sec', 'Val', 'Trp', 'Xaa', 'Tyr', 'Glx']
>>> from Bio.Alphabet import IUPAC
>>> IUPAC.IUPACProtein.letters
'ACDEFGHIKLMNPQRSTVWY'
>>> IUPAC.unambiguous_dna.letters
'GATC'
>>> IUPAC.ambiguous_dna.letters
'GATCRYWSMKHBVDN'
>>> IUPAC.ExtendedIUPACProtein.letters
'ACDEFGHIKLMNPQRSTVWYBXZ'
>>> IUPAC.ExtendedIUPACDNA.letters
'GATCBDSW'
```

# BioPython

## Seq

```
>>> from Bio.Seq import Seq
>>> import Bio.Alphabet
>>> seq = Seq('CCGGGTT',Bio.Alphabet.IUPAC.unambiguous_dna)
>>> seq.transcribe()
Seq('CCGGGUU', IUPACUnambiguousRNA())
>>> seq.translate()
Seq('PG', IUPACProtein())
>>> rna_seq = Seq('CCGGGUU',Bio.Alphabet.IUPAC.unambiguous_rna)
>>> rna_seq.transcribe()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "/home/sb/Seq.py", line 520, in transcribe
raise ValueError("RNA cannot be transcribed!")
ValueError: RNA cannot be transcribed!
>>> rna_seq.translate()
Seq('PG', IUPACProtein())
>>> rna_seq.back_transcribe()
Seq('CCGGGTT', IUPACUnambiguousDNA())
```



# BioPython

## MutableSeq

```
>>> seq = Seq('CCGGGTAAACGTA',Bio.Alphabet.IUPAC.unambiguous_dna)
>>> seq[:5]
Seq('CCGGG', IUPACUnambiguousDNA())
>>> len(seq)
13
>>> print seq
CCGGGTAAACGTA
>>> seq[0]='T'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AttributeError: 'Seq' instance has no attribute '__setitem__'
>>> mut_seq = seq.tomutable()
>>> mut_seq
MutableSeq('CCGGGTT', IUPACUnambiguousDNA())
>>> mut_seq[0]='T'
>>> mut_seq
MutableSeq('TCGGGTT', IUPACUnambiguousDNA())
```

# BioPython

## MutableSeq

```
>>> mut_seq.reverse()
>>> mut_seq
MutableSeq('TTGGGCT', IUPACUnambiguousDNA())
>>> mut_seq.complement()
>>> mut_seq
MutableSeq('AACCCGA', IUPACUnambiguousDNA())
>>> mut_seq.reverse_complement()
>>> mut_seq
MutableSeq('TCGGGTT', IUPACUnambiguousDNA())
```

# BioPython

## SeqRecord

```
>>> from Bio.SeqRecord import SeqRecord
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import generic_protein
>>> rec = SeqRecord(Seq("mdstnvrsgmksrkkkpktviddddcmtcsacqs"+
"klvkisditkvsldyintmrgntlacaacgssklIndfas", generic_protein),
id="P20994.1", name="P20994", description="Protein A19",
dbxrefs=["Pfam:PF05077", "InterPro:IPR007769", "DIP:2186N"])
>>> rec.annotations["note"] = "A simple note"
```

# BioPython

## Alignment

```
>>> # Import all required classes
... from Bio import Alphabet
>>> from Bio.Alphabet import IUPAC
>>> from Bio.Align.Generic import Alignment
>>> from Bio.Seq import Seq
>>> # Create and name our two sequences
... seq1 = 'MHQAIFIYQIGYPLKSGYIQSIRSPEYDNW'
>>> seq2 = 'MH--IFIYQIGYALKSGYIQSIRSPEY-NW'
>>> # Initialize an alignment object
... a = Alignment(Alphabet.Gapped(IUPAC.protein))
>>> # Add the sequences to this alignment object
... a.add_sequence("asp",seq1)
>>> a.add_sequence("unk",seq2)
>>> print a
Gapped(IUPACProtein(), '-') alignment with 2 rows and 30 columns
MHQAIFIYQIGYPLKSGYIQSIRSPEYDNW asp
MH--IFIYQIGYALKSGYIQSIRSPEY-NW unk
```

# BioPython

## ClustalW

```
>>> from Bio.Clustalw import MultipleAlignCL
>>> cl = MultipleAlignCL('inputfile.fasta')
>>> cl.set_output('cltest.txt')
>>> print("Command line: %s"%cl)
Command line: clustalw inputfile.fasta -OUTFILE=cltest.txt
>>> clpath='c:\\windows\\program file\\clustal\\clustalw.exe'
>>> cl = MultipleAlignCL('inputfile.fasta',command=clpath)
>>> from Bio.Clustalw import do_alignment
>>> align = do_alignment(cl)
#####
>>> from Bio.Clustalw import MultipleAlignCL
>>> cl = MultipleAlignCL('inputfile.fasta')
>>> cl.gap_open_pen=5
>>> cl.gap_ext_pen=3
>>> cl.new_tree='outtree.txt'
>>> print(cl)
clustalw inputfile.fasta -NEWTREE=outtree.txt -align -GAPOPEN=5<=
-GAPEXT=3
```

# BioPython

## SeqIO

```
>>> ##### LEITURA DE ARQUIVOS
>>> from Bio import SeqIO
>>> f_in = open('/home/sb/bioinfo/a19.gbk')
>>> SeqIO.parse(f_in, 'genbank').next()
SeqRecord(seq=Seq('MDSTNVRSGMKSRRKKPKTTVIDDDDDDCMTCSACQSKLV
KISDIT<=
KVSLDYINT...FAS', IUPACProtein()), id='P20994.1', name='P20994',<=
description='Protein A19.', dbxrefs=[])
>>> from Bio import SeqIO
>>> f_in = open('/home/sb/bioinfo/a19.gbk')
>>> SeqIO.parse(f_in, 'genbank').next()
SeqRecord(seq=Seq('MDSTNVRSGMKSRRKKPKTTVIDDDDDDCMTCSACQSKLV
KISDIT<=
KVSLDYINT...FAS', IUPACProtein()), id='P20994.1', name='P20994',<=
description='Protein A19.', dbxrefs=[])
```

# BioPython

## SeqIO

**TABLE 10.2:** Sequence and Alignment Formats

Format name	Description	Alignment - Sequence
ace	Reads the contig sequences from an ACE assembly file.	S
clustal	Output from Clustal W or X	A
embl	The EMBL flat file format.	S
emboss	The “pairs” and “simple” alignment format from the EMBOSS tools.	A
fasta	A simple format where each record starts with an identifier line starting with a “>” character, followed by lines of sequence.	A/S
fasta-m10	Alignments output by Bill Pearson’s FASTA tools when used with the -m 10 command line option.	A
genbank	The GenBank or GenPept flat file format.	S
ig	IntelliGenetics file format, also used by MASE.	A/S
nexus	Used by MrBayes and PAUP. See also the module Bio.Nexus which can also read any phylogenetic trees in these files.	A
phd	Output from PHRED.	S
phylip	Used by the PHYLIP tools.	A
stockholm	Used by PFAM.	A
swiss	Swiss-Prot (UniProt) format.	S
tab	Simple two column tab separated sequence files.	S

# BioPython

## SeqIO

```
>>> ##### ESCRITA DE ARQUIVOS
>>> from Bio import SeqIO
>>> from Bio.Seq import Seq
>>> from Bio.SeqRecord import SeqRecord
>>> fh = open('NC2033.txt')
>>> f_out = open('NC2033.fasta', 'w')
>>> rawseq = fh.read().replace('\n', '')
>>> #record = [SeqRecord(Seq(rawseq), 'NC2033.txt', '', '')]
>>> record = (SeqRecord(Seq(rawseq), 'NC2033.txt', '', ''),)
>>> SeqIO.write(record, f_out, 'fasta')
>>> f_out.close()
>>> fh.close()
>>>
>>> from Bio import SeqIO
>>> fo_handle = open('myseqs.fasta', 'w')
>>> readseq = SeqIO.parse(open('myseqs.gbk'), "genbank")
>>> SeqIO.write(readseq, fo_handle, "fasta")
>>> fo_handle.close()
```



# BioPython

## AlignIO

```
fi = open('/home/sb/bioinfo/example.aln')
fo = open('/home/sb/bioinfo/example.phy', 'w')
align = AlignIO.read(fi, "clustal")
AlignIO.write([align], fo, "phylip")
fo.close()
```

# BioPython

BLAST

- Basic Local Alignment Search Tool (BLAST)
- Programa de busca de similaridades entre sequencias
- Busca uma consulta em um banco de dados
- Execução remota (NCBI) e local

# BioPython

## BLAST

```
# blastall(blast executable, program name, database, input file, [align_view=7], [parameters])
```

```
>>> from Bio.Blast import NCBIStandalone as BLAST
>>> b_exe = '/home/sb/blast-2.2.20/bin/blastall'
>>> f_in = 'seq3.txt'
>>> b_db = '/home/sb/blast-2.2.20/data/TAIR8cds'
>>> rh, eh = BLAST.blastall(b_exe, "blastn", b_db, f_in)
>>>
>>> rh.readline()
<?xml version="1.0"?>
>>> rh.readline()
'<!DOCTYPE BlastOutput PUBLIC "-//NCBI//NCBI BlastOutput/EN" <=
"http://www.ncbi.nlm.nih.gov/dtd/NCBI_BlastOutput.dtd">\n'
```

# BioPython

BLAST

```
>>> fh = open('testblast.xml','w')
>>> fh.write(rh.read())
>>> fh.close()
from Bio.Blast import NCBIXML
for blast_record in NCBIXML.parse(rh):
    # Do something with blast_record
```

# BioPython

BLAST

```
from Bio.Blast import NCBIXML
xmlfh = open('/home/sb/bioinfo/other.xml') # BLAST output file.
for record in NCBIXML.parse(xmlfh):
    for align in record.alignments:
        print align.title
```

```
gi|110804074|ref|NC_008258.1| Shigella flexneri 5 str. 8401
gi|89106884|ref|AC_000091.1| Escherichia coli W3110 DNA
gi|117622295|ref|NC_008563.1| Escherichia coli APEC O1
```

# BioPython

## Entrez - eUtils: Retrieving Bibliography

```
from Bio import Entrez
my_em = 'user@example.com'
db = "pubmed"
# Search de Entrez website using esearch from eUtils
# esearch returns a handle (called h_search)
h_search = Entrez.esearch(db=db, email=my_em,
term="python and bioinformatics")
# Parse the result with Entrez.read()
record = Entrez.read(h_search)
# Get the list of Ids returned by previous search
res_ids = record["IdList"]
# For each id in the list
for r_id in res_ids:
    # Get summary information for each id
    h_summ = Entrez.esummary(db=db, id=r_id, email=my_em)
    # Parse the result with Entrez.read()
    summ = Entrez.read(h_summ)
    print(summ[0]['Title'])
    print(summ[0]['DOI'])
    print('=====')
```

# BioPython

## Entrez - eUtils: Gene Information

```
from Bio import Entrez
my_em = 'user@example.com'
db = "gene"
term = 'cobalamin synthase homo sapiens'
h_search = Entrez.esearch(db=db, email=my_em, term=term)
record = Entrez.read(h_search)
res_ids = record["IdList"]
for r_id in res_ids:
    h_summ = Entrez.esummary(db=db, id=r_id, email=my_em)
    summ = Entrez.read(h_summ)
    print(r_id)
    print(summ[0]['Description'])
    print(summ[0]['Summary'])
    print('=====')
```

# BioPython

## PDB (Protein Database)

```
import gzip
from Bio.PDB.PDBParser import PDBParser
def disorder(structure):
    for chain in structure[0].get_list():
        for residue in chain.get_list():
            for atom in residue.get_list():
                if atom.is_disordered():
                    print residue, atom
    return None

pdbfn = '/home/sb/bioinfo/pdb1apk.ent.gz'
handle = gzip.GzipFile(pdbfn)
parser = PDBParser()
structure = parser.get_structure("test", handle)
disorder(structure)
```



# BioPython

## PROSITE

```
from Bio import Prosite
handle = open("prosite.dat")
records = Prosite.parse(handle)
for r in records:
    print(r.accession)
    print(r.name)
    print(r.description)
    print(r.pattern)
    print(r.created)
    print(r.pdoc)
    print("=====")
PS00001
ASN_GLYCOSYLATION
N-glycosylation site.
N-{P}-[ST]-{P}.
APR-1990
PDOC00001
=====
PS00004
CAMP_PHOSPHO_SITE
cAMP- and cGMP-dependent protein kinase phosphorylation site.
[RK](2)-x-[ST].
APR-1990
```

# BioPython

## Restriction

```
>>> from Bio import Restriction
>>> Restriction.EcoRI
EcoRI
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet.IUPAC import IUPACAmbiguousDNA
>>> alfa = IUPACAmbiguousDNA()
>>> gi1942535 = Seq('CGCGAATTCGCG', alfa)
>>> Restriction.EcoRI.search(gi1942535)
[5]
>>> Restriction.EcoRI.catalyse(gi1942535)
(Seq('CGCG', IUPACAmbiguousDNA()), Seq('AATTCGCG',
IUPACAmbiguousDNA()))
>>> enz1 = Restriction.EcoRI
>>> enz2 = Restriction.HindIII
>>> batch1 = Restriction.RestrictionBatch([enz1, enz2])
>>> batch1.search(gi1942535)
{EcoRI: [5], HindIII: []}
```

# BioPython

## Restriction

```
>>> dd = batch1.search(gi1942535)
>>> dd.get(Restriction.EcoRI)
[5]
>>> dd.get(Restriction.HindIII)
[]
>>> batch1.add(Restriction.EarI)
>>> batch1
RestrictionBatch(['EarI', 'EcoRI', 'HindIII'])
>>> batch1.remove(Restriction.EarI)
>>> batch1
RestrictionBatch(['EcoRI', 'HindIII'])
```

# BioPython

## SeqUtils – DNA Utils

•CG , GC skew

-Percentual de Purinas e Piridiminas

```
->>> from Bio.SeqUtils import GC
```

```
->>> GC('gacgatcggtattcgtag')
```

```
-50.0
```

•Melting Temperature

-Temperatura de Desnaturação

```
->>> from Bio.SeqUtils import MeltingTemp
```

```
->>> MeltingTemp.Tm_staluc('tgcagtacgtatcgt')
```

```
-42.211472744873447
```

# BioPython

## SeqUtils – DNA Utils

### .Checksum Algorithms

-Calcula Checksum dos arquivos (Integridade)

```
->>> from Bio.SeqUtils import CheckSum
```

```
->>> myseq = 'acaagatgccattgtcccccggcctcctgctgctgct'
```

```
->>> CheckSum.gcg(myseq)
```

```
-1149
```

```
->>> CheckSum.crc32(myseq)
```

```
--2106438743
```

```
->>> CheckSum.crc64(myseq)
```

```
-'CRC-A2CFDBE6AB3F7CFF'
```

```
->>> CheckSum.seguid(myseq)
```

```
-'9V7Kf19tfPA5TntEP75YiZEm/9U'
```

# BioPython

## SeqUtils - Protein Utils

```
from Bio.SeqUtils.ProtParam import ProteinAnalysis
from Bio.SeqUtils import ProtParamData
from Bio import SeqIO
fh = open('/home/sb/bioinfo/pdbaa')
for rec in SeqIO.parse(fh,'fasta'):
    myprot = ProteinAnalysis(str(rec.seq))
    print(myprot.count_amino_acids())
    print(myprot.get_amino_acids_percent())
    print(myprot.molecular_weight())
    print(myprot.aromaticity())
    print(myprot.instability_index())
    print(myprot.flexibility())
    print(myprot.isoelectric_point())
    print(myprot.secondary_structure_fraction())
    print(myprot.protein_scale(ProtParamData.kd, 9, .4))
fh.close()
```

# BioPython

## Sequencing - Phd Files (Phred)

```
import pprint
from Bio.Sequencing import Phd
fn = '/home/sb/bt/biopython-1.50/Tests/Phd/phd1'
fh = open(fn)
rp = Phd.RecordParser()
# Create an iterator
it = Phd.Iterator(fh,rp)
for r in it:
    # All the comments are in a dictionary
    pprint.pprint(r.comments)
    # Sequence information
    print('Sequence: %s' % r.seq)
    # Quality information for each base
    print('Quality: %s' % r.sites)
fh.close()
```

# BioPython

## Sequencing - Ace Files (CAP3,Phrap...)

```
import pprint
from Bio.Sequencing import Phd
fn = '/home/sb/bt/biopython-1.50/Tests/Phd/phd1'
fh = open(fn)
rp = Phd.RecordParser()
# Create an iterator
it = Phd.Iterator(fh,rp)
for r in it:
    # All the comments are in a dictionary
    pprint.pprint(r.comments)
    # Sequence information
    print('Sequence: %s' % r.seq)
    # Quality information for each base
    print('Quality: %s' % r.sites)
fh.close()
```



# BioPython

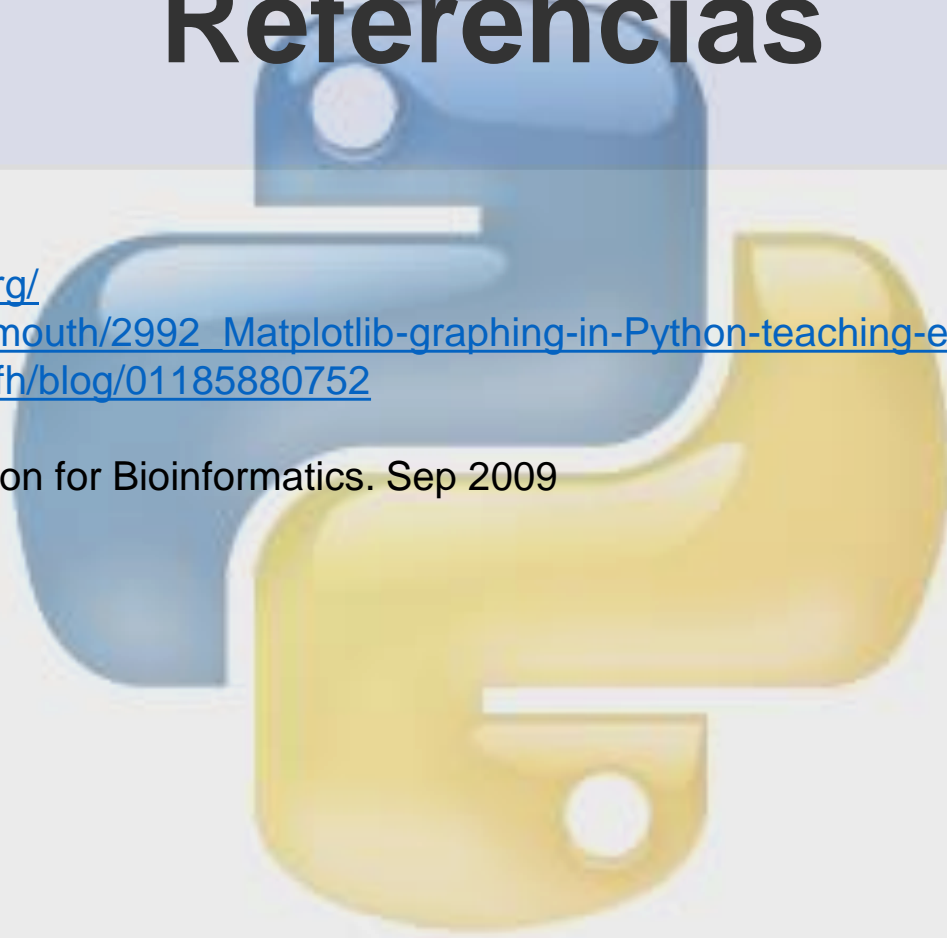
## SwissProt

```
from Bio import SwissProt
fh = open('spfile.txt')
records = SwissProt.parse(fh)
for record in records:
    print('Entry name: %s' % record.entry_name)
    print('Accession(s): %s' % ','.join(record.accessions))
    print('Keywords: %s' % ','.join(record.keywords))
    print('Sequence: %s' % record.sequence)
fh.close()
```

```
from Bio import SwissProt
fh = open('/home/sb/bioinfo/spfile.txt')
record = SwissProt.parse(fh).next()
for att in dir(record):
    if not att.startswith('__'):
        print(att,getattr(record,att))
```

# Referências

- [.http://www.python.org/](http://www.python.org/)
- [.http://diveintopython.org/](http://diveintopython.org/)
- [.http://www.wellho.net/mouth/2992\\_Matplotlib-graphing-in-Python-teaching-examples.html](http://www.wellho.net/mouth/2992_Matplotlib-graphing-in-Python-teaching-examples.html)
- [.http://logarithmic.net/pfh/blog/01185880752](http://logarithmic.net/pfh/blog/01185880752)
- [.http://www.scipy.org/](http://www.scipy.org/)
- .Bassi, Sebastian, Python for Bioinformatics. Sep 2009





Obrigado! \o/