- La **complexité** d'un algorithme est le nombre d'opérations élémentaires qu'il réalise, exprimé en fonction de la taille de l'entrée. Exemples d'opérations élémentaires :
  - opérations sur les nombres : +, - *,
  - comparaisons de nombres : ==, <=, <, !=
  - sur les listes : `L.append(e)`, `L[i]`...

```
def somme(n):
    s = 0
    for k in range(n):
        s += k
    return s
```

Complexité : n additions.

```
def somme(n):
    return n*(n - 1)//2
```

Complexité : 3 opérations élémentaires (1 addtion, 1 multiplication, 1 division)

- On note $f(n) = \mathrm{O}(g(n))$ si $\exists A,\ f(n) \leq Ag(n)$, pour n assez grand.
  Intuitivement : « $\mathrm{O}(f(n))$ » signifie « au plus une constante fois $f(n)$ ».
  En pratique : pour mettre une complexité sous la forme $\mathrm{O}(...)$, **on conserve seulement le terme dominant (le plus grand), sans la constante**.
  Exemples :

– $18n^3 - n + 20 = \mathrm{O}(n^3)$

– $n\ln(n) + 3n^2 = \mathrm{O}(n^2)$

– $2^n + 25n^3 = \mathrm{O}(2^n)$

```
for i in range(n):
    print("Hello World")
```

Complexité : n.

```
for i in range(n):
    for j in range(p):
        print("Hello World")
```

Complexité : np.

```
for i in range(n):
    for j in range(i):
        print("Hello World")
```

Complexité : $0+1+2+3+...+n-1 = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} = \mathrm{O}(n^2)$.

```
for i in range(n):
    print("Hello World")
for i in range(n):
    print("Hello World")
```

Complexité : $2n = \mathrm{O}(n)$.

- Exercice : Écrire un algorithme pour calculer le nombre de diviseurs d'un entier n.
  Une première solution en $\mathrm{O}(n)$:

```
nb_div = 0
for d in range(2, n + 1):
    if n % d == 0:
        nb_div += 1
```

Une meilleure solution en $\mathrm{O}(\sqrt{n})$, en utilisant le fait que si d divise n alors $\frac{n}{d}$ divise aussi n (on peut donc compter deux fois les diviseurs jusqu'à $\sqrt{n}$) :

```
nb_div = 0
for d in range(2, int(n**.5)):
    if n % d == 0:
        nb_div += 2
if d*d == n:
    nb_div += 1
```