# Homework

Python-3

**Sup'Biotech 3**

Python

Pierre Parutto

November 21, 2016

## Preamble

### Document Property

| | |
|---|---|
| Authors | Pierre Parutto |
| Version | 1.0 |
| Number of pages | 8 |

### Contact

Contact the assistant team at: supbiotech-bioinfo-bt3@googlegroups.com

### Copyright

## Contents

1 Introduction 3
1.1 File Architecture . . . 3
1.2 Submission . . . 3
1.3 Cheating . . . 3

2 Example 3
2.1 Question . . . 3
2.2 Answer . . . 3
2.3 Testing your code . . . 4

3 Introduction 4

4 Sequences 4
4.1 Is equal (2 points) . . . 4
Example . . . 4
4.2 Seek And Destroy (2 points) . . . 4
Example . . . 5

5 Binary Search Trees 5
5.1 Count Num Odd (2 points) . . . 5
Example . . . 5
5.2 BST To List (3 points) . . . 5
Example . . . 5

6 Sequence Alignment 6
6.1 Collecting Gaps (2 points) . . . 6
Example . . . 6
6.2 Similarity From An Alignment (4 points) . . . 6
Example . . . 6
6.3 Similarity From Sequences (5 points) . . . 7
Example . . . 8

# 1 Introduction

In this homework we will apply the basic programming skills that you have just acquired to crack some biology-related questions.

## 1.1 File Architecture

**You must respect** the following architecture for your work:

```
login_l
├── AUTHORS
└── src
    └── files.py
```

You must have a folder named with your login, this folder must contain:

1. **A text file** named *AUTHORS* containing your first and last names.
2. **A folder** named *src* that contains your code files.

## 1.2 Submission

- Deadline: until **Wednesday December 7, 23h42**;
- Submission: **a zip file** named: *login_l.zip* to upload on the bioinformatics intranet.

**A bad architecture of your submission may result in a 2 points penalty.**

## 1.3 Cheating

Basically, **DO NOT CHEAT**. Your work will be automatically tested against cheating. If two people are detected as cheaters, they will receive the **grade 0** for the homework and the administration will be told of their attempt. All detected cheaters are treated equally, I do not care who wrote the code and who took it.

# 2 Example

Here is an example of how to answer an homework question.

## 2.1 Question

**File:** *toto.py*

Write the function called `my_sum(a: int, b:int) -> int` that returns the sum of `a` and `b`.

## 2.2 Answer

The content of the file *toto.py* is thus:

```python
def my_sum(a, b):
    return a + b
```

### 2.3 Testing your code

Although not mandatory in this first homework, **I strongly advise you to test your code**. You can do that by calling your function with some values and checking that the answer is what you expect. For example, your file *toto.py* becomes:

```python
def my_sum(a, b):
    return a + b

print(my_sum(5, 9) == 16)
print(my_sum(5, 0) == 5)
print(my_sum(5, -5) == 0)
```

And the Python output is:

```
True
True
True
```

**You must remove all your tests before you submit your code. In your code files I want only function declaractions and nothing else.**

## 3 Introduction

## 4 Sequences

**File:** *seqs.py*

### 4.1 Is equal (2 points)

Write the **recursive** function `is_equal(l1: list, l2: list) -> bool` that returns `True` if the two **ordered** lists `l1` and `l2` are equal (they contain only the same values in the same order) and `False` otherwise.

**Example**

```
>>> is_equal([1, 2, 3], [])
False
>>> is_equal([1, 2, 3], [1, 3])
False
>>> is_equal([1, 3, 5], [1, 3, 5])
True
```

### 4.2 Seek And Destroy (2 points)

Write the function `seek_and_destroy(s: str, m: str) -> str` that returns the string `s` where all the appearances of the motif `m` have been removed.

**Note :** `m` will never be the empty string and `len(m) <= len(s)`.

**Example**

```
>>> seek_and_destroy("ATGTG", "G")
"ATT"
>>> seek_and_destroy("ATGTG", "TG")
"A"
>>> seek_and_destroy("AAAAA", "A")
""
```

## 5 Binary Search Trees

**File:** *bst.py*

A Binary Search Tree (BST) is defined in Python as a list of tree elements:

```
[left son, value, right son]
```

Where `left son` and `right son` are themselves BSTs (represented as lists). Thus for a node `t`, `t[0]` gives the left son, `t[1]` the value of the node and `t[2]` the right son. An empty BST is represented as an empty list `[]`. In a BST, the value of a node must be greater than all the values contained in its left son and smaller than all the values contained in its right son.

### 5.1 Count Num Odd (2 points)

Write the **recursive** function `count_num_odd(t: list) -> int` that returns the number of odd values in the BST `t`.

**Example**

```
>>> count_num_odd([[], 5, []])
1
>>> count_num_odd([[[], 2, []], 5, [[], 8, []]])
1
>>> count_num_odd([[[[], 3, [[], 4, []]], 5, []], 7, []])
3
```

### 5.2 BST To List (3 points)

Write the **recursive** function `BST_to_list(t: list) -> list` that returns the **ordered** list of all the elements contained in the BST `t`.

**Example**

```
>>> BST_to_list([[[], 1, []], 5, [[], 7, []]])
[1, 5, 7]
>>> BST_to_list([[[[], 3, []], 4, [[], 5, []]], 8, []])
[3, 4, 5, 8]
>>> BST_to_list([[[[], 3, []], 4, []], 8, [[[[], 9, []], 10, []], 11, []]])
[3, 4, 8, 9, 10, 11]
```

## 6 Sequence Alignment

**File:** *align.py*

An alignment of two (nucleic or proteic) sequences consists in finding the superposition of these sequences that maximizes over all sites the number of identical residues. This value is called a similarity score. Alignment algorithms usually take into account three types of mutations: substitution, insertion and deletion. A substitution produces two different residues at the same position while insertion and deletion will produce gaps (represented by `-` characters) in one of the sequence. Considering these three types of mutations, the similarity score of the two superposed sequences $\alpha, \beta$, $s(\alpha, \beta)$ is defined as:

$$s(\alpha,\beta) = \sum_{i=0}^{|x|} \sigma(\alpha_i, \beta_i) - \sum_{g \in G(\alpha)} \gamma(g) \sum_{g \in G(\beta)} \gamma(g)$$

where $\sigma(a,b) = \begin{cases} m & \text{if } a = b \text{ and } (a \text{ and } b \neq \texttt{-}) \\ n & \text{if } a \neq b \text{ and } (a \text{ and } b \neq \texttt{-}) \\ 0 & \text{if } a \text{ or } b = \texttt{-} \end{cases}$ , $G(s)$ gives the ensemble of gaps in the sequence $s$ (it corresponds to the function `collect_gaps` defined below), $\gamma(g) = e \times |g|$ gives the score attributed to a gap of size $|g|$ and $m, n, e \in \mathbb{R}$. Usually, $m$, the score of a match is $> 0$ and $n$ the score of a substitution (mismatch) and $e$ the cost of a gap of size one are $< 0$.

### 6.1 Collecting Gaps (2 points)

A gap consists in one or more `-` (dash) characters in **one** of the sequences. The size of a gap $g$, denoted $|g|$, is the number of `-` characters composing it. A gap of size $n$ represents either $n$ insertions in the sequence $x$ if it is located in $y$ or $n$ deletions in $x$ if it is located in $x$.

Write the function `collect_gaps(s: str) -> list` that returns the list strings corresponding to all the gaps in the sequence `s`.

**Example**

```
>>> collect_gaps("ATTAG")
[]
>>> collect_gaps("ATT---AG-")
["---", "-"]
>>> collect_gaps("AT----AA--A-A-")
["----", "--", "-", "-"]
```

### 6.2 Similarity From An Alignment (4 points)

If the sequences are already aligned, it is easy to find their similarity score by computing $s(\alpha, \beta)$ as defined above.

Write the function `sim(alpha: str, beta: str, m: float, n: float, e: float) -> float` that given two aligned sequences `alpha`,`beta` and the three scoring parameters `m`, `n`, `e`, returns the similarity score $s(\alpha, \beta)$.

**Example**

```
>>> sim("ATGAC", "ATCCC", 1, -0.5, -0.5)
2
>>> sim("GC--GGG", "GAAAGGG", 1, -0.5, -0.5)
2.5
>>> sim("GCGCGC-", "G-----C", 1, -0.5, -0.5)
-2
```

### 6.3 Similarity From Sequences (5 points)

The Needleman-Wunsch algorithm allows to compute a measure of evolutionary-related similarity between two (not already aligned) sequences `x` and `y`.

To compute the similarity, the algorithm builds a matrix $S$ of size `(len(x)+1)`×`(len(y)+1)`, where each line corresponds to a character from the sequence $x$ from top to bottom ($s_{1,*}$ corresponds to $x_0$, $s_{2,*}$ to $x_1$, ...) and each column to a character from the sequence $y$ from left to right ($s_{*,1}$ corresponds to $y_0$, $s_{*,2}$ to $x_2$, ...). The first line and row correspond to the empty string for $x$ and $y$ respectively.

The algorithm takes five inputs:

1. $x$, the first sequence;
2. $y$, the second sequence;
3. $m$, the cost of seeing two identical characters at the same position (also called a match);
4. $n$, the cost of seeing two different characters at the same position (also called a mismatch);
5. $e$, the cost of having a gap (`-`) at a position (an insertion or deletion).

The matrix $S = s_{i,j}$ is filled as follows:

- The borders are filed with: $s_{0,j} = e \times i$ (first line) and $s_{i,0} = e \times i$ (first column);
- The interior is filled using the following formula:

$$s_{i,j} = max \begin{cases} m + s_{i-1,j-1} & \text{if } x_{i-1} = y_{j-1} \\ n + s_{i-1,j-1} & \text{if } x_{i-1} \neq y_{j-1} \\ e + s_{i-1,j} \\ e + s_{i,j-1} \end{cases}$$

To fill $S$, you first have to fill the borders and then the interior. To fill the interior you will need for each cell the cells directly to the left, above and on the top-left diagonal. You can thus fill the matrix either line by line or column by column. The similarity score is given by the value at the bottom-right corner: $s_{|x|,|y|}$.

Write the function
`needleman_wunsch(x: str, y: str, m: float, n: float, e: float) -> float` that returns the similarity score between `x` and `y`, given the score parameters `m`, `n`, `e`.

**Note:** To answer this question, you first have to create a numpy matrix, then fill it and finally return the value contained in the bottom-right cell.

**Example**

```
>>> needleman_wunsch("AAAA", "AA", 1, -0.5, -0.5)
1
>>> needleman_wunsch("GCACA", "GGCACA", 1, -0.5, -0.5)
4.5
>>> needleman_wunsch("GCGCGC", "GC", 1, -0.5, -0.5)
0
```