Vocabulaire.

→ Un ***pseudo-code*** est la description (en français) d'un algorithme.

→ Une ***implémentation*** est un programme écrit en Python permettant d'exécuter un algorithme.

Exercice 1. Exponentiation rapide

Le but de l'exercice est de calculer x^n où $n \in \mathbb{N}$ et $x \in \mathbb{R}$ à l'aide d'un algorithme dichotomique. Dans tout l'exercice, il est interdit d'utiliser l'opérateur `**`.

1. Commençons par implémenter l'algorithme naïf de mise à la puissance.
   (a) Donner un pseudo-code permettant d'obtenir x^n à l'aide de n tours de boucle.
   (b) Écrire une fonction `puiss` qui implémente cet algorithme.
   (c) Quelle est la complexité de votre fonction ?

Pour calculer x^n à l'aide d'une dichotomie, l'idée est de diviser la valeur de n par 2 à chaque tour de boucle. Pour cela, nous utiliserons la relation :

$$\begin{cases} x^n = 1 & \text{si } n = 0 \\ x^n = (x^2)^{n/2} & \text{si } n > 0 \text{ est pair} \\ x^n = x \times (x^2)^{(n-1)/2} & \text{si } n > 0 \text{ est impair} \end{cases} \tag{1}$$

Par exemple, pour calculer 2^{21} :

$$\begin{aligned} 2^{21} &= 1 \times (2)^{21} \\ &= 2 \times (4)^{10} \\ &= 2 \times (16)^{5} \\ &= 32 \times (256)^{2} \\ &= 32 \times (65\,536)^{1} \\ &= 2\,097\,152 \times (4\,294\,967\,296)^{0} \end{aligned}$$

p	y	m
1	2	21
2	4	10
2	16	5
32	256	2
32	65 536	1
2 097 152	4 294 967 296	0

Dans les calculs qui précèdent, chaque égalité (sauf la première) s'obtient grâce à l'une des relations de l'équation (1). Les étapes du calcul sont caractérisées par trois variables `p`, `y` et `m` comme présenté dans le tableau. Cet algorithme peut être qualifié de "dichotomique" car la valeur de `m` est divisée par 2 (au moins) à chaque itération.

2. (a) Donner un pseudo-code permettant de calculer x^n à l'aide d'une exponentiation rapide. Vous devez bien sûr utiliser le principe décrit ci-dessus en manipulant trois variables `p`, `y` et `m`.
   (b) Écrire une fonction `puiss_dicho` qui implémente cet algorithme.
   (c) (facultatif) Quelle est la complexité de votre fonction ?
3. (a) Vérifier à l'aide de tests que la fonction `puiss_dicho` est plus rapide que la fonction `puiss`. On pourra utiliser la fonction `perf_counter` du module `time` pour mesurer les temps d'exécution.
   (b) (facultatif) Expliquer comment vérifier numériquement que les complexités trouvées aux questions 1c et 2c sont correctes.

Exercice 2. Concaténations de chaînes de caractères

Dans cet exercice, on souhaite calculer la chaîne de caractères obtenue en concaténant toutes les chaînes de caractères d'une liste. Par exemple, à partir de `L1` ou de `L2`, on obtient `"Le cheval c'est genial"` :

```
L1 = ["L","e"," ","c","h","e","v","a","l"," ","c","'","e","s","t"," ","g","e","n","i","a","l"]
L2 = ["Le ","ch","","eva","l"," ","c'est gen","ial",""]
```

Soit `concat` la fonction suivante :

```
def concat(L):
    """concat(L: list[str]) -> str"""
    C = [""]
    for s in L:
        C.append(C[-1] + s)
    return C[-1]
```

1. (a) (facultatif) On numérote les tours de boucle de `0` à `len(L)-1`. Au début du tour de boucle numéro `k`, déterminer la valeur de `C[-1]` en fonction de `L`.
   (b) (facultatif) Dans le cas où toutes les chaînes de caractères de `L` sont de taille 1, montrer que la fonction `concat` s'exécute en temps quadratique en `len(L)`. **Rappel :** étant données deux chaînes de caractères `s1` et `s2`, la concaténation `s1 + s2` se calcule en temps $\mathcal{O}\Big(\texttt{len(s1)} + \texttt{len(s2)}\Big)$.

Pour obtenir un algorithme dichotomique, l'idée est de diviser la taille de la liste par 2 (environ) à chaque étape. Pour cela, on concatène les chaînes de caractères deux par deux jusqu'à ce que la liste soit de taille 1. Par exemple, voici l'évolution de la liste `L1` :

```
["L","e"," ","c","h","e","v","a","l"," ","c","'","e","s","t"," ","g","e","n","i","a","l"]
["Le"," c","he","va","l ","c'","es","t ","ge","ni","al"]
["Le c","heva","l c'","est ","geni","al"]
["Le cheva","l c'est ","genial"]
["Le cheval c'est ","genial"]
["Le cheval c'est genial"]
```

2. (a) Donner le pseudo-code de cet algorithme dichotomique.
   (b) Écrire la fonction « `concat_dicho(L: list[str]) -> str` » correspondante.
3. (facultatif) Supposons que toutes les chaînes de caractères de `L` soient de taille 1 et que `len(L)` soit de la forme 2^k avec $k \in \mathbb{N}$. Dans les questions qui suivent, "un tour de boucle" consiste à diviser la taille de la liste par 2.
   (a) (facultatif) Déterminer le nombre de tours de boucle dans la fonction `concat_dicho`. Justifier.
   (b) (facultatif) Montrer que le temps d'exécution d'un tour de boucle lors d'un appel à `concat_dicho(L)` est linéaire en `len(L)`. Justifier.
   (c) (facultatif) En déduire le temps d'exécution de la fonction `concat_dicho`.
4. Vérifier à l'aide de tests que la fonction `concat_dicho` est plus rapide que la fonction `concat`.

Exercice 3. Nombre d'occurrences dans une liste triée

1. Écrire une fonction qui prend en entrée une liste triée `L` ainsi qu'un entier `x` et renvoie le nombre d'occurrences de `x` dans `L`. Votre fonction devra être de complexité logarithmique en `len(L)`.
2. Écrire une fonction qui prend en entrée une liste triée `L` ainsi qu'un indice `i` et renvoie le nombre d'occurrences de `L[i]` dans `L`. Votre fonction devra être de complexité logarithmique en `m` où `m` est la valeur renvoyée.

Exercice 4. Trichotomie (épreuve 2019 du concours e3A)

Le but de la recherche par trichotomie est le même que celui de la recherche par dichotomie : étant donnés un entier `x` et une liste triée `L`, il s'agit de déterminer si `x` appartient à `L`. Dans la recherche par dichotomie, la taille de la sous-liste dans laquelle on recherche `x` est divisée par 2 (ou plus) à chaque étape. Une recherche par trichotomie suit le même principe, mais la taille de la sous-liste est divisée par 3 (ou plus) à chaque étape.

1. Écrire une fonction « `appartient_tricho(L: list[int], x: int) -> bool` » qui indique si `x` appartient à `L`.

Exercice 5. Méthode `join` pour les chaînes de caractères

Écrire une fonction de signature « `join(sep: str, L: list[str]) -> str` » qui renvoie la concaténation des éléments de `L` séparés par `sep`. Par exemple :

`join(" ", ["J'aime","l'informatique"])`	vaut	`"J'aime l'informatique"`
`join("--", ["toto","titi","tata"])`	vaut	`"toto--titi--tata"`
`join("", ["Mots", "Sans", "Espace"])`	vaut	`"MotsSansEspace"`
`join("...", ["a","b","c","d","e","f"])`	vaut	`"a...b...c...d...e...f"`
`join("--", [])`	vaut	`""`

Remarque. En réalité, `join` existe déjà en Python : il suffit d'écrire `sep.join(L)`. Bien sûr vous n'avez pas le droit d'utiliser cette méthode dans cet exercice.

Remarque. Si on définit « `s = join(" ", ["J'aime","l'informatique"])` », alors :

→ `s[0]` vaut `'J'` et pas `'"'`.

→ `s[-1]` vaut `'e'` et pas `'"'`.

→ `len(s)` vaut 21 et pas 23.

Exercice 6. Méthode `split` pour les chaînes de caractères

Écrire une fonction de signature « `split(s: str) -> list[str]` » qui renvoie la liste constituée des mots de `s`. Par exemple :

`split("Ceci est un test")`	vaut	`["Ceci", "est", "un", "test"]`
`split("Avec six espaces")`	vaut	`["Avec", "six", "espaces"]`
`split("UnSeulMot")`	vaut	`["UnSeulMot"]`
`split("  test  ")`	vaut	`["test"]`
`split("")`	vaut	`[]`

Si besoin, on pourra lire les indications ci-dessous.

Remarque. En réalité, `split` existe déjà en Python : il suffit d'écrire `s.split()`. Bien sûr vous n'avez pas le droit d'utiliser cette méthode dans cet exercice.

Indications (essayez de résoudre l'exercice sans lire ce qui suit).

1. À l'aide d'une boucle `while`, écrire une fonction de signature « `debut_mot(s: str, i: int) -> int` » qui renvoie le plus petit indice `j` $\geqslant$ `i` tel que `s[j]` n'est pas un espace. Si pour tout `j` $\geqslant$ `i`, le caractère `s[j]` est un espace, votre fonction renverra `len(s)`. Par exemple, avec `s = " a bc d "`, on obtient :

`i`	0	1	2	3	4	5	6	7	8
`debut_mot(s,i)`	1	1	3	3	4	6	6	8	8

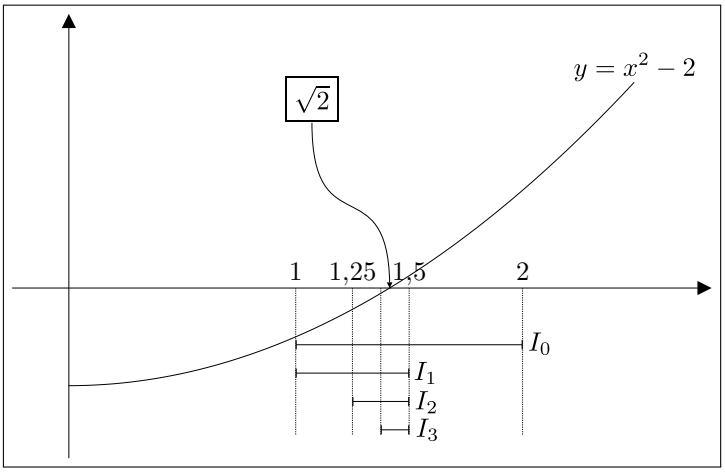
2. À l'aide d'une boucle `while`, écrire une fonction de signature « `fin_mot(s: str, i: int) -> int` » qui renvoie le plus petit indice `j` $\geqslant$ `i` tel que `s[j]` est un espace. Si pour tout `j` $\geqslant$ `i`, le caractère `s[j]` n'est pas un espace, votre fonction renverra `len(s)`. Par exemple, avec `s = " a bc d "`, on obtient :

`i`	0	1	2	3	4	5	6	7	8
`fin_mot(s,i)`	0	2	2	5	5	5	7	7	8

3. À l'aide des fonctions précédentes, écrire la fonction `split`.

Exercice 7. Approximation de $\sqrt{2}$ par dichotomie

Soit $\varepsilon > 0$ un réel. Notre but est de donner une approximation de $\sqrt{2}$ à ε-près, c'est à dire de calculer un nombre x tel que $|x - \sqrt{2}| \leqslant \varepsilon$. La difficulté est que les seules opérations autorisées sont l'addition, la soustraction, la multiplication et la division (en particulier, vous ne pouvez pas utiliser l'opérateur `**0.5` de Python ou la fonction `sqrt` du module `math`). Pour cela, on va construire des intervalles $I_0, I_1, I_2, \ldots$ tels que pour tout $k \in \mathbb{N}$, on ait $\sqrt{2} \in I_k$ et I_{k+1} soit au moins deux fois plus petit que I_k.



Soit f la fonction $x \mapsto x^2 - 2$. Les intervalles I_k sont définis par récurrence :

→ On pose $I_0 = [\,1\,;2\,]$.

→ Soit $k \in \mathbb{N}$. Supposons que l'intervalle I_k ait été construit et construisons l'intervalle I_{k+1}. Pour cela, on note a_k et b_k les bornes de I_k, c'est à dire que $I_k = [\,a_k\,;b_k\,]$. Comme $\sqrt{2} \in I_k$, on a :

$$f(a_k) \leqslant 0 \qquad \text{et} \qquad f(b_k) \geqslant 0.$$

On pose $m_k = \dfrac{a_k + b_k}{2}$, alors :

- Si $f(m_k) = 0$, alors I_{k+1} est défini par $I_{k+1} = [\,m_k\,;m_k\,]$.
- Si $f(m_k) > 0$, comme $f(a_k) \leqslant 0$ et que la fonction f est continue, alors elle s'annule sur l'intervalle $[\,a_k\,;m_k\,]$. On pose donc $I_{k+1} = [\,a_k\,;m_k\,]$ ce qui garantit que $\sqrt{2} \in I_{k+1}$.
- Si $f(m_k) < 0$, comme $f(b_k) \geqslant 0$ et que la fonction f est continue, alors elle s'annule sur l'intervalle $[\,m_k\,;b_k\,]$. On pose donc $I_{k+1} = [\,m_k\,;b_k\,]$ ce qui garantit que $\sqrt{2} \in I_{k+1}$.

Soit $k \in \mathbb{N}$ le plus petit entier tel que l'intervalle $I_k = [\,a_k\,;b_k\,]$ soit de taille inférieure ou égale à 2ε ; alors le réel $x_0 = \dfrac{a_k + b_k}{2}$ est une approximation de $\sqrt{2}$ à ε-près.

1. Écrire une fonction `approx_sqrt2` qui prend en entrée un flottant `epsilon` et renvoie le réel x_0 défini ci-dessus.
2. **Facultatif.** Donner en le justifiant le nombre exact de tours de boucle en fonction de ε. On pourra supposer que la condition $f(m_k) = 0$ n'est jamais vérifiée. On attend une réponse sous la forme d'une formule mathématiques, pas d'un programme.