TP 1

# Révisions : Objets et programmation



## Compétences attendues.

- ✓ Connaître et savoir manipuler les différents objets de base du langage `Python` (les entiers, les nombres flottants, les booléens, les chaînes de caractères, les variables).
- ✓ Savoir programmer en `Python` : fonctions d'entrée et de sortie, boucles `if`, `for`, `while`.
- ✓ Savoir définir une fonction en `Python`.

## Liste des commandes `Python` exigibles aux concours.

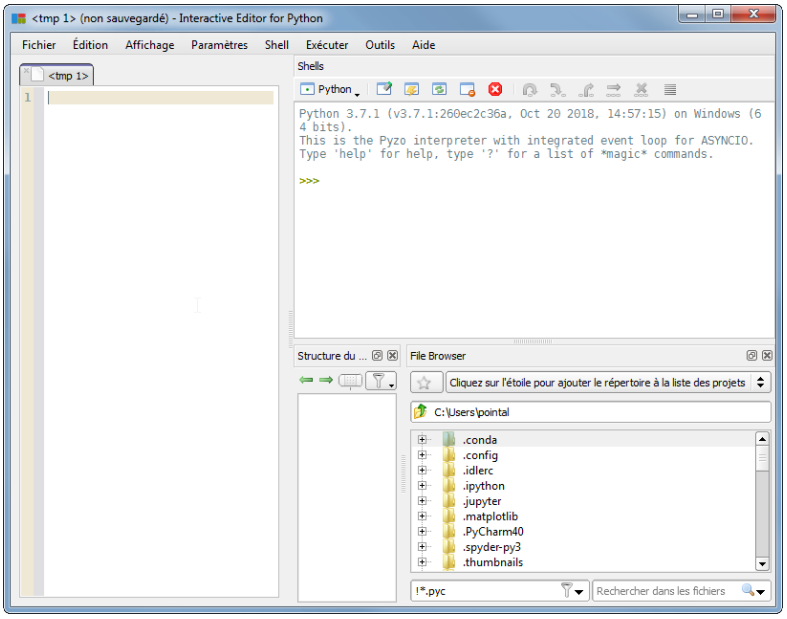
- Opérations sur les entiers et les flottants : `+`, `-`, `*`, `/`, `**`
- Comparaison et connecteurs logiques pour les booléens : `==`, `>`, `<`, `>=`, `<=`, `!=`, `True`, `False`, `and`, `or`, `not`
- Programmation : `=`, `input`, `print`, `if`, `elif`, `else`, `range`, `for`, `while`, `def`, `return`


Mathieu Mansuy
Professeur en ECG deuxième année spécialité mathématiques approfondies au Lycée Louis Pergaud (Besançon)
Page personnelle : `mathieu-mansuy.fr/`
E-mail : `mansuy.mathieu@hotmail.fr`

# En préambule

`Python` est un langage de programmation conçu par le néerlandais Guido van Rossum. Sa première version public date de 1991. Son nom est un clin d'oeil à la série télévisée *Monty Python's Flying Circus* dont ce développeur est fan. Il est libre et gratuit, à télécharger ici. Nous n'utiliserons pas directement `Python`. Nous passerons par un *environnement de développement intégré* (en anglais *IDE* pour *interactive development environnement*). Il en existe plusieurs : `EduPython`, `Spider`... Nous utiliserons pour notre part l'IDE `Pyzo`. Vous devez l'installer sur vos ordinateurs personnels afin de pouvoir l'utiliser pour préparer ou réviser un TP ou un TD de maths.

Au lancement de `Python` (avec l'IDE `Pyzo`), une fenêtre contenant trois sous-fenêtres s'ouvre :




- En haut à droite : la **console** (ou **interpréteur** ou **shell** en anglais) :

  C'est « la feuille de travail » qui permet de taper après le symbole `>>>` des commandes ou des instructions et d'analyser les résultats. Il est possible d'entrer plusieurs instructions en même temps en les séparant par un point-virgule `;`. L'appui sur la touche `Entrée` provoque l'exécution des instructions.

- À gauche : l'**éditeur de texte** :

  C'est dans cette fenêtre qu'on rédigera des programmes ou des fonctions qui pourront être enregistrés, puis exécutés dans la console. Sauf dans le cas de commandes ou d'instructions très simples, on utilisera systématiquement l'éditeur de texte.

En cas de besoin, vous disposez de l'aide `Python` qui indique la syntaxe précise de chaque instruction, accompagnée de nombreux exemples. Elle est accessible en se rendant dans la rubrique « Aide » depuis la barre de menu.

# 1 Objets et commandes de base

## 1.1 Calcul numérique simple

`Python` peut s'utiliser comme une calculatrice : on saisit une instruction directement dans la console (après le symbole `>>>`) puis on l'exécute en appuyant sur la touche `Entrée`.

**Définition.**

`Python` distingue les nombres entiers, de type `int`, des nombres à virgules, aussi appelés nombres flottants et de type `float`. Les opérations sur ces nombres sont :

`+`, `-`, `*`, `/` et `**` (pour la puissance).

**Remarque.** Attention donc, `2` et `2.0` ne sont pas les mêmes objets en `Python`, le premier est de type entier `int` alors que le second est de type flottant `float`.

**Exemple.** Taper dans la console les instructions suivantes :

```
>>> (2+3)*4-5; (5.0-3.0)/2.0;
>>> ((2.0+3)**2)/5.0
>>> 4/2; (-1)**2/(-1)**(-3)
```

**Remarques.**

- Attention à parenthèser correctement lorsqu'on entre des instructions sur Python.
- Le résultat d'opérations sur les entiers n'est pas nécessairement un entier : par exemple, une division d'entiers donne systématiquement un résultat de type `float`. Il est possible de passer du type `int` au type `float` et inversement. Entrer par exemple dans la console les instructions suivantes :

  ```
  >>> float(3); float(-10); int(2.6); int(-3.7)
  >>> int(4/2)
  ```

- Qu'en est-il si lors d'une opération, l'un des arguments est un entier et l'autre un flottant ?

  ```
  >>> type(3+5.6)
  ```

  On constate donc que dans tous les cas, le résultat est de type `float`.
- Il n'est pas possible de modifier les instructions déjà entrées dans la console. Par contre, on peut rappeler d'anciennes commandes en utilisant les flèches du clavier ↑ et ↓.

## 1.2 Les booléens

Une expression booléenne peut prendre la valeur `True` ou `False` (de type `bool` sur `Python`).

**Définition.**

- **Comparaisons - tests :** `==` (test d'égalité), `<`, `>`, `<=`, `>=`, `!=` (différent de).
- **Connecteurs logiques :** `and` (et), `or` (ou), `not` (négation).

**Exemple.** Taper dans la console les instructions suivantes :

```
>>> 2 == 3; 1.4 < 2; 3 >= 1; 1.2 != 2.3
>>> 2 < 3 <= 1
>>> (1<2) and (2**2==4)
>>> (-1>=0) or (2**2==4)
>>> not (2<3)
```

**Remarques.**

1. Attention : pour tester l'égalité, il faut bien taper `==` et non `=` qui est une affectation (et qu'on verra dans la suite) !
2. Le `or` est un ou inclusif : par exemple, l'instruction `x==y or y==z` renvoie `True` si on a soit $x = y$, soit $y = z$, soit les 2.

## 1.3 Les chaînes de caractères

Pour définir une chaîne de caractères (de type `str` sur `Python`), on utilisera des guillemets simples `'`.

**Exemple.** Taper dans la console l'instruction suivante :

```
>>> 'Mon message'
>>> 'Maths'+'Info'
```

### 1.4 Les variables

**Définition.**

- Une **variable** est un emplacement de mémoire dans lequel on peut stocker un objet `Python`, que ce soit une fonction, une expression ou une valeur particulière.
- On nomme les variables par une chaîne de caractères. À noter que `Python` distingue les minuscules des majuscules (par exemple, `a` et `A` désignent deux variables distinctes).
- Pour affecter une valeur à une variable, on utilise le symbole `=`. La valeur placée à droite est affectée à la variable dont le nom est écrit à gauche.
- Le type d'une variable est la nature de son contenu. Nous rencontrerons essentiellement les types suivants : `int` (les entiers), `float` (les nombres flottants), `bool` (les booléens), `str` (les chaînes de caractères), `array` (les tableaux).

**Exemple.** Taper dans la console les instructions suivantes :

```
>>> a = 2; b = 3
>>> a**2-b; a+1 == b
```

Il faudra bien avoir en tête que ces affectations sont **globales** et qu'elles seront utilisées par `Python` dans toute la feuille de travail. On peut libérer la place mémoire occupée par une variable à l'aide de la commande `del`.

**Exemple.** Taper dans la console les instructions suivantes :

```
>>> a
>>> del a
>>> a
```

## 2 Programmer en Python

### 2.1 Les scripts

Taper directement dans la console ne permet pas de faire facilement des modifications lorsque plusieurs lignes d'instructions ont été tapées. Si le nombre d'instructions est important, on utilise l'éditeur de texte :

- On tape les instructions dans la fenêtre de l'éditeur de texte (appelé le **script**) ;
- On l'exécute ensuite dans la console à partir du menu **Exécuter**.

**Exemple.** Taper dans l'éditeur de texte les commandes suivantes, les exécuter et constater le résultat dans la console.

```
1 a = 1; b = -6; c = 5;
2 b**2-4*a*c
```

**Conseils de mise en forme des scripts.**
L'éditeur reconnait le langage `Python` : il met en couleur les mots-clefs, un décalage de début de ligne appelé indentation se fait automatiquement lorsqu'on commence une boucle ou un test, il ferme automatiquement les parenthèses ouvertes... Pour que les scripts soient facilement compréhensibles, on suivra les conseils suivantes :

- Écrire une seule instruction par ligne, et sauter des lignes entre les différentes étapes du script.
- Il est absolument indispensable de respecter l'indentation des boucles, des structures conditionnelles... surtout lorsqu'elles s'imbriquent les unes dans les autres. C'est grâce à cela que `Python` peut repérer le début et la fin d'une boucle ou d'un programme.
- Commenter son script (rôle de chaque variable, fonctionnement du programme, . . . ) pour s'y retrouver plus rapidement lorsqu'on reprendra le programme. Les lignes de commentaires commencent par le symbole #. Tout ce qui suit # ne sera pas exécuté.
- Le double symbole ## permet de scinder le script à l'aide de délimiteurs horizontaux. On peut alors exécuter uniquement la partie entre deux délimiteurs successifs à l'aide du menu **Exécuter** ou en faisant `Ctrl + Entrée`. C'est particulièrement utile en TP pour passer d'un exercice à l'autre...

**Pratique.** En sélectionnant une partie d'un programme et en faisant `Ctrl + R` ou **Édition -> Commenter**, toute la zone sélectionnée se retrouve précédée d'un #. Elle ne sera donc pas exécutée. C'est très pratique pour repérer une erreur dans un programme : on exécute le programme partie par partie. On constate ainsi ce qui fonctionne ou pas, et on détermine de cette manière où est l'erreur. Pour activer de nouveau une zone mise en commentaire, on la sélectionne et on effectue `Ctrl + T` ou **Édition -> Décommenter**.

## 2.2 Les fonctions d'entrée et de sortie

**Définition.**

- **Instructions de saisie :**

  L'instruction `x = input('Mon message')` écrit le message `Mon message` sur l'écran puis donne la main à l'utilisateur pour qu'il rentre la valeur à affecter à la variable `x`.

- **Instructions d'affichage :**

  L'instruction `print('Mon message')` écrit le message `Mon message` sur l'écran.

  L'instruction `print(x)` affiche le contenu de la variable `x`.

**Remarques.**

1. L'instruction `print(x,y,z)` provoque l'affichage des contenus des variables `x`, `y`, `z` dans cet ordre sur une même ligne. Il est aussi possible de combiner affichage de messages et affichage de contenus de variables.
2. Attention : en faisant `x = input('Mon message')`, la valeur entrée par l'utilisateur et affectée à la variable `x` est systématiquement de type chaîne de caractères `str`. Si on veut un entier ou un nombre flottant, il faudra utiliser les commandes `int` et `float`.

**Exemple.** Taper dans l'éditeur de texte les instructions suivantes puis exécuter :

```
1 n = int(input('Entrer n :'))
2 print('Le nombre qui suit ',n,' est ',n+1)
```

## 2.3 Instructions conditionnelles

**Définition.**

Pour effectuer un bloc d'instructions sous certaines conditions, on utilise une boucle `if`. Voici la syntaxe :

`if` $condition_1$ :
    $instructions_1$
`elif` $condition_2$ :
    $instructions_2$
...
`elif` $condition_{k-1}$ :
    $instructions_{k-1}$
`else` :
    $instructions_k$

Si la $condition_1$ est vraie, le premier bloc d'instructions ($instructions_1$) est effectué. Si la $condition_1$ est fausse, on fait un deuxième test. Si la $condition_2$ est vraie, le deuxième bloc d'instructions ($instructions_2$) est effectué. Ce procédé est répété jusqu'à la $condition_{k-1}$. Si la $condition_{k-1}$ est vraie, le bloc d'instructions ($instructions_{k-1}$) est effectué. Si la $condition_{k-1}$ est fausse, le dernier bloc d'instructions ($instructions_k$) est effectué.

✎ **Syntaxe.**

- Il faut penser aux deux points après chaque *condition* et après le `else`.
- Bien respecter l'indentation : c'est elle qui permet à Python de reconnaître le bloc d'instructions à exécuter et où se termine la structure conditionnelle. Une fois les instructions écrites, si l'on doit poursuivre le script, on revient au même niveau d'indentation que le `if`.

**Exemple.** Taper dans l'éditeur de texte le script suivant puis l'exécuter. Que calcule-t-il ?

```
a = float(input(' Entrer un reel a : '))
if a >= 0 :
    print(a)
else :
    print(-a)
```

## 2.4 Boucle for

**Définition.**

La commande `range` énumère des entiers en progression arithmétique.

- `range(n)` énumère les entiers successifs de 0 à $n-1$.
- `range(n,m)` énumère les entiers successifs de $n$ à $m-1$ (avec $n < m$).
- `range(n,m,r)` énumère les entiers en progression arithmétique de raison $r$ (entier positif ou négatif, appelé le pas), de $n$ inclu à $m$ exclu.

**Remarques.**

- La commande `range` ne provoque pas d'affichage.
- On prendra garde à deux choses : l'énumération s'arrête **avant** l'entier pris comme extrémité à droite, et par défaut, elle commence à 0 et non à 1. Par exemple :
  - `range(7)` énumère les entiers $0, 1, 2, 3, 4, 5, 6$.
  - `range(3,9)` énumère les entiers $3, 4, 5, 6, 7, 8$.
  - `range(-2,8,2)` énumère les entiers $-2, 0, 2, 4, 6$.
  - `range(4,0,-1)` énumère les entiers $4, 3, 2, 1$.

**Définition.**

Pour répéter un bloc d'instructions un nombre déterminé de fois, on utilise une boucle `for`. La syntaxe est la suivante :

`for` *variable* `in range(`*début*`,`*fin*`,`*pas*`) :`
  *instructions*

La *variable* parcourt dans l'ordre les entiers de *début* inclu à *fin* exclu en suivant le *pas* régulier et, à chaque étape, les *instructions* sont effectuées.

**Remarques.**

- Comme pour les instructions conditionnelles, il faut faire attention aux deux points après le `range` et à l'indentation !
- Pour le `range`, on utilisera lorsque c'est possible les expressions simplifiées `range(n,m)` ou `range(n)`.
- On peut remplacer l'énumérateur `range` par une liste, une chaîne de caractères ou un tableau. Par exemple, si la variable `L` contient une liste, la boucle

  `for` *variable* `in L :`
    *instructions*

  répète les instructions indentées, une fois pour chaque élément de la liste.

**Exemple.** Taper dans l'éditeur de texte le script suivant puis l'exécuter. Que calcule-t-il ?

```
p = 1
for k in range(1,101) :
    p = p*k
print(p)
```

## 2.5 Boucle while

**Définition.**

Pour répéter un bloc d'instructions tant qu'une condition est vérifiée, on utilise une boucle `while`. La syntaxe est la suivante :

```
while condition :
    instructions
```

Tant que la *condition* est vérifiée, les *instructions* sont effectuées. Dès que la *condition* n'est plus vérifiée, les *instructions* sont ignorées.

**Remarques.**

- Même remarque que pour les boucles `if` et `for` : attention aux deux points et à l'indentation !
- Attention également à la condition : elle doit se révéler fausse à un moment donné, sinon la boucle est sans fin...

**Exemple.** Taper dans l'éditeur de texte le script suivant puis l'exécuter. Que calcule-t-il ?

```
1 s = 0
2 i = 2
3 while i <=100 :
4     s = s+i
5     i = i+2
6 print(s)
```

## 2.6 Les fonctions

**Définition.**

- La commande

  ```
  def nom(x) :
      instructions
      return ...
  ```

  permet de créer une fonction, nommée ici `nom`, qui à chaque variable `x` associe la quantité définie après le `return`.
- Une fonction peut posséder plusieurs variables. L'en-tête est alors `def nom(x1,x2,...,xn) :`.

**Exemple.** Pour définir la fonction $f : x \mapsto x^2 + 2x + 1$, recopier le script suivant dans l'éditeur de texte :

```
1 def f(x) :
2     return x**2 + 2*x + 1
```

Après avoir exécuté, on peut alors utiliser cette fonction dans la console. Taper par exemple :

```
>>> f(1); f(0); f(-1)
```

**Remarques.**

- Toujours la même remarque : attention aux deux points et à l'indentation !
- Les fonctions seront systématiquement à définir dans l'éditeur de texte. Une fois qu'une fonction a été chargée (le script qui la contient a été sauvé et exécuté), elle est disponible jusqu'à la fin de la session. Elle peut donc être appelée par son nom dans la console ou dans une autre fonction. L'appel de la fonction dans la console provoque l'affichage de la valeur retournée.
- Les variables utilisées dans le corps d'une fonction sont des variables locales : elles n'existent pas à l'extérieur de la fonction.

**Mise en garde.**

1. Lorsqu'on définit une fonction, inutile d'ajouter les fonctions d'entrée et de sortie `input` et `print` à celle-ci. L'utilisateur est sensé connaitre l'argument à rentrer pour la fonction. `Python` lui, renverra le contenu des variables placées après le `return`.

   Dans l'exemple précédent, l'utilisateur rentre donc un entier ou un flottant à la fonction `f`, et `Python` renvoie le résultat du calcul donné après le `return` en sortie.

2. L'instruction `return` arrête le déroulement de la fonction. Si cette instruction est rencontrée, le programme s'arrête donc (en affichant le contenu de la variable à retourner) et le code situé après le `return` ne s'exécutera pas.

**Exemple.** Recopier la fonction suivante dans l'éditeur de texte :

```
def minimum(x,y) :
    if x<=y :
        return x
    else :
        return y
```

Enregistrer et exécuter cette fonction puis tester la sur quelques exemples.

# 3 Exercices

### Exercice 1 (★)

Écrire un programme qui demande trois réels $a$, $b$ et $c$ et qui affiche le nombre de racines réelles du polynôme du second degré $P : x \mapsto ax^2 + bx + c$. Selon les cas, votre programme devra afficher le message "$P$ n'admet pas de racine", "$P$ admet une racine" ou "$P$ admet deux racines".

---

### Exercice 2 (★)

1. Définir la fonction $f$ suivante sur `Python` : $f : x \mapsto \begin{cases} 1/x & \text{si } x \neq 0, \\ 0 & \text{sinon} \end{cases}$

2. On donne le script suivant :

```
def g(x) :
    if x < -1 :
        y = -2
    if x >= -1 and x<= 1 :
        y = 2*x
    if x > 1 :
        y = 2
    return y
```

   (a) Déterminer l'expression mathématique de la fonction $g$ ainsi définie.

   (b) Proposer un script analogue au précédent qui utilise `elif`.

---

### Exercice 3 (★)

On considère les sommes doubles suivantes :

$$S_n = \sum_{1\leq i,j\leq n} \frac{i}{2^j} \quad \text{et} \quad T_n = \sum_{1\leq i\leq j\leq n} \frac{i^3}{j(j+1)}.$$

1. (a) Écrire un programme qui, étant donné un entier $n \geq 1$, calcule et affiche $S_n$.

   (b) Même question pour $T_n$.

2. Calculer à la main $S_n$ et $T_n$ en fonction de $n$.

---

**Exercice 4 (★)**
Soit $(u_n)_{n\in\mathbb{N}}$ une suite définie par $u_0 = 1$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = u_n^2 + 1$.

1. Définir une fonction `suite` qui, à un entier naturel $n$ en entrée, renvoie la valeur de $u_n$.
2. (a) Montrer que la suite $(u_n)$ est croissante, puis que $\lim\limits_{n\to+\infty} u_n = +\infty$.
   (b) Définir une fonction `rang` qui, à un réel $a$, renvoie le rang du premier terme de la suite $(u_n)$ qui est supérieur ou égal à $a$.

---

**Exercice 5 (★)**
Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 = \dfrac{1}{2}$ et $\forall n \in \mathbb{N}$, $u_{n+1} = \dfrac{2u_n}{u_n + 1}$. On admet que $\lim\limits_{n\to+\infty} u_n = 1$.

1. Définir une fonction `valabs`, qui prend en entrée un flottant et qui renvoie sa valeur absolue.
2. À l'aide de la fonction `valabs`, écrire un programme permettant de déterminer le plus petit entier naturel $n$ pour lequel on a :
$$|u_n - 1| \leq 10^{-3}.$$

---

**Exercice 6 (★★)**
1. On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 1$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = 2nu_n + 3$.
   Écrire une fonction `suiteu` prenant en entrée un entier naturel $n$ et renvoyant la valeur de $u_n$.
2. Même question avec $(v_n)_{n\in\mathbb{N}}$ définie par $v_0 = 1$, $v_1 = -2$ et pour tout $n \in \mathbb{N}$, $v_{n+2} = 2v_{n+1} - v_n$.
3. On considère les suites $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$ définies par $a_0 = 1$, $b_0 = 2$ et les relations :
$$\forall n \in \mathbb{N},\ a_{n+1} = \frac{a_n^2}{a_n + b_n} \quad \text{et} \quad b_{n+1} = \frac{b_n^2}{a_n + b_n}.$$
   Écrire une fonction `suites` ayant en entrée un entier naturel $n$ et qui renvoie les réels $a_n$, $b_n$.

---

**Exercice 7 (★★)**
Soient $(u_n)_{n\in\mathbb{N}^*}$ et $(v_n)_{n\in\mathbb{N}^*}$ les suites définies par : $\forall n \in \mathbb{N}^*,\ u_n = \sum\limits_{k=1}^{n} \dfrac{1}{k^2}$ et $v_n = u_n + \dfrac{1}{n}$.

1. Montrer que les suites $(u_n)_{n\in\mathbb{N}^*}$ et $(v_n)_{n\in\mathbb{N}^*}$ sont adjacentes.
2. Écrire une fonction `approx` qui, à un réel strictement positif $\varepsilon$, associe une approximation de $\sum\limits_{k=1}^{+\infty} \dfrac{1}{k^2}$ à $\varepsilon$ près.

---

**Exercice 8 (★★ - Raisonnement par dichotomie)**
On considère une fonction $f$ continue et strictement monotone sur un intervalle $[a, b]$ (avec $a < b$) telle que $f(a)f(b) < 0$. Le théorème de la bijection assure que l'équation $f(x) = 0$ possède une unique solution $\alpha$ sur $]a, b[$.
On construit deux suites $(a_n)$ et $(b_n)$ définies de la manière suivante : $a_0 = a$, $b_0 = b$ et pour tout entier naturel $n$ :

- Si $f(a_n)f\left(\frac{a_n+b_n}{2}\right) \leq 0$, alors $\alpha \in \left]a_n, \frac{a_n+b_n}{2}\right[$. On pose $a_{n+1} = a_n$ et $b_{n+1} = \frac{a_n+b_n}{2}$.
- Si $f(a_n)f\left(\frac{a_n+b_n}{2}\right) > 0$, alors $\alpha \in \left]\frac{a_n+b_n}{2}, b_n\right[$. On pose $a_{n+1} = \frac{a_n+b_n}{2}$ et $b_{n+1} = b_n$.

1. Étude des suites $(a_n)$ et $(b_n)$.

(a) Montrer que les suites $(a_n)$ et $(b_n)$ sont adjacentes et convergent vers $\alpha$.

(b) Montrer que pour tout $n \in \mathbb{N}$, on a $b_{n+1} - a_{n+1} = \dfrac{b_n - a_n}{2}$, puis que $b_n - a_n = \frac{b-a}{2^n}$.

2. On considère l'équation $(E) : 1 - x - x^3 = 0$, où $x$ désigne un réel. On note $f$ la fonction définie sur $\mathbb{R}$ par $f(x) = 1 - x - x^3$.

(a) Justifier que $(E)$ possède une unique solution $\alpha$, et vérifier que $\alpha$ appartient à $]0, 1[$.

(b) Écrire un script, incluant la définition de la fonction $f$ par l'en-tête `def f(x)`, permettant de trouver et d'afficher une valeur approchée de $\alpha$ à 0,001 près.

---