TP 1

Révisions : Listes 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, les listes).
- ✓ 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.

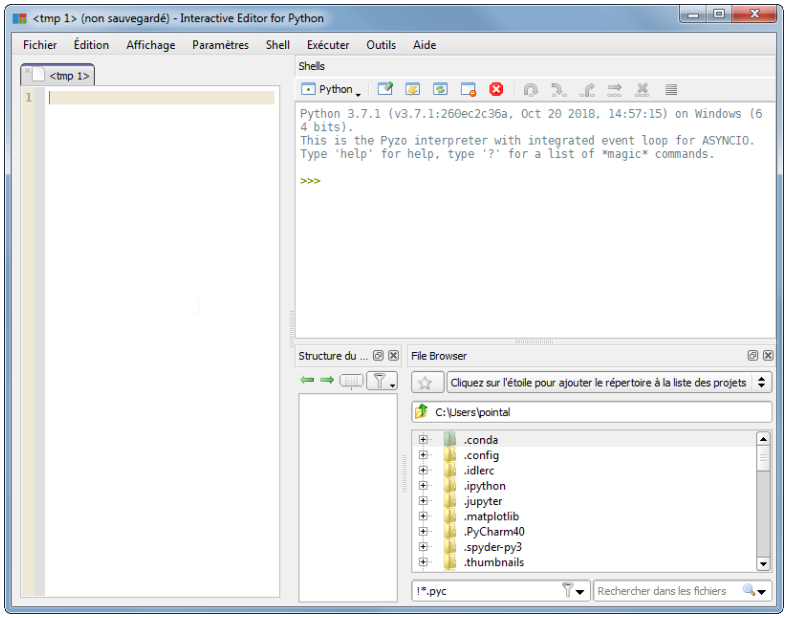
- Opérations sur les entiers et les flottants : `+`, `-`, `*`, `/`, `**`
- Comparaison et connecteurs logiques pour les booléens : `==`, `>`, `<`, `>=`, `<=`, `!=`, `True`, `False`, `and`, `or`, `not`
- Programmation : `=`, `input`, `print`, `if`, `elif`, `else`, `range`, `for`, `while`, `def`, `return`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

Anthony Mansuy
Professeur de Mathématiques en deuxième année de CPGE filière ECG au Lycée Clemenceau (Reims)
Page personnelle : `http://anthony-mansuy.fr`
E-mail : `mansuy.anthony@hotmail.fr`

En préambule

`Python` est un langage de programmation conçu par Guido van Rossum, la première version étant sortie en 1991. Il est libre et gratuit, à télécharger ici. `Python` peut fonctionner avec plusieurs éditeurs : `EduPython`, `Pyzo`, `Spider`... Nous utiliserons cette année l'éditeur `Pyzo`. Vous devez l'installer sur vos ordinateurs personnels pour pouvoir l'utiliser le plus souvent possible en cours d'informatique, lorsque vous préparez un TP ou lorsque vous révisez.

Au lancement de `Python` (avec l'éditeur `Pyzo`), deux fenêtres s'ouvrent :



1. 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 instruction en même temps en les séparant par un point-virgule. L'appui sur la touche « Entrée » provoque l'exécution des instructions.

2. L'**éditeur de texte** :

   C'est ici qu'on écrira 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 cliquant sur 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 entrant.

Définition.

Les opérations sur les nombres entiers ou à virgules (respectivement de type `int` et `float` sur `Python`) :

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

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 lorsque vous entrez des instructions sur Python !
- Le résultat d'opérations sur les entiers ne donnent nécessairement un entier (par exemple, une division d'entier 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)
```

- 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 soit 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 !
2. Le `or` est 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. Attention, `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 croiserons essentiellement les types suivants : `int` (les entiers), `float` (les nombres à virgules), `bool` (les booléens), `str` (les chaînes de caractères), `list` (les listes), `array` (les tableaux).

Exemple. Taper dans la console les instructions suivantes :

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

Il s'agira de rester vigilant car ces affectations sont **globales** et seront utilisées par `Python` dans toute la feuille de travail. On peut libérer la place mémoire occupée par une variable en la détruisant à l'aide de la fonction `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 lignes suivantes :

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

Exécuter les instructions et constater le résultat dans la console.

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 rapidement compréhensibles, on respectera les quelques règles suivantes :

- Écrire une seule instruction par ligne.
- Commenter son script (rôle de chaque variable, d'une boucle, ...) 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.
- Sauter des lignes entre les différentes étapes du script et les commenter.
- 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 pratique en TP pour passer d'un exercice à l'autre...
- Respecter l'indentation (i.e. mettre un espace au début de la ligne) des boucles, des structures conditionnelles... surtout lorsqu'elles s'imbriquent les unes dans les autres. C'est indispensable pour que `Python` puisse repérer le début et la fin d'une boucle ou d'un programme.

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 à virgule, 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.

Remarques.

1. Il faut penser aux deux points après chaque *condition* et après le `else`.
2. On prendra soin de 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 de tabulation 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.

1. La commande `range` ne provoque pas d'affichage.
2. 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 $0, 1, 2, 3, 4, 5, 6$ (début à 0, on avant de 1 en 1, arrêt avant 7).
   - `range(3,9)` énumère les entiers $3, 4, 5, 6, 7, 8$ (début à 3, on avant de 1 en 1, arrêt avant 9).
   - `range(-2,8,2)` énumère les entiers $-2, 0, 2, 4, 6$ (début à -2, on avance de 2 en 2, arrêt avant 8).
   - `range(4,0,-1)` énumère les entiers $4, 3, 2, 1$ (début à 4, on recule de 1 en 1, arrêt avant 0).

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.

1. Comme pour les instructions conditionnelles, il faut faire attention aux deux points après le `range` et à l'indentation !
2. Pour le `range`, on utilisera lorsque c'est possible les expressions simplifiées `range(n,m)` ou `range(n)`.
3. 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.

1. Même remarque que pour les boucles `if` et `for` : attention aux deux points et à l'indentation !
2. 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 ?

```
s = 0
i = 2
while i <=100 :
    s = s+i
    i = i+2
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 $f(x) = x^2 + 2x + 1$, recopier le script suivant dans l'éditeur de texte :

```
def f(x) :
    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.

1. Toujours la même remarque : attention aux deux points et à l'indentation !
2. 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 appelé 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.
3. Les variables utilisées dans le corps d'une fonction sont des variables locales : elles n'existent pas à l'extérieur de la fonction.

Attention.

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 réel 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 sur quelques exemples.

3 Les listes

3.1 Déclaration d'une liste

Définition.

Une liste est une série de valeurs, non nécessairement de même type, que l'on déclare entre crochets en séparant les différentes valeurs par des virgules.

Remarque. Une liste peut n'avoir aucun élément : c'est la liste vide `[]`.

Exemple. Taper dans la console les instructions suivantes :

```
>>> u = [1, True, 2.4, -7, 'monmessage']
>>> u
```

Définition.

Pour déclarer/définir une liste sur Python, plusieurs possibilités :

- *Déclaration exhaustive :* on écrit à la main tous les éléments de la liste entre crochets en séparant les différentes valeurs par des virgules (comme on l'a fait dans l'exemple ci-dessus).
- *À l'aide de la commande* `list(range(n,m,p))` *:* elle permet d'obtenir la liste des éléments énumérés par la commande `range(n,m,p)`.
- *Déclaration en compréhension :* si `u` est une liste (ou une énumération utilisant `range`), dont les éléments sont $x_1, \dots, x_n$, alors :
  - la liste dont les éléments sont $f(x_1), \dots, f(x_n)$ est déclarée ainsi :

    `[f(x) for x in u]`
  - la liste formée des éléments x_i de `u` vérifiant une certaine *condition* est déclarée ainsi :

    `[x for x in u if` *condition*`]`

Exemple. Taper dans la console les instructions suivantes :

```
>>> v = list(range(1,8,3))
>>> w = [2*k+1 for k in v]
```

```
>>> z = [k**2 for k in range(100) if k**2<10]
>>> v; w; z
```

Définition.

Considérons `u` une liste de n éléments. Les éléments de `u` sont indexés :

- de gauche à droite : `u[0]` est le premier élément, `u[1]` est le deuxième, ..., `u[n-1]` est le dernier.
- de droite à gauche : `u[-1]` est le dernier élément, `u[-2]` est l'avant-dernier, ..., `u[-n]` est le premier.

Attention.

Il faut veiller à ne pas sortir de la liste sinon `Python` affiche un message d'erreur du type `list index out of range`

Exemple. Taper dans la console les instructions suivantes :

```
>>> u[4]
>>> v[-1]
>>> z[0]
>>> w[5]
```

Remarques.

1. On peut modifier l'élément d'indice i d'une liste `u` en affectant à `u[i]` une nouvelle valeur. Taper par exemple dans la console les instructions suivantes :

   ```
   >>> u[1] = 5
   >>> u
   ```

2. Attention à la copie d'une liste. Taper par exemple dans la console les instructions suivantes :

   ```
   >>> u = [1, True, 2.4, -7, 'monmessage']
   >>> v = u
   >>> u; v
   >>> v[1] = 5
   >>> u; v
   ```

   La copie d'une liste n'est pas une liste indépendante, elle reste solidaire de la liste-mère. Pour éviter ce problème, on peut la copier élément par élément :

   ```
   >>> v = [x for x in u]
   ```

3.2 Fonctions et commandes sur les listes

Définition.

Considérons deux variables `u` et `v` contenant des listes et `x` une troisième variable quelconque.

- `u.append(x)` : ajoute l'élément `x` à la fin de la liste `u`.
- `u+v` : renvoie la liste formée par la concaténation/juxtaposition des listes `u` et `v` dans cet ordre.
- `u*n` (où `n` est de type entier) : renvoie la liste obtenue en concaténant `n` fois la liste `u`.
- `len(u)` : renvoie la longueur de la liste `u` (c'est-à-dire son nombre d'éléments).
- `x in u` : renvoie `True` si `x` est un élément de `u`, `False` sinon.
- `u.count(x)` : renvoie le nombre d'éléments de la liste `u` ayant la valeur `x`.
- `del u[i]` : supprime de la liste `u` l'élément d'indice i.

Exemple. Taper dans la console les instructions suivantes :

```
>>> u.append(3)
>>> w+z
>>> w*2
>>> len(u)
>>> 4 in z
>>> w.count(5)
>>> del u[4]
```

4 Exercices

Exercice 1 (★)

Écrire un programme qui demande trois réels a, b et c et qui affiche le nombre de racines réels du polynôme du second degré $P(x) = ax^2 + bx + c$. 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` :

$$f(x) = \begin{cases} 1/x & \text{si } x \neq 0, \\ 0 & \text{sinon} \end{cases}$$

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 $\forall n \in \mathbb{N},\ u_{n+1} = u_n^2 + 1$. On admet que $\lim\limits_{n \to +\infty} u_n = +\infty$.

1. Écrire un programme qui, étant donné un entier naturel n, calcule et affiche la valeur de u_n.

2. Écrire un programme qui calcule et affiche le rang du premier terme de cette suite qui est supérieur ou égal à un réel a entré par l'utilisateur.

Exercice 5 (★)

Soit $(u_n)_{n\in\mathbb{N}}$ une 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 $\lim\limits_{n\to+\infty} u_n = 1$.

Écrire un programme permettant de déterminer le plus petit entier naturel n pour lequel

$$|u_n - 1| \leq 10^{-3}.$$

On pourra commencer par définir la fonction valeur absolue sur `Python` puis l'utiliser dans le programme demandé.

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 en `Python` ayant en entrée un entier naturel n et qui renvoie 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 en `Python` ayant en entrée un entier naturel n et qui renvoie le couple 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 un programme qui donne une approximation de $\sum\limits_{k=1}^{+\infty} \dfrac{1}{k^2}$ à ε près pour $\varepsilon > 0$ donné.

Exercice 8 (★)

1. Créer la liste r dont les éléments sont tous les multiples de 5 compris entre 0 et 52.

2. Créer la liste s dont le premier terme est 0, le dernier est 1 et dont les termes progressent d'un dixième à chaque fois.

3. Déterminer les valeurs de a, b et c pour que `range(a,b,c)` énumère : $5, 4, 3, 2, 1, 0$.

4. Créer les listes $u = [1, 2, 4, 8, 16, 32]$ et $v = \left[1, \dfrac{1}{4}, \dfrac{1}{9}, \dfrac{1}{16}, \dfrac{1}{25}, \dfrac{1}{36}, \dfrac{1}{49}\right]$.

5. (a) Créer la liste $x = [3, 1, 0, 2, 4, 0, 2, 1, 6, 0, 0, 3, 2, 0, 0, 1, 0, 3]$.

   (b) Créer la liste y formée à partir de la liste x en remplaçant les 0 par des 10.

Exercice 9 (★★)

1. Écrire une fonction `retirer` qui prend en argument une liste `L` et un élément `x` et renvoie la liste `L` à laquelle on a retiré les éléments égaux à `x` si la liste `L` en contient, ou bien qui renvoie la liste `L` si elle ne contient pas d'élément égal à `x`.

2. Écrire une fonction `dernierevaleurnonnulle` qui prend en argument une liste `L`, qui renvoie 0 si tous les éléments de la liste sont nuls, ou qui renvoie la dernière valeur non nulle de la liste sinon.

Exercice 10 (★★)

1. L'appartenance d'un élément donné à une liste est l'objet de la commande `in` mais il est bien entendu possible de construire un programme qui effectue le meme travail.

   Écrire une fonction `chercher` en `Python` qui prend en arguments d'entrée une liste `L` et un élément `x` et retourne la valeur `True` si l'élément `x` est présent dans la liste, et `False` sinon.

2. En modifiant la fonction précédente, écrire une fonction `chercherindice` en `Python` qui prend en arguments d'entrée une liste `L` et un élément `x` et retourne le premier indice où apparait l'élément `x` s'il est présent dans la liste, et `-1` sinon.

Exercice 11 (★★)

1. `Python` possède une fonction `max` qui, pour une liste `L`, renvoie à l'appel de `max(L)` le maximum de la liste.

   Écrire une fonction `monmax` qui réalise la même chose.

2. Écrire une fonction `max2` qui prend en argument une liste `L` (contenant au moins deux éléments) et renvoie la valeur du deuxième maximum de la liste (c'est-à-dire le deuxième plus grand élément de la liste) avec la consigne suivante :

   (a) En utilisant la fonction `monmax` précédente.

   (b) En faisant une recherche séquentielle qui ne parcourt la liste `L` qu'une seule fois.

Exercice 12 (★★★)

Écrire une fonction `plusproches` qui prend en argument une liste `L` (contenant au moins deux éléments) et renvoie les deux éléments les plus proches (c'est-à-dire les deux éléments dont l'écart est le plus petit parmi tous les couples d'éléments de la listes.

Exercice 13 (★★★)

On souhaite écrire un programme de recherche d'un élément dans une liste triée en utilisant le principe de dichotomie.

1. Écrire une fonction `milieu` qui, pour deux entiers naturels `a` et `b`, renvoie la partie entière de la moyenne de `a` et `b`.

2. Écrire une fonction `recherchedicho` qui, étant donné un élément `x` et une liste triée `L`,

   - initialise `n` à la longueur de `L`, `deb` à 0 et `fin` à `n-1` ;
   - tant que la liste étudiée n'est pas réduite à un seul élément (c'est-à-dire $n \neq 1$) :
     - affecter à `m` la valeur de l'indice central ;
     - si la liste contient un nombre pair d'éléments (c'est-à-dire si $(-1)^n = 1$) :
       - si la moyenne des valeurs centrales est plus grande que `x`, affecter à `fin` la valeur `m` ;
       - sinon, affecter à `deb` la valeur `m+1` ;
     - sinon, la liste contient un nombre impair d'éléments :
       - si l'élément central est `x` : la recherche s'achève et affecter à `deb` et à `fin` la valeur `m` ;
       - sinon, si l'élément central est plus grand que `x`, affecter à `fin` la valeur `m-1` ;
       - sinon, affecter à `debut` la valeur `m` ;
     - affecter à `n` la longueur de la nouvelle liste à étudier, c'est-à-dire `fin - deb + 1` ;
   - à la fin de la boucle tant que, la liste ne contient qu'un élément et :
     - si cette valeur est `x`, renvoyer `True` ;
     - sinon, renvoyer `False`.

3. Expliquer, ligne par ligne, ce que fait le programme `recherchedicho`.
