# First Sequences: Strings

## Traverse a string

### Algo

```
procedure print_string(string  s)
    variables
        integer i
begin
    i ← 1
    n = length(s)
    while i <= n do
        write(s[i], '\n')
        i ← i + 1
    end while
end
```

```
procedure print_string(string  s)
    variables
        integer i
begin
    for i ← 1 to length(s) do
        write(s[i], '\n')
    end for
end
```

---

## 1 Classics

**Exercise 1.1 (Search)**

1. Write a function that searches for a character in a string. It returns the position of the character if found, -1 otherwise.
2. Write a function that tests whether the string $S_1$ is a substring of the string $S_2$. If it is the case, it returns the position of $S_1$ first character in $S_2$, -1 otherwise.

---

**Exercise 1.2 (Palindrome)**

Write a function that tests whether a string is a palindrome.

*Some palindromes:*

- Engage le jeu que je le gagne !
- Never odd or even.
- Nice hat, Bob Tahecin.
- God! A red nugget! A fat egg under a dog!

Three levels:

**level 0:** The string contains only non accented lower letters (no spaces).
Ex: "engagelejeuquejelegagne".

**level 1:** The string contains non accented lower letters and spaces. First and final characters are not spaces. There is no double space.
Ex: "nice hat bob tahecin".

**level +:** The string contains any kind of character: accented, upper, ponctuation...
Ex: "Tu l'as trop écrasé César, ce port salut."

# 2 Some Archi and ...

### Exercise 2.1 (Conversions)

1. Write a function that converts an integer $n$ in his equivalent in $p$-bit two's complement representation (in a string).

   *Examples of output:*

```
>>> integer_to_twoscomp(-42, 8)
'11010110'
>>> integer_to_twoscomp(42, 8)
'00101010'
```

2. Write the function that computes the inverse conversion:

```
>>> twoscomp_to_integer("11010110", 8)
-42
>>> twoscomp_to_integer("00101010", 8)
42
```

---

### Exercise 2.2 (Frequency)

1. Write a function that returns the most frequent character in a string as well as its number of occurrences.

2. The following functions are given:

```
>>> help(ord)
    ord(c) -> integer
    Return the integer ordinal of a one-character string.

>>> ord('A')
65

>>> help(chr)
    chr(i) -> Unicode character
    Return a Unicode string of one character with ordinal i...

>>> chr(65)
'A'
```

   Actually, the string contains only "classic" characters (with codes from 0 to 255).

   Write a more efficient version of the previous question function.

3. Write a function that calculates the number of different characters occurring in a string.