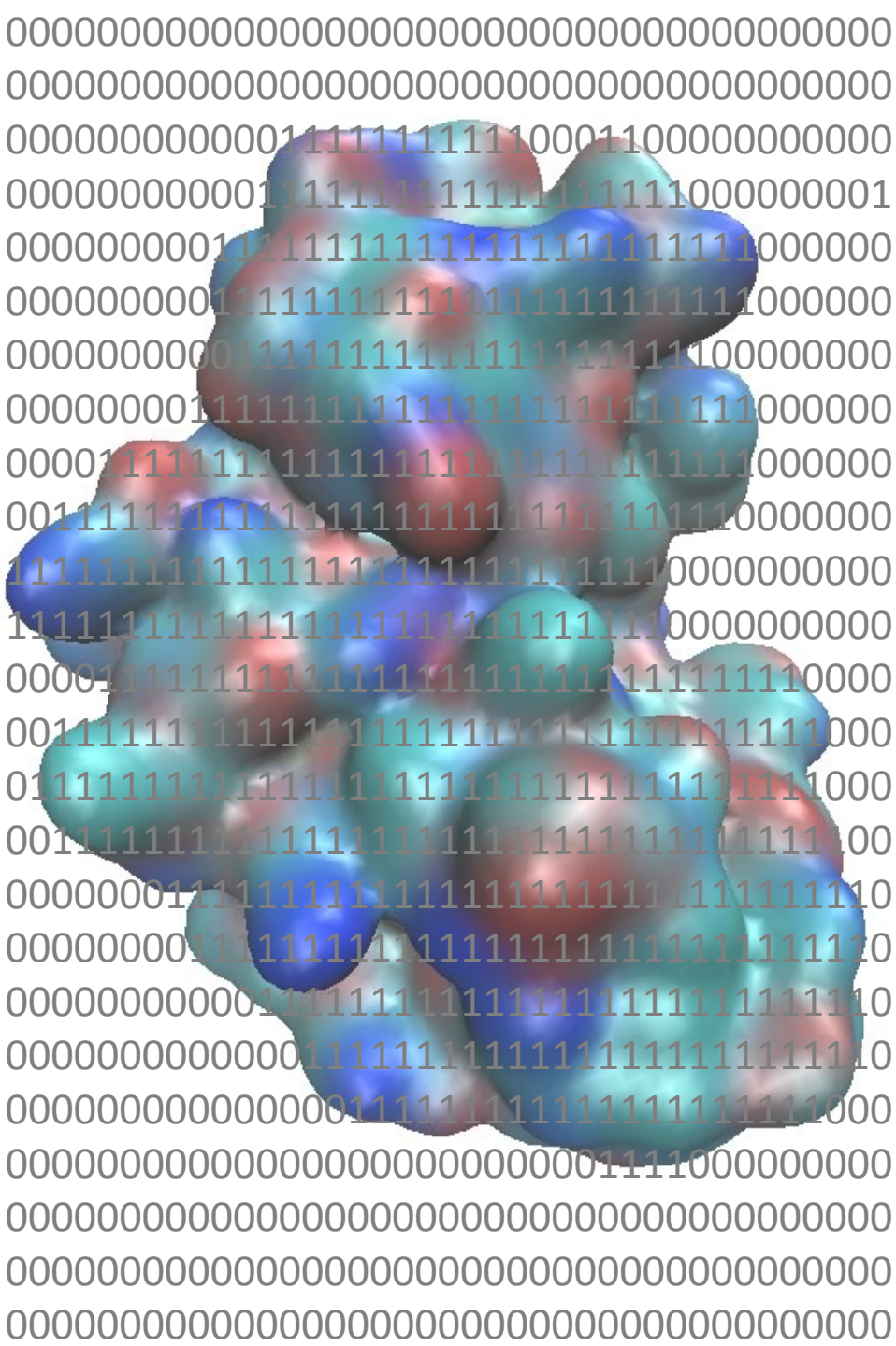


python™
www.python.org

Bioinformática Aplicada

Aula 9

python™
www.python.org

Read Genome

Programa: *readGenome.py*

Resumo
Programa para a leitura da sequência de bases de um genoma armazenado num arquivo FASTA. Após a leitura, será mostrada na tela as seguintes informações: identificador do FASTA, número de adeninas, número de timinas, número de citosinas, número de guaninas, número de bases não identificadas, número de citosinas e guaninas, número de adeninas e timinas, porcentagem de citosinas e guaninas, porcentagem de adeninas e timinas, porcentagem de bases não identificadas, número total de bases lidas e número de bases não identificadas.

A parte inicial faz a leitura do nome do arquivo FASTA, atribui seu conteúdo à variável *seqIn* e edita de forma a transformar uma lista em string sem *"\n"*, como já visto.

```
# Program to read a sequence in FASTA format
# Set initial value for count_bases
count_bases = 0

# Reads input file name
fastaIn = input("\nGive the input DNA file name => ")
fh = open(fastaIn,'r')  # Opens input file
seqIn = fh.readlines()  # Reads all of the lines in a file and returns them as elements in a list
fh.close()              # Closes input file
header = seqIn[0]       # Assigns first line to header

# Removes the item at the given position in the list, and return it.
seqIn.pop(0)     # Removes first element of the list (get rid of the first line)
# Transforms list in string
seq=''.join(seqIn)
# Removes all "/n" from the string seq
seq = seq.replace("\n","")
```

Lê o nome do arquivo de entrada

Lê o conteúdo do arquivo de entrada

Deleta a primeira linha arquivo de entrada

Transforma lista em string

Remove newlines da string

Agora contamos cada uma das bases com o método *.count()* aplicado à string *seq*. Assim, teremos o número total de cada base atribuído às variáveis específicas para contar as bases (*count_A*, *count_T*, *count_C* e *count_G*. Além disso, criamos a variável *count_X* para as bases não reconhecidas, conforme mostrado abaixo.

Conta as bases presentes na string

```
# Count bases
count_bases = len(seq)
count_C = seq.count("C")
count_G = seq.count("G")
count_A = seq.count("A")
count_T = seq.count("T")
count_X = count_bases - count_C - count_G - count_A - count_T
```

Por último, mostramos os resultados. Optamos por calcular as porcentagens diretamente na função *print()*, como mostrado abaixo.

Mostra os resultados na tela

```
# Shows percentage for each base in the random sequence
print("FASTA ID:",header)
print("Number of A: ",count_A)
print("Number of T: ",count_T)
print("Number of C: ",count_C)
print("Number of G: ",count_G)
print("Number of non-identified bases: ",count_X)
print("Number of C + G : ",count_C + count_G)
print("Number of A + T : ",count_A + count_T)
print("Percentage of C + G :",100*(count_C+count_G)/count_bases)
print("Percentage of A + T :",100*(count_A+count_T)/count_bases)
print("Percentage of non-identified bases :",100*(count_X)/count_bases)
print("Total bases: ",count_bases)
```

Ao rodarmos o programa para o arquivo *sequence.fasta*, com a sequência do genoma do *Mycobaterium tuberculosis*, temos os seguintes resultados:

```
Give the input DNA file name => sequence.fasta
FASTA ID: >gi|57116681|ref|NC_000962.2| Mycobacterium tuberculosis H37Rv chromosome, complete genome

Number of A:  758565
Number of T:  758379
Number of C:  1449985
Number of G:  1444603
Number of non-identified bases:  0
Number of C + G :  2894588
Number of A + T :  1516944
Percentage of C + G : 65.61412225956879
Percentage of A + T : 34.38587774043121
Percentage of non-identified bases : 0.0
Total bases:  4411532
```

Até o momento vimos dados do tipo sequência, que são strings e números, bem como **listas**, que são formas de colocarmos um conjunto de dados e relacioná-los com sua posição numa lista, por exemplo, à variável *my_list* foi atribuída uma lista com o código de três letras para os aminoácidos, como segue:

```
my_list =["ALA","ARG","ASN","ASP","CYS","GLU","GLN","GLY","HIS","ILE",
          "LEU","LYS","MET","PHE","PRO","SER","THR","TRP","TYR","VAL"]
```

Assim, se quisermos mostrar todo o conteúdo atribuído à variável *my_list,* basta usarmos um loop *for* e colocarmos uma função *print()*, como mostrado abaixo.

```
for aa in my_list:
    print(aa)
```

A execução do código mostrará um aminoácido por linha, o arquivo *showAA1.py* tem o código do programa.

Outra estrutura de dados disponível em Python é chamada **dicionário**. Funciona com se fosse um dicionário comum, onde temos associada a um verbete uma definição. Assim, na definição de um dicionário em Python, teremos pares, onde para uma **chave** colocada à esquerda, teremos um **valor** à direita. Vejamos um exemplo, o dicionário *my_dict* tem o código de uma letra para os aminoácidos (chaves) e, à cada chave foi atribuída uma string com o código de três letras (valor), como indicado abaixo.

```
my_dict ={"A": "ALA","R": "ARG","N": "ASN","D": "ASP",
        "C": "CYS","E": "GLU","Q": "GLN","G": "GLY",
        "H": "HIS","I": "ILE","L": "LEU","K": "LYS",
        "M": "MET","F": "PHE","P": "PRO","S": "SER",
        "T": "THR","W": "TRP","Y": "TYR","V": "VAL"
        }
```

Cada definição do dicionário é chamada de **item**, assim temos 20 itens em *my_dict.* Para chamarmos uma definição do dicionário, podemos usar a chave para recuperar o valor, como segue:

```
print(my_dict["A"])
```

A função *print(my_dict["A"])* mostrará o valor atribuído à string "A", ou seja, "ALA".

Vejamos o programa *showAA2.py*, que mostra os valores associados às chaves que trazem os códigos de uma letra.

```
my_list =["A","R","N","D","C","E","Q","G","H","I",
          "L","K","M","F","P","S","T","W","Y","V"]
my_dict ={"A": "ALA","R": "ARG","N": "ASN","D": "ASP",
        "C": "CYS","E": "GLU","Q": "GLN","G": "GLY",
        "H": "HIS","I": "ILE","L": "LEU","K": "LYS",
        "M": "MET","F": "PHE","P": "PRO","S": "SER",
        "T": "THR","W": "TRP","Y": "TYR","V": "VAL"
        }
for aa in my_list:
    print(my_dict[aa])
```

No código temos a lista *my_list* com os aminoácidos, com o código de uma letra. No dicionário, temos associado à cada código de uma letra o valor com a string com três letras. O loop *for* mostra para cada chave lida da lista *my_list,* o valor associado. O resultado é que temos os códigos de três letras para cada aminoácido mostrado na tela. Veja que na formação de qualquer dicionário, sempre temos um elemento à esquerda, chamado de chave, separado por dois pontos (:), associado a um valor, que pode ser um número, string ou lista.

Dicionários, como as listas, são de uso comum em programas desenvolvidos para bioinformática. Veremos um programa para calcular a massa molecular de uma proteína, a partir da leitura da sua sequência de aminoácidos armazenada num arquivo FASTA. O programa fará uso do seguinte dicionário.

```
aaMW = {"A": 71.0779,"R": 156.1857,"N": 114.1026,"D": 115.0874,
        "C": 103.1429,"E": 129.114,"Q": 128.1292,"G": 57.0513,
        "H": 137.1393,"I": 113.1576,"L": 113.1576,"K": 128.1723,
        "M": 131.1961,"F": 147.1739,"P": 97.1152,"S": 87.0773,
        "T": 101.1039,"W": 186.2099,"Y": 163.1733,"V": 99.1311
        }
```

Veja, na definição de um dicionário em Python, podemos deixar todos os itens numa linha somente, ou pularmos linhas, como no exemplo acima. Fica a critério do programador.

Massa molecular de proteínas (versão 1)

Programa: *proteinMW1.py*

Resumo
Programa para calcular a massa molecular de proteínas, a partir da sequência lida de um arquivo no formato FASTA. O usuário fornecerá o nome do arquivo FASTA. O programa calcula a massa molecular, a partir da informação sobre a estrutura primária da proteína. Será usado um dicionário, para armazenar a informação sobre a massa molecular de cada aminoácido.

Definimos um dicionário com a massa molecular de cada aminoácido (valor), associada a cada código de uma letra do aminoácido (chave)

```
# Program to calculate the molecular weight of a protein using the information stored
# in a FASTA file
mw = 0

# Source for residue molecular weights: http://www.matrixscience.com/help/aa_help.html (Accessed
# on May 8th 2014)
# To calculate the mass of a neutral peptide or protein, sum the residue masses plus the masses
# of the terminating
# groups (e.g. H at the N-terminus and OH at the C-terminus).
aaMW = {"A": 71.0779,"R": 156.1857,"N": 114.1026,"D": 115.0874,
        "C": 103.1429,"E": 129.114,"Q": 128.1292,"G": 57.0513,
        "H": 137.1393,"I": 113.1576,"L": 113.1576,"K": 128.1723,
        "M": 131.1961,"F": 147.1739,"P": 97.1152,"S": 87.0773,
        "T": 101.1039,"W": 186.2099,"Y": 163.1733,"V": 99.1311
        }
```

Dicionário com a massa molecular

Depois da leitura da sequência, como visto em programas anteriores, temos a string *seq*. Usamos um *loop for*, destacado em vermelho, para atribuirmos o valor da massa molecular de cada aminoácido à variável *mw*. Somamos ao final a massa molecular referente aos terminais da proteína. Por último, mostramos o resultado na tela.

Leitura e edição da sequência de aminoácidos

```
# Reads input file name
fastaIn = input("\nGive the input file name => ")
fh = open(fastaIn,'r')  # Opens input file
seqIn = fh.readlines()  # Reads all of the lines in a file and returns them as elements in a list
fh.close()              # Closes input file
# Removes the item at the given position in the list, and return it.
seqIn.pop(0)    # Removes first element of the list (get rid of the first line)
# Transforms list in string, so we will not get error with .replace()
seq=''.join(seqIn)
# Removes all "/n" from the string seq
seq = seq.replace("\n","")
```

Loop for para somar as massas dos aminoácidos

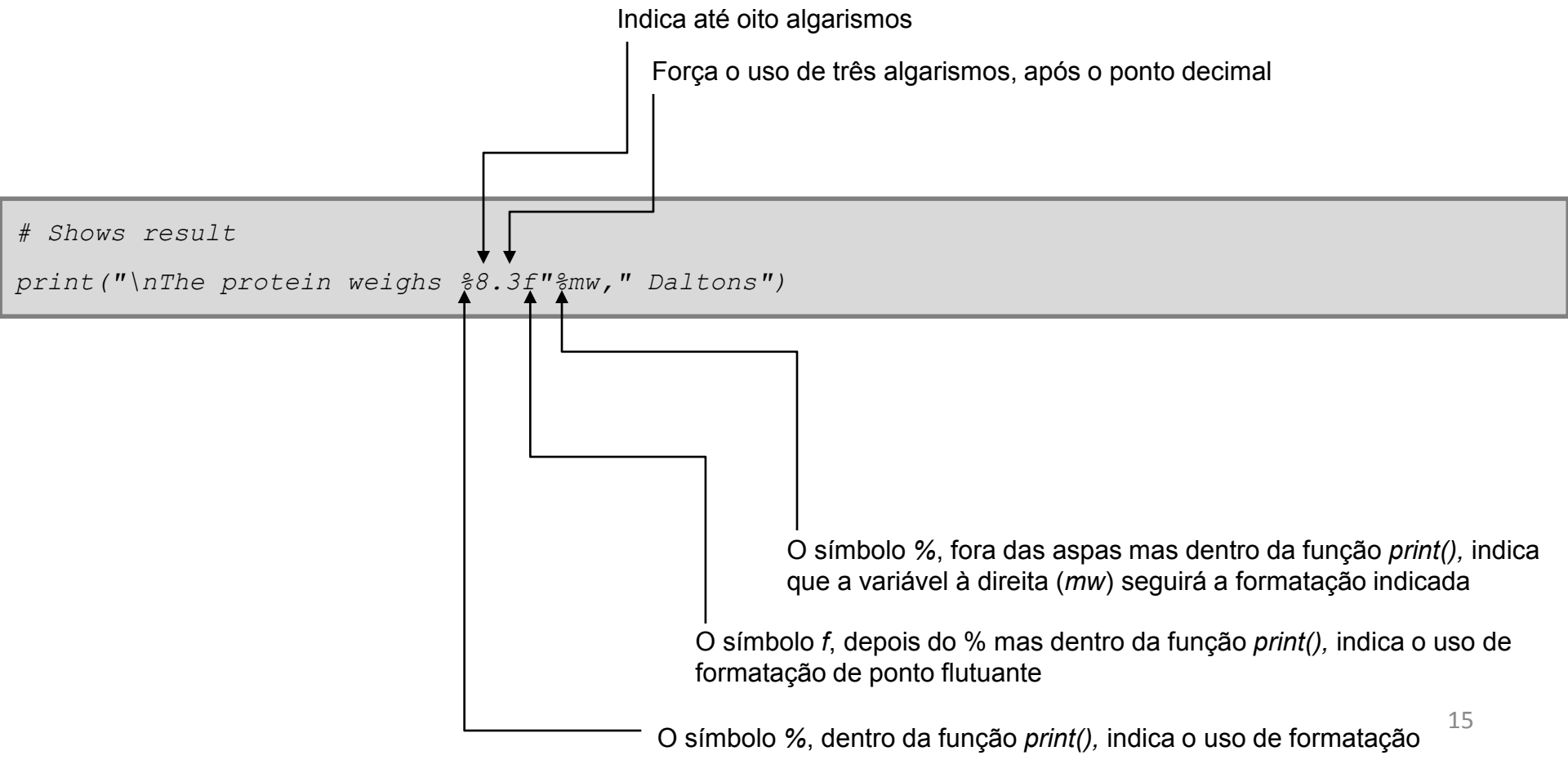
```
for aa in seq:
    mw += float(aaMW[aa])
mw = mw + 18.0148 # Sums water molecule
print("\nThe protein weighs",mw," Daltons")
```

Usando-se o programa para o arquivo FASTA *1kxy.fasta*, obtemos o resultado abaixo.

```
Give the input file name => 1kxy.fasta

The protein weighs 14312.020100000016  Daltons
```

Podemos formatar a saída da função *print()*, para definirmos o número de casas de um número de ponto flutuante (*float*) com o símbolo *%*. A linha abaixo traz uma função *print()*, para mostrar a massa molecular com até oito casas, sendo três casas após o ponto decimal. O Programa *proteinMW1a.py* traz esta modificação.



Usando-se o programa *proteinMW1a.py*, para o arquivo FASTA *1kxy.fasta*, obtemos o resultado abaixo.

```
Give the input file name => 1kxy.fasta

The protein weighs 14312.020  Daltons
```

Massa molecular de proteínas (versão 2)

Programa: *proteinMW2.py*

Resumo
Programa para calcular a massa molecular de proteínas e a porcentagem em massa de cada aminoácido presente na sequência. O programa lê a sequência de um arquivo no formato FASTA. O usuário fornecerá o nome do arquivo FASTA. O programa calcula a massa molecular e a porcentagem em massa dos aminoácidos, a partir da informação sobre a estrutura primária da proteína. Será usado um dicionário para armazenar a informação sobre a massa molecular de cada aminoácido.

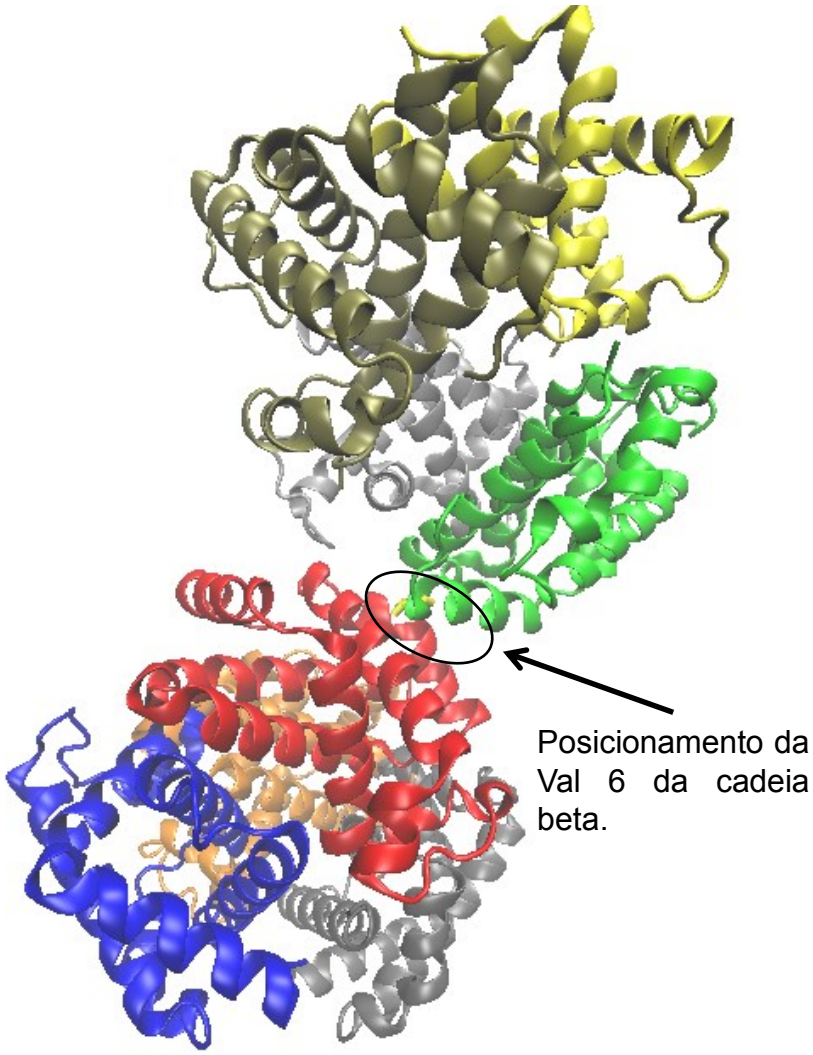
python™
www.python.org

Cálculo da porcentagem de aminoácidos hidrofóbicos
Programa: *aaHydro.py*

Resumo
Programa para calcular a porcentagem de resíduos de aminoácidos hidrofóbicos presentes na proteína. A informação sobre a estrutura primária é lida de um arquivo no formato FASTA padrão, cujo o nome é fornecido pelo usuário. Os resultados são mostrados na tela. Além de informações sobre o número total de aminoácidos presente na proteína, é mostrada a primeira linha do arquivo FASTA, que traz a identificação da proteína.

Resíduos hidrofóbicos têm uma leve tendência de apresentarem-se enterrados na estrutura da proteína. A presença de um aminoácido hidrofóbico, como a valina, na superfície da proteína, cria um ponto de contato desfavorável com o meio aquoso. Tal situação pode levar à formação de oligômeros de proteínas, como observado para a hemoglobina de indivíduos com anemia falciforme.

Referência: Harrington DJ, Adachi K, Royer Jr WE. The high resolution crystal structure of deoxyhemoglobin S. J.Mol.Biol 1997; 272: 398-407



Estrutura cristalográfica da hemoglobina com a mutação Glu->Val na posição 6 da cadeia beta. Na estrutura temos 2 tetrâmeros e a val 6 localiza-se na interface dos tetrâmetros (elipse), ocultando-a do solvente.
Código PDB: 2HBS

Abaixo temos 10 arquivos FASTA para a cadeia beta da hemoglobina humana. Teste os arquivos FASTA indicados e identifique aquele que apresenta a hemoglobina da anemia falciforme. Use como critério a porcentagem de resíduos hidrofóbicos. A cadeia beta da hemoglobina de indivíduos com anemia falciforme, apresenta uma mutação de glutamato (polar) para valina (hidrofóbico), assim terá uma porcentagem maior de aminoácidos hidrofóbicos.
Lista de arquivos a serem testados:

Hb1.fasta, Hb2.fasta, Hb3.fasta, Hb4.fasta, Hb5.fasta, Hb6.fasta, Hb7.fasta Hb8.fasta, Hb9.fasta, e Hb10.fasta .

.

-BRESSERT, Eli. SciPy and NumPy. Sebastopol: O'Reilly Media, Inc., 2013. 56 p.

-DAWSON, Michael. **Python Programming, for the absolute beginner.** 3ed. Boston: Course Technology, 2010. 455 p.

-HETLAND, Magnus Lie. **Python Algorithms. Mastering Basic Algorithms in the Python Language.** Nova York: Springer Science+Business Media LLC, 2010. 316 p.

-IDRIS, Ivan. **NumPy 1.5. An action-packed guide dor the easy-to-use, high performance, Python based free open source NumPy mathematical library using real-world examples. Beginner's Guide.** Birmingham: Packt Publishing Ltd., 2011. 212 p.

-KIUSALAAS, Jaan. **Numerical Methods in Engineering with Python.** 2ed. Nova York: Cambridge University Press, 2010. 422 p.

-LANDAU, Rubin H. **A First Course in Scientific Computing: Symbolic, Graphic, and Numeric Modeling Using Maple, Java, Mathematica, and Fortran90**. Princeton: Princeton University Press, 2005. 481p.

-LANDAU, Rubin H., PÁEZ, Manuel José, BORDEIANU, Cristian C. **A Survey of Computational Physics. Introductory Computational Physics.** Princeton: Princeton University Press, 2008. 658 p.

-LUTZ, Mark. **Programming Python.** 4ed. Sebastopol: O'Reilly Media, Inc., 2010. 1584 p.

-MODEL, Mitchell L. **Bioinformatics Programming Using Python**. Sebastopol: O'Reilly Media, Inc., 2011. 1584 p.

-TOSI, Sandro. **Matplotlib for Python Developers.** Birmingham: Packt Publishing Ltd., 2009. 293 p.

Última atualização:22 de maio 2019.