# DNA motifs

The term *DNA sequencing* refers to the techniques that are used to determine the order of nucleobases adenine (A), cytosine (C), guanine (G) and thymine (T) for a DNA molecule. The standard method to represent these bases is by using the first letter of their names, ACGT, which allows DNA to be represented as a string that consists of only these four letters. DNA strings are usually some millions of characters long.

*Substring matching* is the process of determining if a shorter string (a substring) occurs within a longer string. Substring matching has an important role in reconstructing an unknown DNA string of various smaller parts (assemblage), and searching for interesting substrings (motifs) within a known DNA string.

Python provides a string method find(substring[, begin][, end]) that prints the smallest index (integer) in which the $substring$ can be found within the interval $begin \leq index < end$. The parameters $begin$ and $end$ are optional, and the function prints the value -1 if the substring can't be found withing the given string. However, genome researchers generally want to find *all* locations in which a substring can be found in a given DNA string. Not only the position of the first appearance.

## Assignment

Write a function

motifs(sequence, subsequence[, begin][, end])

where $begin$ and $end$ are optional arguments with standard values that respectively equal the begin and the end of a string *sequence*. The values $begin$ and $end$ must be interpreted in the same way as when slicing strings or lists: \[0 = begin \leq index < end = \text{len(string)}\] The function must print a list of positions in which the *subsequence* occurs within the given *sequence*. As positions in the list, the indices of the first letters of the subsequence of the sequence are recorded, to which applies that $begin\leq index < end$.

## Example

```
>>> motifs('AAA', 'A')
[0, 1, 2]
>>> motifs('AAA', 'A', begin=1)
[1, 2]
>>> motifs('AAA', 'A', end=2)
[0, 1]
>>> motifs('AAA', 'A', begin=1, end=2)
[1]
>>> motifs('AAA', 'AA')
[0, 1]
>>> motifs('AAA', 'C')
[]
>>> motifs('AGGAATGCTCGTAGGATACTGAATGCTCGGACGTACGCT', 'GGA')
[1, 13, 28]
```

De term *DNA sequenering* verwijst naar technieken die gebruikt worden om de volgorde van de

nucleobasen adenine (A), cytosine (C), guanine (G) en thymine (T) te bepalen voor een DNA molecule. De standaardmanier om deze basen voor te stellen is door de eerste letter van hun naam te gebruiken, ACGT, waardoor DNA kan voorgesteld worden als een string die enkel uit deze vier letters bestaat. DNA strings zijn doorgaans enkele miljoenen basen (karakters) lang.

*Substring matching* is het proces waarbij bepaald wordt of een kortere string (de substring) voorkomt binnen een langere string. Substring matching speelt een zeer belangrijke rol bij het reconstrueren van een onbekende DNA string uit verschillende kleinere stukken (assemblage), en bij het zoeken naar interessante substrings (motieven) binnen een gekende DNA string.

Python voorziet standaard in een string methode find(substring[, start][, end]) die de kleinste index (integer) teruggeeft waarop de $substring$ gevonden wordt binnen het interval $start \leq index < end$. De parameters $start$ en $end$ zijn optioneel, en de functie geeft de waarde -1 terug indien de substring niet wordt teruggevonden in de gegeven string. Genoomonderzoekers willen doorgaans echter *alle* locaties vinden waarop een substring kan teruggevonden worden in een gegeven DNA string. Niet enkel de positie van het eerste voorkomen.

## Opgave

Schrijf een functie

motieven(sequentie, subsequentie[, begin][, einde])

waarbij $begin$ en $einde$ optionele argumenten zijn met standaardwaarden die respectievelijk gelijk zijn aan het begin en het einde van de string *sequentie*. De waarden $begin$ en $einde$ moeten hierbij op dezelfde manier geïnterpreteerd worden als bij het slicen van strings of lijsten: \[0 = begin \leq index < einde = \text{len(string)}\] De functie moet een lijst van posities teruggeven waarop de *subsequentie* voorkomt binnen de gegeven *sequentie*. Als posities in de lijst worden de indices opgenomen van de eerste letter van de subsequentie binnen de sequentie, waarvoor geldt dat $begin \leq index < einde$.

## Voorbeeld

```
>>> motieven('AAA', 'A')
[0, 1, 2]
>>> motieven('AAA', 'A', begin=1)
[1, 2]
>>> motieven('AAA', 'A', einde=2)
[0, 1]
>>> motieven('AAA', 'A', begin=1, einde=2)
[1]
>>> motieven('AAA', 'AA')
[0, 1]
>>> motieven('AAA', 'C')
[]
>>> motieven('AGGAATGCTCGTAGGATACTGAATGCTCGGACGTACGCT', 'GGA')
[1, 13, 28]
```