Lycée Faidherbe, 2019-2020

T.P. d'informatique

MPSI & PCSI

Version du 6 février 2020

Table des matières

TP I

Introduction

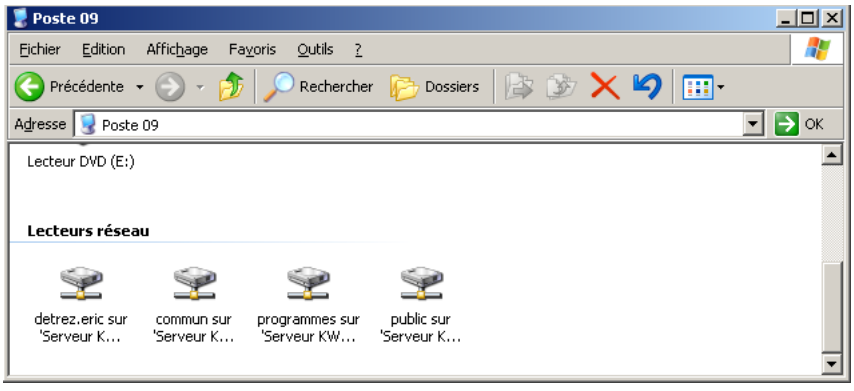
Résumé

Dans ce T.P. nous allons prendre contact avec l'environnement de travail utilisé durant l'année et expérimenter quelques ressources de python. Une grande partie des exercices seront approfondis durant le déroulement de l'année.
Pendant tous les T.P. il est capital de ne jamais rester passif : essayez, lisez les documentations, posez des questions à vos enseignants ou à ceux de vos camarades qui sont les plus rapides.

1 Écosystème

Nous utiliserons des outils dans un environnement qui sera, selon les séances, Windows ou Linux. Leurs fonctionnalités sont semblables mais il est utile de se familiariser avec chacun d'eux.
Pour utilisez les ordinateurs vous devez vous connecter avec votre identifiant (nom.prénom) et votre mot de passe. Le mot de passe initial est votre date de naissance et il **doit** être changé [1].

1.1 Répertoires et fichiers



Le dossier qui contient vos données de manière persistante est le dossier réseau à votre nom.

- Sous Windows il est visible comme un disque dur
- Sous Linux c'est un répertoire à votre nom qui n'a pas une maison comme icône.

Créez un répertoire pour l'informatique dans votre dossier, entraînez-vous à y créer des sous-répertoires pour chaque TP.

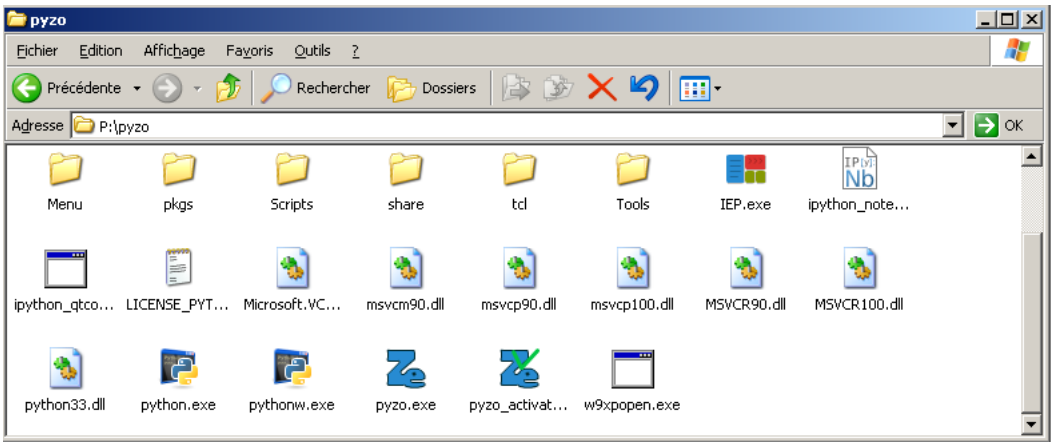
1. Cela n'est possible que sur les ordinateurs sous Windows.

2 Présentation de python

2.1 La boucle interactive

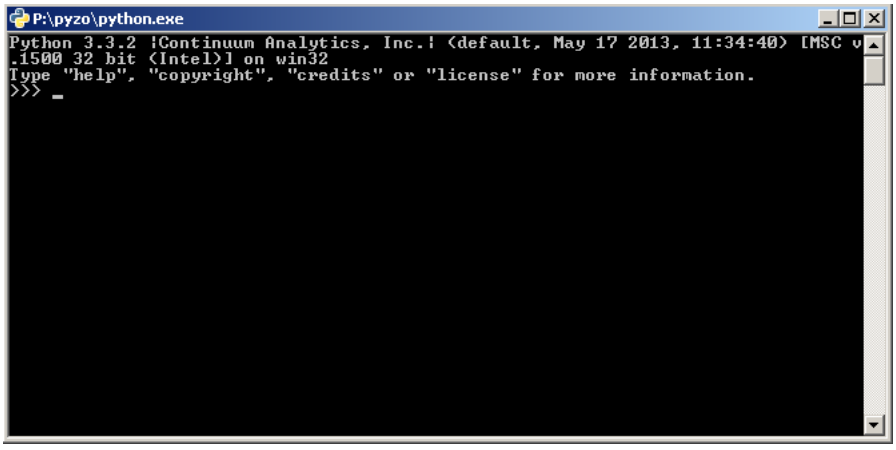
Le moyen le plus immédiat d'utiliser python est la boucle interactive lancée depuis un terminal.

- On peut trouver un terminal adapté, `winpython.exe` dans le dossier `Pyzo`



- Sur les postes Linux on ouvre le terminal par un clic droit sur un dossier "Ouvrir un terminal ici" puis on tape `python`. Python a eu une évolution importante en passant de la version 2 à la version 3 : une version 2.7 subsiste encore (pour l'instant) mais nous utiliserons python3. Dans le cas où on aurait plusieurs versions de Python installées (2 et 3) on lance Python3 avec la commande `python3`.
  À notre niveau les différences ne sont pas nombreuses : la plus visible est que `print` est devenu une fonction et nécessite que ses arguments soient passés entre parenthèses.

Le programme python attend nos instructions et répond immédiatement.
On peut faire exécuter quelques instructions simples : `2 + 2`, ...



On voit ici une particularité de python : c'est un langage interprété. Cela signifie qu'il ne fabrique pas, avec nos instructions, un programme autonome comme le font d'autres langages comme C ou Pascal. Il traduit nos commandes et les fait exécuter directement par l'ordinateur.
L'avantage est que l'on teste très facilement les programmes, l'inconvénient est qu'on a toujours besoin de python et qu'il lui faut tout traduire à chaque fois qu'on a besoin de notre programme.
On peut néanmoins lancer une suite d'instructions en python (un script) directement : on l'exécute par la commande `python "nom_du_fichier_avec_chemin"`.

2.2 Limitations

Cette boucle interactive est un peu primitive,

- il est difficile de gérer des programmes de plus d'une ligne (nous écrirons des programmes d'une dizaine de ligne)
- en particulier il sera difficile de corriger une erreur
- on ne garde pas de trace de ce qui a été écrit.

Il existe une boucle interactive améliorée, souvent installée par défaut, `iPython`. Il y a une version autonome **iPython qtConsole** qui se lance dans une fenêtre, il existe aussi une version **notebook** qui se lance dans un navigateur. Vous trouverez sur internet des cours qui utilisent cette version.

Cependant la méthode la plus générale de programmation consiste à écrire le code dans un éditeur puis à le faire compiler ou interpréter par le langage. Dans le cas de l'interprétation d'un langage interactif l'interprétation peut se faire de manière accumulative : chaque nouveau code est ajouté et on enrichit petit à petit les outils qui sont à notre disposition.

Cette dernière méthode est possible dans Python.

On travaillera donc sous la forme d'un va-et-vient entre les instruction dans l'éditeur et leurs interprétations dans la console.

Il existe de nombreux logiciels qui intègrent un éditeur et une console Python, on parle d'**environnements de développement intégrés** ou IDE en anglais, pour Integrated Development Environment.

On trouve des IDE spécifiques à Python.

- **Pyzo** est utilisé par les oraux de Centrale, c'est celui qui a été choisi dans ce lycée.
- **Spyder** est assez semblable à Pyzo, certains le préfèrent.
- **idle** est souvent installé par défaut, il est utilisé à l'oral de l'ENSAM.
- **Thonny** est un outil à l'aspect épuré mais qui permet une installation facile des modules. De plus son mode de suivi de l'exécution (debug) est très instructif.

Les IDE généralistes (netBeans, Eclipse) peuvent être utilisés avec Python.

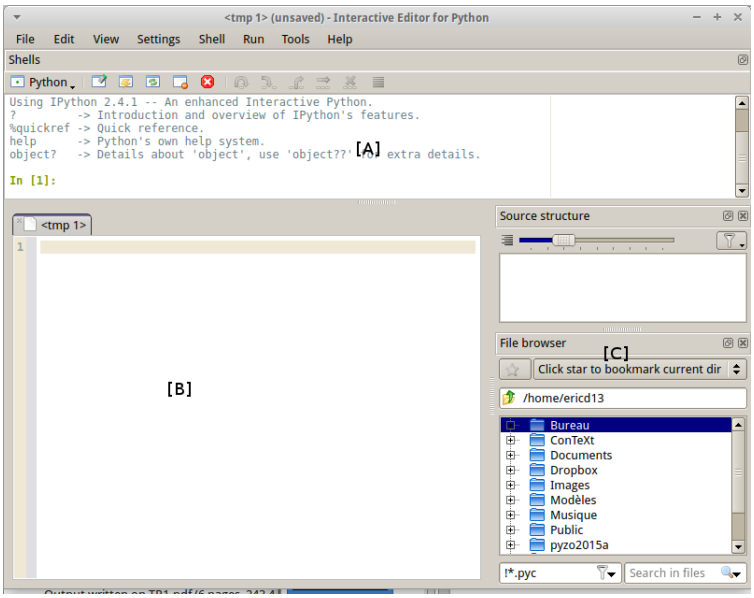
Les éditeurs de texte ont souvent, nativement ou à l'aide de greffons, la possibilité de travailler directement avec Python : emacs, gedit, notepad++ (windows), textmate (mac) ...

3 Pyzo

3.1 Présentation

La fenêtre initiale de Pyzo contient plusieurs parties :

- la console, [A], c'est l'interpréteur comme vu ci-dessus
- l'éditeur, [B], qui permet d'écrire les programmes et de les modifier
- des fenêtres d'utilitaires, [C], qui nous n'utiliserons pas pour l'instant.



On peut déplacer les composantes, sauf celle de l'éditeur, et fermer les outils qui ne sont pas utiles ; on pourra les retrouver en les ouvrant dans le menu **Outils**
Il est indispensable d'employer l'éditeur dès que l'on veut rentrer plusieurs lignes afin d'obtenir un résultat.
Il est par ailleurs recommandé que le fichier écrit dans l'éditeur soit complet :

1. on commence par les importations ; il n'est pas gênant qu'elles soit répétées, python ne les effectue qu'une fois
2. on poursuit par les différentes fonctions
3. on finit par les lignes principales qui appliquent ces fonctions.

On pourra utiliser un fichier-squelette à garder dans son répertoire.
Exemple (ce qui suit le caractère # n'est pas lu par Python).

```
# Importations


# Fonctions


# Programme principal
```

Par ailleurs il vaut mieux faire un fichier par exercice et il est **indispensable** de les sauvegarder régulièrement.

3.2 Le premier programme

Dans la fenêtre de l'éditeur effacer les lignes déjà écrites et écrire le programme le plus employé pour un premier exemple :

```
print("Hello World")
```

- On commence par sauvegarder le fichier en choisissant un emplacement que l'on peut retrouver et qui garde les données, le plus simple est le répertoire personnel du serveur du lycée que l'on peut lire partout dans le lycée.
- Il faut maintenant faire exécuter le fichier dans la console python
  - soit par un item *Exécuter le fichier* du menu *Exécuter*
  - soir par un raccourci ([F5] ou [Ctrl]+[Entrée])
- Tout se passe presque comme si on avait écrit les instructions dans la console.

Rappel : il est indispensable de sauvegarder régulièrement son travail. Pour cela il faut utiliser le raccourci-clavier [Ctrl] + S : on appuie sur la touche [Ctrl] puis, tout en maintenant l'appui sur [Ctrl], on appuie sur la touche S. On peut alors relâcher les deux touches.

3.3 Fonction `print`

En fait le comportement n'est pas exactement le même.
Lorsque l'on écrit une expression dans la console la valeur du résultat est affichée ; quand on le fait depuis un script le résultat est calculé maisn'est pas affiché.

```
>>> a = 1
>>> a
1
```

```
a = 1
a
----------------------------
>>>  (executing line ...)
```

Pour faire afficher le résultat il faudra utiliser la fonction `print`
Il y a plusieurs méthodes pour afficher le résultat dans un texte :

- On peut donner plusieurs arguments à `print`, séparés par des virgules. Python insère lui-même des espaces.

```
print("La somme de",a,"et",b,"vaut",a+b)
```

- On peut réserver des emplacements

```
print("La somme de {} et {} vaut {}".format(a,b,a+b))
```

Les deux affichent : `La somme de 3 et 4 vaut 7`

4 Activités

4.1 Types de base

- **Entiers**
  Vérifier par quelques calculs que les opérations d'addition, `+`*, de multiplication,* `*` *ou d'exponentiation* `**` *fonctionnent comme prévu ; en particulier il n'y a pas de limite à la taille des entiers, par exemple* `2**100`.

  Par contre la division avec `/` *donne un résultat avec virgule, même dans le cas d'un entier divisé par un diviseur de cet entier (dans Python3). L'opération qui correspond à la division euclidienne est* `//`*, le reste est* `n % p`*. Tester ces opérations*
- **Réels**
  Dans python les réels sont appelés `float`.
  Il faut se souvenir que les `float` ne sont que des approximations décimales (en fait binaires) des réels souhaités. Il est donc illusoire de tester si deux résultats de calculs sont égaux même s'ils devraient l'être.
  Tester
  — `0.1 + 0.2`
  — `10 * 0.1`
  — `0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1`
  — `1.21**4.32` *(exponentiation)*
  — `(5.0**0.5)**2.0` *cela devrait calculer* $\sqrt{5}^2$.
  Dans python les fonctions et constantes mathématiques ne sont pas présentes par défaut. Il faut les importer depuis un module `math` (ou `numpy`).

— soit en chargeant le module par `import math` ; on appelle alors les fonctions par `math.cos`.
— soit en chargeant toutes les fonctions du module par [2] `from math import *` ;
on appelle alors les fonctions par `exp`.
— soit en chargeant les fonctions utilisées une par une par `from math import tan` ;
on peut alors appeler chaque fonction directement.

- **Booléens**
  Un autre type élémentaire dans Python est `bool`.
  Ce sont des termes qui peuvent prendre la valeur `False` (faux) ou `True` (vrai).
  Les opérations associées sont `and` (et), `or` (ou), `not` (non).
  — *Tester ces opérations.*
  — *En particulier tester* `True or x, x or True, False or x` *sans avoir défini* `x`.
  — *Trouver un comportement semblable avec* `and`.
- **Complexes**
  Python sait manipuler les complexes, sous la forme `x + yj` avec x et y réels ;
  $1 + i$ se note `1.0 + 1.0j` (ou `1 + 1j`).
  Tester les opérations classiques.
  Les opérations non élémentaires sur les complexes sont dans un module `cmath` semblable à `math` dont nous ne nous servirons que peu. Mais il est amusant d'avoir un résultat de arccos(2).

```
>>> import cmath
>>> cmath.acos(2)
```

4.2 Fonction `type`

On a vu en cours que les valeurs en Python avaient un **type** : entier (`int`), flottant (`float`), chaîne de caractères (`str`), booléen (`bool`) ...
La fonction `type` permet de connaître le type d'un résultat :

```
>>> type(4)
<class 'int'>
```

La réponse semble compliquée mais le type peut être testé simplement

```
>>> type(4) == int
True
>>> type(4) == float
False
```

On pourra tester que des valeurs égales peuvent avoir un type différent

```
>>> 1 == 1.0
>>> type(1) == type(1.0)
```

On peut mélanger curieusement des types, bien que typé, Python fait des conversions implicites.

```
>>> type(1 or True)
```

Un résultat peut avoir des types différents selon les valeurs

```
>>> type(1 or True)
>>> type(0 or True)
```

On retiendra de ces remarques qu'il faut éviter de mélanger des types différents même si cela est possible dans Python.

2. * est une abréviation pour dire tout

4.3 Exercices

1. Voici un petit programme :

```
x = 10
y = 15
z = x + y
x = y
y = z
print(x + y + z)
```

   Que va-t-il donner comme résultat ? Vérifier en l'exécutant depuis l'éditeur.
2. Que voudrait-on faire avec les lignes suivantes ?

```
x = 10
y = 15
y = x
x = y
```

   Obtient-on le résultat souhaité ? Proposer une manière d'échanger les valeurs de deux variables.
3. Écrire un programme qui calcule une durée exprimée en secondes dans une variable `t` sous la forme jours, heures, minutes, secondes.
   Par exemple, pour `t = 325415` le programme devra afficher
   `325415 secondes correspondent à 3 jours 18h 23mn 35s`
   Rappels : `//` pour la division entière, `%` pour le reste.
4. Le polynôme $aX^2 + bX + c$ est défini par trois variables réelles (`float`) `a`, `b` et `c`.
   On suppose que son **discriminant est positif**.
   Écrire un programme qui affiche les deux racines du polynôme.
   Rappel : `sqrt` pour la racine.
5. On définit une variable `n` qui est un entier compris entre 0 et 15.
   Calculer et afficher l'écriture en base 2 de n sous la forme de 4 entiers, chacun valant 0 ou 1 : pour $n = 6$ le programme devra afficher `0110`.
   Transformer ce programme pour qu'il affiche l'entier dont l'écriture en base 2 est le miroir de celle de n.
   Par exemple $13 \to 1101 \to 1011 \to 11$ et $1 \to 0001 \to 1000 \to 8$
6. Il sera parfois utile de tester le temps de calcul des programmes
   Python permet d'importer une fonction simple qui donne une durée : le nombre de secondes écoulées depuis une date précise. La plupart du temps, cette date est l'**Epoch Unix**, le premier janvier 1970 à 0h00.

```
from time import time
print(time())
```

   Mesurez le temps pris par un programme pour être exécuté en calculant le temps avant et après le programme.

4.4 Tracé de fonctions

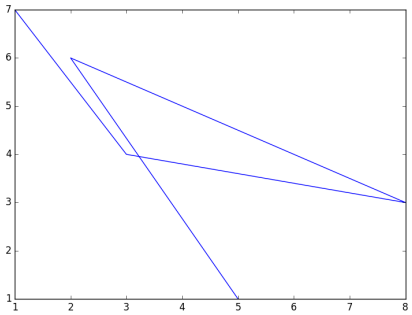
Python permet de tracer le graphe de fonctions. Les outils utiles seront vus au second semestre : nous allons les utiliser comme des boîtes noires.
On a besoin d'utiliser deux bibliothèques : elles ne font pas parties des bibliothèques incluses dans la distribution d'origine. Si vous faites une installation personnelle de Python il faudra peut-être les télécharger en plus.

- `numpy` permet la gestion de suites de valeurs : définition, calculs, ...
- `matplotlib` est le module qui permet l'affichage de données. Les fonction utiles sont dans le sous-module `pyplot`.
- Nous les importerons sous la forme

```
import numpy as np
import matplotlib.pyplot as plt
```

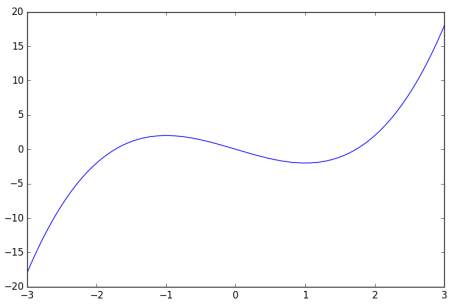
Les abréviations `np` et `plt` sont arbitraires mais elles sont universellement utilisées.
La fonction `plot` de `matplotlib.pyplot` reçoit 2 ensembles de données avec le même nombres d'éléments, $(x_0, x_1, \ldots, x_{n-1})$ et $(y_0, y_1, \ldots, y_{n-1})$ et trace la ligne brisée qui passe par les points de coordonnées (x_i, y_i).

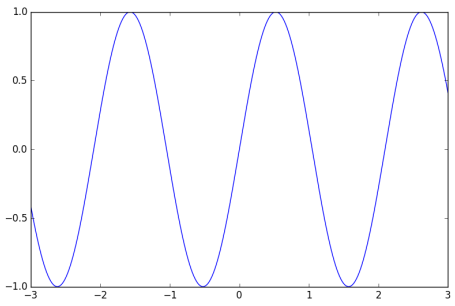
```
plt.plot((1,3,8,2,5),(7,4,3,6,1))
```



Pour une fonction il faut un grand ensemble de données. Pour cela on peut construire une suite de n valeurs régulièrement espacées entre a et b par `x = np.linspace(a,b,n)`.
La liste des ordonnées se calcule alors en considérant x comme une variable : `y = x**3-x`.
Si on veut appliquer des fonctions mathématiques il faut employer celles que fournit `numpy` : `y = np.sin(3*x)`.

```
x= np.linspace(-3,3,1000)
y1 = x**3-3*x
plt.plot(x,y1)
y2 = np.sin(3*x)
plt.plot(x,y2)
plt.show()
```





On remarque que le cadre s'adapte automatiquement.
La fonction `plt.plot` comporte de nombreuses options et on peut ajouter de nombreux éléments aux graphes. Voir la documentation à `http://matplotlib.org/api/pyplot_api.html`

Expérimentez des tracés. On pourra aussi tracer des arcs paramétrés.
Les différents tracés s'accumulent dans la fenêtre graphique. Pour passer à un nouveau graphe ou peut fermer la fenêtre ou utiliser l'instruction `plt.clf()` (**cl**ear **f**rame).

5 Compléments

5.1 Entrées-sorties

Lorsqu'on écrit un programme on peut souhaiter qu'il réponde à des entrées qui ne sont pas toujours les mêmes. Le programme principal peut donc interagir avec l'utilisateur en lui demandant d'entrer des paramètres. La fonction `input` est utile dans ce cas.
L'instruction `arg = input(ch)`

- affiche la chaîne de caractères `ch` (en général elle demande d'entrer la valeur de quelque chose),
- attend que l'on saisisse une chaîne de caractère au clavier suivie de l'appui de la touche **return**,
- puis affecte cette chaîne à la variable `arg`.

Le résultat est une chaîne de caractères, si on attend un nombre pour l'utiliser dans une expression, il faut convertir la chaîne à l'aide des fonctions `int` ou `float`.

- Écrire les instructions qui permettent le dialogue suivant

```
Quel est ton nom ?
Taupin
Bonjour Taupin
```

- Écrire les instructions qui permettent le dialogue suivant

```
Entrer le premier nombre :
3.54
Entrer le second nombre (non nul) :
2.27
3.54 + 2.27 = 5.8100000000000005
3.54 - 2.27 = 1.27
3.54 * 2.27 = 8.0358
3.54 / 2.27 = 1.5594713656387664
```

- On pourra reprendre les exercices 3, 4 et 5 du paragraphe**3.4** sous forme interactive.

5.2 Fractions

Il existe un module, `fractions`, qui permet de gérer le cauchemar des collégiens : les fractions. Il ne contient qu'une fonction, `Fraction`, qui permet de définir une variable sous forme de fraction.

```
from fractions import Fraction
a = Fraction(2,4)
print(a)
----------------------
Fraction(1, 2)
```

On remarquera que la fraction est simplifiée.
On peut effectuer des opérations.

```
b = Fraction(1,6)
print(a+b)
----------------------
Fraction(2,3)
```

Nous verrons que le codage des réels les transforme en décimaux. On peut voir les réels sous leur forme de fraction.

```
>>> Fraction(math.sqrt(2))
Fraction(6369051672525773, 4503599627370496)
```

On peut même trouver une approximation avec un petit dénominateur.

```
>>> p = Fraction(math.pi)
>>> p
Fraction(884279719003555, 281474976710656)
>>> p.limit_denominator(50)
Fraction(22, 7)
```

La méthode `limit_denominator(n)` transforme la fraction par la meilleure approximation avec un dénominateur majoré par n.

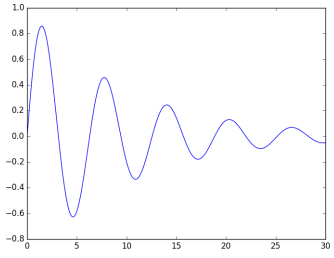
Expérimentez

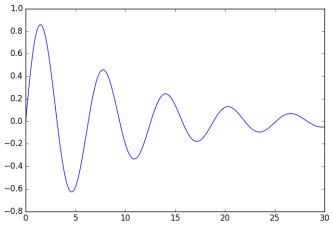
5.3 Variation d'un paramètre dans un graphe

Ce paragraphe utilise l'écriture d'une fonction.
On a parfois besoin de visualiser les graphes d'une fonction avec un paramètre variable.
On peut dessiner plusieurs graphes mais on est limité à un petit nombre et la lecture peut être difficile.
Il existe une fonction, indépendante de l'interface graphique, qui permet de faire varier interactivement un paramètre. Cela se fait sous la forme d'un curseur. Il y a de nombreuses étapes car on modifie la structure interne de la représentation.

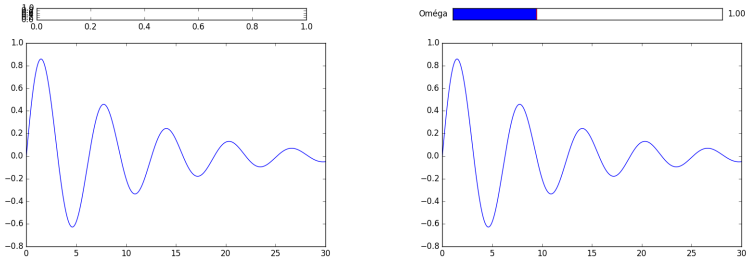
```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from matplotlib.widgets import Slider
4
5  plt.clf()
6  a0 = 1
7  x = np.linspace(0,30,1000)
8  y = np.exp(-x/10)*np.sin(a0*x)
9  gr = plt.plot(x,y)[0]
10
11 plt.subplots_adjust(top=0.8)
12 placeCurseur = plt.axes((0.15, 0.88, 0.7, 0.04))
13 curseur = Slider(placeCurseur, 'Oméga', 0.1, 3,valinit=1)
14
15 def modifier(a):
16     gr.set_ydata(np.exp(-x/10)*np.sin(a*x))
17
18 curseur.on_changed(modifier)
```

1. On importe les modules classiques (lignes 1 et 2) et la fonction qui définit le curseur (ligne3).
2. On commence par créer un graphe avec une valeur du paramètre par défaut en affectant une variable avec le graphe.
   La ligne 5 efface l'éventuel graphe qui existerait.
   Le `[0]` à la ligne 9 sert à extraire la première composante du résultat de `plot`.
3. On réserve une place pour le curseur dans le graphe. Pour cela on diminue la proportion du graphe dans la fenêtre à la ligne 11 : `top=0.8` signifie que le haut du graphe est à 80% du cadre, par défaut il est à 90%. On peut aussi réserver de la place en bas, (`bottom=0.2`).

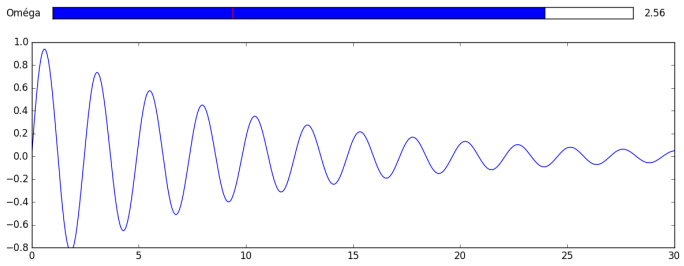




4. On définit le rectangle du curseur à la ligne 12.
   L'ensemble de 4 paramètres (noter qu'ils ne forment qu'une variable pour la fonction d'où les doubles parenthèses) signifient ici
   le rectangle part à 15% de la gauche et à 88% du bas du cadre
   sa largeur est de 70% de celle du cadre et sa hauteur de 4% de la hauteur du cadre.
   Tout reste proportionné si on change la taille de la fenêtre.
5. On place le curseur dans le rectangle à la ligne 13.
   La fonction `Slider` admet 4 paramètres plus un paramètre nommé.
   (a) Le premier est le rectangle défini auparavant
   (b) `'Oméga'` est l'intitulé qui sera écrit à gauche du curseur
   (c) `0.1` et `3` sont les bornes entre lesquelles varie le curseur
   (d) `valinit=1` permet de donner une valeur initiale, il est utile de donner celle du graphe initial.



6. Il faut maintenant mettre à jour le graphe quand la valeur du curseur déterminée à la souris change.
   On commence par définir une fonction de changement qui reçoit la valeur du curseur (lignes 15 et 16). `gr.set_ydata` modifie l'ensemble des ordonnées du graphe (`gr` ici).
   Cette fonction est appelée à chaque changement de valeur du paramètre en l'associant au curseur par la méthode `on_changed` (ligne 18).



Expérimentez.

TP II

Fonctions

Résumé *Dans ce T.P. nous allons utiliser des fonctions utiles de python, définir les premières fonctions et utiliser les instructions de branchement.*
Dans un premier temps on donnera un cadre de présentation du code écrit dans l'éditeur qu'il conviendra de respecter à chaque T.P.

1 Mise en page dans l'éditeur

Dans les T.P. nous importerons des fonctions définis dans des modules, nous définirons nos propres fonctions et nous utiliserons ces fonctions pour obtenir des résultats.
Il est utile de structurer le code écrit.
Pour cela chaque T.P. devra être écrit en séparant les 3 parties décrites ci-dessus.
On commencera donc par indiquer, par un commentaire, les parties selon la forme ci-dessous.

```
## Importations



## Fonctions



## Principal
```

Importations est l'endroit où on importe les modules, par exemple `from math import *`
C'est aussi ici que l'on pourra charger des fichiers et définir des constantes
Fonctions est l'endroit où on écrit les fonctions.
Principal est l'endroit où on écrit les script qui utilisent les fonctions.

2 Fonctions

2.1 Utiliser le résultat

On veut écrire une fonction qui calcule le carré d'une variable.
On propose les 3 écritures suivantes à écrire dans la partie `Fonctions`.

```
def f1(x):
    y=x**2
    return y
```

```
def f2(x):
    y=x**2
    print(y)

def f3(x):
    y=x**2
```

Pour chacune on teste le calcul par le script suivant à écrire dans la partie `Principal`

```
print("******* f1 *******")

print("Calcul direct")
f1(5)

print("Calcul imprimé")
print("f1(5) = ",f1(5))

print("Résultat enregistré")
a = f1(5)
print("f1(5) a été enregistré sous la forme : ",a)
```

On notera que la fonction `print` peut avoir plusieurs arguments ; ils seront tous représentés à l'écran séparés par un espace.

Exercice II.1 — Retour de fonctions
Écrire la suite du script pour `f2` *et* `f3`*.*
Avant l'exécution *écrire avec un stylo (donc prévoir) les résultats renvoyés par la machine. Lancer ensuite l'exécution puis vérifier les réponses : discuter.*

> On gardera le principe que, sauf si une sortie écran est demandée explicitement,
> **la valeur renvoyée par une fonction se fait par l'instruction `return`.**

2.2 Paramètres

Un fonction peut ne pas avoir de paramètres : on peut écrire une fonction qui fait une suite constante d'instructions, on peut simuler un événement aléatoire, ...
Le module `random` contient une fonction `randint` telle que `random.randint(a,b)` renvoie un entier choisi au hasard entre a et b (bornes comprises).

Exercice II.2 — Fonction sans paramètre
Écrire une fonction sans variable `tirageDe()` *qui simule un tirage de dé.*
Faire quelques appels de cette fonction.

Exercice II.3 — Fonction avec variable optionnelle
Écrire une fonction `bonjour` telle que

- `bonjour()` écrive "*Bonjour inconnu*" à l'écran
- `bonjour("voisin")` écrive "*Bonjour voisin*" à l'écran

2.3 Fonctions simples

Exercice II.4 — Polynôme
Écrire une fonction `P` telle que `P(a)` renvoie la valeur du polynôme $X^3 - 7.2X + 1.4$ en a.

Exercice II.5 — Parité
Écrire une fonction `est_pair` telle que `est_pair(n)` renvoie `True` ou `False` selon que l'entier n est pair ou impair. La fonction **ne doit pas** utiliser l'instruction `if`.

Exercice II.6 — Intervalle
Écrire une fonction `entre` telle que `entre(a, b, x)` renvoie `True` ou `False` selon que le réel x est appartient à l'intervalle $[a;b]$ ou non. La fonction **ne doit pas** utiliser l'instruction `if`.
Remarque : PYTHON permet d'écrire la fonction sans utiliser `and`.

Exercice II.7 — Maximums
La fonction `max` renvoie le maximum de 2 nombres.
Écrire une fonction `max3` renvoyant le plus grand parmi 3 nombres.
Écrire une fonction `max4` renvoyant le plus grand parmi 4 nombres.

Exercice II.8 — Devinette
Importer le module `math` afin de pouvoir utiliser la constante `pi` et la fonction `floor`.
Tester la fonction suivante pour `nombre = pi` et `n = 1`, `n = 2`, `n = 3`.

```
def devine(nombre, n):
    a = 10**(n-1)*nombre
    b = a - floor(a)
    return floor(10*b)
```

Que fait cette fonction ?

3 Boucle `for`

3.1 Répétitions simples

L'usage le plus simple d'une boucle `for` est de faire exécuter un certain nombre de fois les mêmes instructions. On emploie la structure `for i in range(n)` pour répéter n fois, la variable i n'est pas utilisée [1]

Exercice II.9 — La punition
Écrire une fonction `ecrirePunition(texte,n)` qui reçoit deux paramètres
- `texte` est une chaîne de caractères
- `n` est un entier

et qui écrit n fois le texte à l'écran.

Exercice II.10 — Une suite
Écrire une fonction `heron(n,u0)` qui calcule (et renvoie) le terme u_n de la suite (u_p) définie par $u_0 =$ `u0` et $u_{p+1} = \frac{1}{2}\left(u_p + \frac{2}{u_p}\right)$ (suite de Héron).

Exercice II.11 — Suite de Fibonacci : 1
On considère la suite (de Fibonacci) définie par $F_0 = 0$, $F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n$ pour $n \geqslant 0$.
Écrire un programme `fibo(n)` qui renvoie F_n.
On pourra maintenir deux variables qui correspondent à 2 valeurs consécutives de la suite.

3.2 Utilisation de l'indice de boucle

Exercice II.12 — Suite de Fibonacci : 2
Affichez les valeurs de F_0 à F_{12}.

Exercice II.13 — Suite harmonique
Écrire une fonction `harmo(n)` qui renvoie $\displaystyle\sum_{k=1}^{n} \frac{1}{k}$ (pour $n \geqslant 1$).

Exercice II.14 — Factorielle
Écrire une fonction `facto(n)` qui renvoie $n!$.

1. On pourrait écrire `for _ in range(n)`, le caractère "_" représente une variable sans nom.

Exercice II.15 — Coefficients binomiaux
Écrire une fonction `binomial(n,p)` qui renvoie $\binom{n}{p}$.
On pourra écrire deux versions : l'une emploie la définition par des factorielles, une autre utilisera la propriété $\binom{n}{p} = \frac{n}{p}\binom{n-1}{p-1}$ pour $n \geqslant p \geqslant 1$.

Exercice II.16 — Somme de puissances
Écrire une fonction `sommePuissances(n,p)` qui renvoie $\sum_{k=1}^{n} k^p$.
Calculer, pour quelques valeurs de n, `sommePuissances(n,3)-(sommePuissances(n,1))**2`.

Exercice II.17 — Série
On veut étudier la suite défine par la somme $S_n = \sum_{k=0}^{n} \frac{2^k}{k!}$.

- Écrire une fonction `a(n)` qui calcule $a_n = \frac{2^n}{n!}$; en déduire une fonction `S(n)` qui calcule S_n. Combien d'opérations sont effectuées lors de l'appel de `S(n)` ?
- Proposer une fonction `S1(n)` qui calcule S_n en effectuant un nombre d'opération proportionnel à n (en fait proportionnel à $n+1$).

Exercice II.18 — Matchs de foot
Dans un championnat à n équipes, chaque équipe se déplace chez toutes les autres (sauf elle-même). Écrire une fonction `deplacements(n)` qui écrit à l'écran tous les déplacements, les équipes étant symbolisées par un nombre entre 1 et n.

3.3 Suite de Muller (1993)

La suite de Muller est définie par $u_0 = \frac{5}{2}$, $u_1 = \frac{17}{5}$ et $u_{n+2} = 30 - \frac{129}{u_{n+1}} + \frac{100}{u_n.u_{n+1}}$ pour $n \geqslant 0$.

Exercice II.19 — Calcul des termes
Écrire une fonction `muller(n)` qui calcule le terme u_n de cette suite.
On pourra s'inspirer de la fonction qui calcule les termes de la suite de Fibonacci.
Calculer les 50 premières valeurs de u_n. Quelle semble être la limite ?

Exercice II.20 — Étude mathématique
Prouver, par récurrence, que $u_n = \frac{1+4^{n+1}}{1+4^n}$. En déduire la limite de u_n.
On voit apparaître les limites du calcul avec les nombres flottants ; on y reviendra.

Les valeurs de la suites sont des fractions rationnelles. Pour éviter les erreurs d'arrondi on peut utiliser le module `fractions`. Il ne contient qu'une fonction, `Fraction`, qui permet de définir une variable sous forme de fraction.

```
from fractions import Fraction
a = Fraction(2, 6)
print(a)
print(float(a))
-----------------------
1/3
0.3333333333333333
```

On remarquera que la fraction est simplifiée.
On peut effectuer les opérations usuelles avec les fractions.

Exercice II.21 — Usage des fractions
Écrire une fonction `muller_frac(n)` qui calcule le terme u_n de la suite de Muller en utilisant les fractions. Calculer les 50 premières valeurs de u_n ; quelle semble être la limite ?

4 Compléments : polynômes de degré 3

Le but de ces exercices est de déterminer les 3 racines (complexes) d'un polynôme de degré 3 : $P(X) = X^3 + aX^2 + +bX + c$. Même dans le cas d'un polynôme à coefficients réels et dont les 3 racines sont réelles ont doit calculer des valeurs intermédiaires complexes.
Python permet le calcul algébrique des nombres complexes sous la forme `a+bj` pour $a + ib$.

- i est noté `1j`
- $2 + 2i$ est noté `2+2j`
- Lorsque les coefficients doivent être calculés, il sera nécessaire d'utiliser le constructeur `complex` qui prend deux réels en arguments.

```
>>> z = complex(1+3, 2*3)
 4 + 6j
```

- Les parties réelle et imaginaire sont accessibles par les méthodes `real` et `imag`.

```
>>> z = 2+3j
>>> z.real
2.0
>>> z.imag
3.0
```

- On peut calculer la somme, le produit, le quotient de deux nombres complexes.
- On peut calculer **une** racine n-ième : `(-2+2j)**(1/3)` renvoie `(1.0000000000000002+1j)` ; on notera l'erreur d'arrondi.
- On peut définir un nombre complexe à partir de son module r et de son argument φ à l'aide de la fonction `rect(r, phi)` du module `cmath`.

```
import cmath as c
-------------------------
>>> c.rect(2,c.pi/3)
(1.0000000000000002+1.7320508075688772j)
```

On commence par le cas simple $X^3 - u$.

Exercice II.22
Écrire une fonction `racines0(u)` *qui reçoit un complexe u et renvoie ses 3 racines dans $\mathbb{C}$.*

Dans le cas du polynôme $P(X) = X^3 + pX + q$ on démontre (exercice de mathématique) que si on pose $X = a + b$ en imposant $ab = -\frac{p}{3}$ alors $a^3 + b^3 = -q$. Comme on a aussi $a^3b^3 = -\frac{p^3}{27}$ on voit que a^3 et b^3 sont les racines de $Q(X) = X^2 + qX - \dfrac{p^3}{27}$.
On obtient alors 2 valeurs pour a^3 ce qui donne 6 valeurs pour a, cependant on démontre (autre exercice mathématique) que

- si on choisit une des racines de Q, α,
- si on attribue à a prend successivement les 3 racines de α comme valeur,
- et si la valeur de b est déterminée par $ab = -\frac{p}{3}$,

on obtient alors les trois racines de P en calculant $a + b$.

Exercice II.23
Écrire une fonction `racines1(p, q)` *qui reçoit les deux paramètres d'un polynôme $X^3 + pX + q$ et qui en renvoie les 3 racines.*
On supposera p non nul

Exercice II.24
Dans le cas général, $P(X) = X^3 + aX^2 + +bX + c$, montrer que u est racine de P si et seulement si $u + \frac{a}{3}$ est racine d'un polynôme de la forme $P_1(X) = X^3 + pX + q$ dont on déterminera les coefficients p et q en fonction de a, b et c.
En déduire une fonction `racines(a, b, c)` *qui reçoit les trois paramètres d'un polynôme $X^3 + aX^2 + bX + c$ et qui en renvoie les 3 racines.*
On traitera à part le cas $p = 0$ en utilisant une instruction conditionnelle.

On suppose maintenant que les coefficients sont réels.

Exercice II.25
En raison des erreurs d'arrondi les racines réelles peuvent être calculées avec un partie imaginaires non nulle : on conviendra qu'un complexe est réel si sa partie imaginaire est inférieure à 10^{-10}.
Modifier `racines(a, b, c)` *en* `racinesR(a, b, c)` *de telle manière que les racines retournées soient sous forme réelle si ce sont des réels.*
On peut montrer (en étudiant les variations de $f(x) = x^3 + px + q$) que le polynôme $P_1(X) = X^3 + pX + q$ admet 3 racines réelles si et seulement si le discriminant de $Q(X) = X^2 + qX - \frac{p^3}{27}$ est négatif.

5 Solutions

Solution de l'exercice II.1 -

- L'exécution directe ne donne un résultat que pour `f2` mais ce n'est en fait pas utile.
- `f1` imprime bien le résultat, `f2` imprime le résultat mais ce n'est pas au bon endroit.
- `f1` est le seul a donner le résultat attendu à une variable.

Solution de l'exercice II.2 -

```
from random import randint

def tirageDe():
    k = randint(1,6)
    return k
```

Solution de l'exercice II.3 -

```
def bonjour(nom = "Inconnu"):
    print("Bonjour ", nom)
```

Solution de l'exercice II.4 -

```
def P(x):
    return a**3 - 7.2*a+1.4
```

Solution de l'exercice II.5 -

```
def est_pair(n):
    return n%2 == 0
```

Solution de l'exercice II.6 -

```
def entre(a, b, x):
    return a <= x and x <= b
```

L'écriture suivante est possible

```
def entre(a, b, x):
    return a <= x <= b
```

Solution de l'exercice II.7 -

```
def max3(x, y, z):
    a = max(x, y)
    return max (a, z)
```

```
def max4(x, y, z, t):
    a = max(x, y)
    b = max(z, t)
    return max (a, b)
```

Solution de l'exercice II.8 - Elle calcule la n-ième décimale du nombre.

Solution de l'exercice II.9 -

```
def ecrirePunition(texte,n):
      """Entrées : une chaîne de caractères et un entier
         Sortie  : la chaîne est écrite n fois"""
    for i in range(n):
        print(texte)
```

Solution de l'exercice II.10 -

```
def heron(n,u0):
      """Entrées : un entier positif n et un réel u0
         Sortie  : le n-ième terme de la suite de Héron"""
    u = u0
    for i in range(n):
        u = (u + 2/u)/2
    return u
```

Solution de l'exercice II.11 -

```
def fibo(n):
    """Entrée : un entier positif n
       Sortie : le n-ième nombre de Fibonacci"""
    F = 0
    F_suivant = 1
    for i in range(n): # On calcule les couples (F1, F2), (F2,
        F3), ..., (Fn,Fn+1)
        F_vieux = F
        F = F_suivant
        F_suivant = F + F_vieux
    return F
```

Solution de l'exercice II.12 -

```
for i in range(13):
    print("F{} vaut {}".format(i, fibo(i)))
```

Solution de l'exercice II.13 -

```
def harmo(n) :
      """Entrée : un entier strictement positif
         Sortie : la valeur de Hn"""
    h=0
    for k in range(n):
        h = h + 1/(k+1)
    return k
```

Solution de l'exercice II.14 - Il faut initialiser par 1 car on fait des produits.

```
def facto(n):
      """Entrée : un entier positif
         Sortie : la valeur de n!"""
    f = 1
    for k in range(n):
        f = f*(k+1)
    return f
```

Solution de l'exercice II.15 -

```
def binomial(n,p):
    return facto(n)/facto(p)/facto(n-p)
```

Il faut initialiser par 1 car on fait des produits.

```
def binomial(n,p):
    resultat  = 1
    for i in range(1, p+1):
        resultat = resultat*(n-p+i)//i
    return resultat
```

Solution de l'exercice II.16 -

```
def sommePuissances(n,p):
      """Entrées : deux entiers positifs
         Sortie  : la valeur de sum(k^p,k=1..n)"""
    s=0
    for k in range(1, n+1):
        s = s + k**p
    return s
```

```
for n in range(1, 11):
   print("La somme des cubes de 1 à {} vaut {}".format(n,
      sommePuissances(n,3)))
   print("La somme des entiers de 1 à {} au carré vaut {}".
      format(n,(sommePuissances(n,3))**2))
```

Solution de l'exercice II.17 -

- ```
  def a(n):
      terme = 1
      for i in range(n):
          terme = terme*2/(i+1)
      return terme

  def S(n):
      somme = 0
      for k in range(n+1):
          somme = somme + a(k)
      return somme
  ```

  L'appel de `a(k)` effectue $2k$ opérations donc l'appel de `S(n)` effectue $C(n) = \sum_{k=0}^{n} 1 + 2k = n + 1 + 2\sum_{k=0}^{n} k = (n+1)^2$ opérations.
- On calcule a_n dans la boucle

  ```
  def S1(n):
      a = 1
      somme = 0
      for k in range(n+1):
          somme = somme + a
          a = a*2/(k+1)
      return somme
  ```

  On effectue maintenant $C_1(n) = \sum_{k=0}^{n} 3 = 3(n+1)$ opérations.
  ```

Solution de l'exercice II.18 - Ne pas oublier d'enlever les (non-)déplacements à domicile.

```
def deplacements(n):
      """Entrée : un entier positif
         Sortie : tous les déplacements entre i et j
                  pour 1 <= i,j <=n et i != j sont écrits"""
    for i in range(1,n+1):
        for j in range(1,n+1):
            if i != j:
                print("L'équipe {} se déplace contre l'équipe
                   {}.".format(i,j))
```

Solution de l'exercice II.19 -

```
def muller(n):
    u = 5/2
    u_suivant = 17/5
    for i in range(n):
        u_avant = u
        u = u_suivant
        u_suivant = 30 - 129/u + 100/u/u_avant
    return u
```

Dans la partie principale.

```
for k in range(50):
    print("u{} vaut {}".format(k, muller(k)))
```

La suite semble converger vers 25 alors qu'elle semblait s'approcher de 4 au début.

Solution de l'exercice II.20 - $\frac{1+4^1}{1+4^0} = \frac{5}{2} = u_0$, $\frac{1+4^2}{1+4^1} = \frac{17}{5} = u_1$,
si $u_n = \frac{1+4^{n+1}}{1+4^n}$ et $u_{n+1} = \frac{1+4^{n+2}}{1+4^{n+1}}$ alors
$u_{n+2} = 30 - \frac{129(1+4^{n+1})}{1+4^{n+2}} + \frac{100(1+4^n)}{1+4^{n+2}} = \frac{30(1+4^{n+2}) - 129(1+4^{n+1}) + 100(1+4^n)}{1+4^{n+2}}$
$u_{n+2} = \frac{(30-129+100) + 4^n(30.16 - 129.4 + 100)}{1+4^{n+2}} = \frac{1+4^n.64}{1+4^{n+2}} = \frac{1+4^{n+3}}{1+4^{n+2}}$
On peut écrire $u_n = \frac{4^{-n}+4}{4^{-n}+1}$ donc la limite est 4.

Solution de l'exercice II.21 -

```
def muller_frac(n):
    u = Fraction(5, 2)
    u_suivant = Fraction(17, 5)
    for i in range(n):
        u_avant = u
        u = u_suivant
        u_suivant = 30 - 129/u + 100/u/u_avant
    return u

for k in range(50):
    print("u{} vaut {}, {} sous forme décimale".format(k,
       muller_frac(k), float(muller_frac(k))))
```

Solution de l'exercice II.22 -

```
def racines3(u):
    """Entrées : un complexe
       Sortie  : les 3 racines de u"""
    v1 = u**(1/3)
    w = c.rect(1,2*c.pi/3)
    v2 = v1*w
    v3 = v2*w # ou v1/w
    return v1, v2, v3
```

Solution de l'exercice II.23 -

```
def racines1(p, q):
    """Entrées : les coefficients p et q de X**3+pX+q
       Sortie  : les 3 racines du polynôme"""
    delta  = q**2 + 4*p**3/27
    alpha = (-q + delta**(1/2))/2
    a1, a2, a3 = racines0(alpha)
    b1 = -p/3/a1
    b2 = -p/3/a2
    b3 = -p/3/a3
    return a1+b1, a2+b2, a3+b3
```

Solution de l'exercice II.24 - $\left(u+\frac{a}{3}\right)^3 = u^3 + a.u^2 + \frac{a^2}{3}u + \frac{a^3}{27}$ or $u^3 + a.u^2 = -b.u - c$ car u est racine.
Ainsi $\left(u+\frac{a}{3}\right)^3 = \left(\frac{a^2}{3} - b\right)u + \frac{a^3}{27} - c = \left(\frac{a^2}{3} - b\right)\left(u+\frac{a}{3}\right) + \frac{a^3}{27} - c - \frac{a^3}{9} + \frac{ab}{3}$.
$p = b - \frac{a^2}{3}$ et $q = c - \frac{ab}{3} + \frac{2a^3}{27}-$.

```
def racines(a, b, c):
    """Entrées : les coefficients de X**3+aX**2+bX+c
       Sortie  : les 3 racines du polynôme"""
    p = b - a**2/3
    q = c - a*b/3 + 2*a**3/27
    if p == 0:
        u1, u2, u3 = racines0(-q)
    else:
        u1, u2, u3 = racines1(p, q)
    return u1 - a/3, u2 - a/3, u3 - a/3
```

Solution de l'exercice II.25 -

```
def racinesR(a, b, c):
    """Entrées : les coefficients de X**3+aX**2+bX+c
       Sortie  : les 3 racines du polynôme"""
    p = b - a**2/3
    q = c - a*b/3 + 2*a**3/27
    u1, u2, u3 = racines1(p,q)
    v1 = u1 - a/3
    v2 = u2 - a/3
    v3 = u3 - a/3
    if abs(v1.imag) < 1e-10:
        v1 = v1.real
    if abs(v2.imag) < 1e-10:
        v2 = v2.real
    if abs(v3.imag) < 1e-10:
        v3 = v3.real
    return v1, v2, v3
```

TP III

If et listes

Résumé

Dans ce T.P. nous allons utiliser les outils vus en cours :

- *les instructions conditionnelles :* `if, else, elif`,
- *les listes python, vues comme des tableaux de valeurs.*

1 Instruction conditionnelles

Exercice III.1 — Analyser des structures conditionnelles

Plusieurs fonctions ont pour vocation de regarder si un nombre est divisible par 2 ou par 3 (division euclidienne). Indiquer quelle est la valeur renvoyée par les fonction en `20`, `12`, `105` et `7`.
Que peut-on en conclure ?

```
def f1(a):
    if a%2 == 0:
        b = 2
    if a%3 == 0:
        b = 3
    return b
```

```
def f2(a):
    if a%2 == 0:
        b = 2
    if a%3 == 0:
        b = 3
    else:
        b = 0
    return b
```

```
def f3(a):
    if a%2 == 0:
        b = 2
    elif a%3 == 0:
        b = 3
    else:
        b = 0
    return b
```

```
def f4(a):
    if a%2 == 0:
        return 2
    if a%3 == 0:
        return 3
    return 0
```

```
def f5(a):
    if a%2 == 0:
        return 2
    elif a%3 == 0:
        return 3
    return 0
```

```
def f6(a):
    if a%3 == 0:
        b = 3
    elif a%2 == 0:
        b = 2
    else:
        b = 0
    return b
```

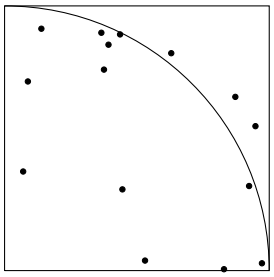
Exercice III.2 — Année bissextile

Écrire une fonction `bissextile(annee)` qui renvoie `True` ou `False` selon que l'année donnée en entrée sous forme d'un entier est ou n'est pas bissextile.
Un année est dite bissextile si c'est un multiple de 4, sauf si c'est un multiple de 100. Toutefois, si c'est un multiple de 400, alors elle est considérée comme bissextile.

Exercice III.3 — Jours dans le mois
Écrire une fonction `jours(mois, annee)` qui renvoie le nombre de jours du mois donné par son numéro (entre 1 et 12) ; le paramètre année sert à déterminer le nombre de jours de février (mois 2) selon que l'année est ou n'est pas bissextile.

Exercice III.4 — Second degré
Écrire une fonction qui étant donnée une équation du second degré $ax^2 + bx + c = 0$ identifiée par ses 3 coefficients réels, renvoie la liste de ses racines réelles. On supposera, sans avoir besoin de le vérifier, que a est non nul.

Exercice III.5 — Méthode de Monte-Carlo
La méthode de Monte-Carlo permet d'approcher l'aire d'une surface incluse dans un rectangle. Pour cela on choisit n points aléatoirement dans le rectangle et on note la proportion de ceux qui sont dans la surface : r. L'aire de la surface sera approchée par $r.S$ où S est l'aire du rectangle. Dans le cas d'un quart-de-cercle inclus dans le carré unité $[0;1] \times [0;1]$ on calcule ainsi une valeur approchée de $\frac{\pi}{4}$.
Dans l'exemple ci-contre on peut approcher π par $4.\frac{12}{15} = 3,2$
La fonction `random()` du module `random` renvoie une valéatoire entre 0 et 1.



Écrire une fonction `approx_pi(n)` qui utilise cette méthode pour déterminer une valeur approchée de π en choisissant n points dans le carré unité.

```
>>> approx_pi(1000000)
3.140632
```

Exercice III.6 — Approximation
Écrire une fonction `approximation(x, n)` qui envoie un couple d'entiers `(a, b)` tel que $\frac{a}{b}$ est la meilleure approximation de x sous forme d'une fraction de dénominateur inférieur ou égal à n.
Par exemple `approximation(math.sqrt(2), 50)` renverra `(41, 29)`
et `approximation(-math.pi, 50)` renverra `(-22, 7)`.

Exercice III.7 — Bataille navale
On considère un jeu de bataille navale simplifié : il ne s'agit pas de couler le porte-avions mais plutôt une barque tenant sur une seule case repérée par ses coordonnées `ligne0` et `colonne0` qui sont des variables globales. On fait un tir sur une case repérée par ses coordonnées `ligne` et `colonne`. On veut alors le comportement suivant :

- si la barque est exactement sur la case considérée, le programme renvoie le texte "**Coulé**",
- si le tir atteint la bonne ligne ou la bonne colonne, le programme renvoie "**En vue**",
- si le tir est totalement raté, le programme renvoie "**À l'eau**".

On propose la fonction suivante :

```
def bataille(ligne,colonne):
    if ligne == ligne0 or colonne == colonne0:
        return "En vue"
    elif ligne == ligne0 and colonne == colonne0:
        return "Coulé"
    else :
        return "À l'eau"
```

1. Pourquoi cette fonction n'est-elle pas correcte ?
2. Proposer une fonction valide.
3. Proposer une fonction qui n'utilise pas les opérateurs `and` et `or`.
4. En général, à la bataille navale, un bateau n'est "en vue" que si la case visée est immédiatement voisine de celle du bateau. Modifier le programme de bataille navale afin de tenir compte de cette règle. On commencera par le cas où les cases diagonalement adjacentes de la barque sont "en vue" puis on traitera le cas où elle ne le sont pas.

Exercice III.8 — Triangles rectangles

On cherche les triangles rectangles dont les longueurs des cotés sont des entiers.

Plus précisément on cherche les entiers a, b et c tels que $a \leqslant b$ et $a^2 + b^2 = c^2$.

On dit que (a, b, c) est un **triplet pythagoricien**.

Écrire une fonction `pythagoriciens(n)` qui renvoie le nombre de triplets pythagoriciens (a, b, c) avec $1 \leqslant a \leqslant n$, $1 \leqslant b \leqslant n$ et $1 \leqslant c \leqslant n$.

Il y a 20 solutions pour $n = 50$, 127 pour $n = 200$ et 881 pour $n = 1000$.

2 Listes

Exercice III.9 — Produit

Écrire une fonction qui calcule le produit des termes d'une liste.

Exercice III.10 — Positivité

Tester la positivité d'une liste est ambigu : on peut tester s'il existe un élément positif ou si tous les éléments sont positifs.

1. Écrire une fonction `tout_positif(liste)` qui renvoie `True` ou `False` selon que tous les éléments de la liste sont positifs ou non.
2. Écrire une fonction `existe_positif(liste)` qui renvoie `True` s'il existe au moins un élément positif dans la liste et `False` sinon.

Exercice III.11 — Retourner une liste

1. Écrire un programme qui renvoie une liste passée en paramètre dans l'ordre inverse.
2. Écrire un programme qui retourne une liste "en place".

Exercice III.12 — Tirage d'un dé

Dans le module `random`, la fonction à deux variables entières `randint` renvoie un entier au hasard compris entre a et b avec a et b compris.

```
from random import randint
print(randint(4, 12))
```

On obtient un entier différent à chaque appel appartenant à $\{4, 5, \ldots, 12\}$.

Écrire une fonction `tiragesDe(n)` qui renvoie une liste de taille 6 dont les valeurs sont les nombres d'occurrences des entiers de 1 à 6 dans n tirages aléatoire d'un dé. Le terme d'indice i de la liste $(0 \leqslant i < 6)$ contiendra le nombre d'occurrences de $i + 1$.

Pa exemple `tiragesDe(60000)` peut renvoyer

`[9912, 9964, 10031, 9949, 10035, 10109]`.

Exercice III.13 — Statistiques

Écrire des fonctions qui calculent la moyenne et la variance des termes d'une liste.

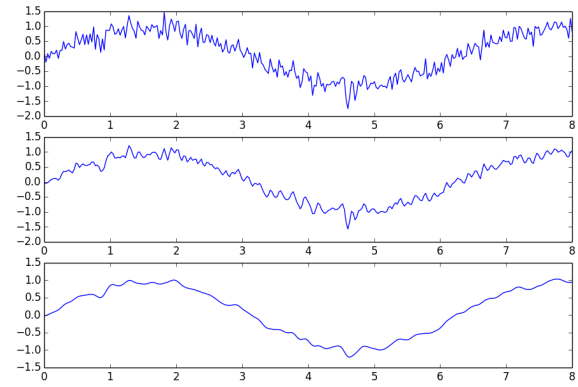
Exercice III.14 — Lissage d'une liste

Lorsque l'on traite des données acquises lors d'une expérimentation, les résultats sont souvent perturbés par des fluctuations, du bruit.

Pour éliminer ce bruit, une technique possible est de lisser les valeurs en les moyennant.
Dans la pratique, si les valeurs de la liste sont $a_0, a_1, \ldots, a_{n-1}$, on crée une liste de taille n et on affecte avec les valeurs b_k définies par

$$\begin{cases} b_0 = \frac{a_0+a_1}{2} \quad b_{n-1} = \frac{a_{n-2}+a_{n-1}}{2} \\ b_k = \frac{a_{k-1}+2a_k+a_{k+1}}{4} \text{ pour } 1 \leqslant k \leqslant n-2 \end{cases}$$

Dans le graphe ci-contre on a calculé (voir ci-dessous) une liste et on a appliqué le lissage puis on a répété 10 fois le lissage pour obtenir la liste représentée en bas.



```
l = [math.sin(i/30)+random.normalvariate(0, 0.1)
                              for i in range(250)]
```

Écrire une fonction `lissage(liste)`

Exercice III.15 — Nombres de Catalan
Les nombres de Catalan sont des entiers qui apparaissent dans de nombreux problèmes de dénombrement. Une de leurs définitions est la définition par récurrence :

$$C_0 = 1 \text{ et } C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} \text{ pour } n \in \mathbb{N}$$

Écrire une fonction `catalan(n)` qui renvoie la liste formée des $n+1$ premiers nombres de Catalan, de C_0 à C_n.

Exercice III.16 — Insertion dans une liste
Écrire une fonction `inserer(x,i,liste)` qui renvoie une nouvelle liste où, à partir de `liste`, on a inséré l'élément `x` à la position `i` et décalé les suivants vers la droite.

Exercice III.17 — Suppression d'une position
Écrire une fonction `supprimer_place(i, liste)` qui renvoie une nouvelle liste où, à partir de `liste`, on a supprimé l'élément d'indice `i`. Les éléments suivants doivent être décalés vers la gauche.

Exercice III.18 — Maximum
Écrire une fonction qui calcule le maximum des termes d'une liste.

Exercice III.19 — Positivité 2
On reprend les questions de l'exercice III.10 mais on demande de cesser les tests des éléments de la liste dès que le résultat est certain.

1. Écrire une fonction `tout_positif(liste)` qui renvoie `True` ou `False` selon que tous les éléments de la liste sont positifs ou non.
2. Écrire une fonction `existe_positif(liste)` qui renvoie `True` s'il existe au moins un élément positif dans la liste et `False` sinon.

Exercice III.20 — Transformation de Cesàro
La transformée de Cesàro [1] d'une suite (u_n) est la suite (v_n) définie par $v_n = \dfrac{1}{n+1}\displaystyle\sum_{k=0}^{n} u_k$.

Écrire une fonction `cesaro(liste)` qui renvoie les n premiers termes de la transformée de Cesàro d'une suite donnée par les éléments de la liste de longueur n.

1. D'après le lemme de Cesàro, si une suite converge vers ℓ, sa transformée de Cesàro converge vers ℓ aussi.

3 Compléments

3.1 Nombres taupins

Dans les temps anciens, chaque taupin avait un numéro secret, son **nombre taupin**.
Les premiers étudiants avaient le numéro 1.
Pour les étudiants suivant le nombre était attribué par le parrain.

- Si le parrain, de numéro n, est carré il attribue le numéro $2n$ à son filleul.
- Si le parrain est cube (de numéro n) il attribue le numéro $3n-1$ à son filleul : un cube sait faire 2 opérations à la fois.

On peut remarquer que tous les entiers ne sont pas attribués : par exemple 7 ne peut pas être de la forme $2n$ ni de la forme $3n-1$.
La question se pose de savoir si un entier est un nombre taupin possible : par exemple 1000 est obtenu par la suite $1 \to 2 \to 5 \to 14 \to 28 \to 56 \to 167 \to 500 \to 1000$.
Pour déterminer si un entier n est un nombre taupin on propose l'algorithme suivant :

1. on crée une liste de booléens, `taupin` de taille $n+1$, initialisée à `False`,
2. on initialise, 1 est un nombre taupin,
3. pour chaque k de 1 à n, on admet que la valeur de `taupin[k]` détermine si k et un nombre taupin ; dans ce cas $2k$ et $3k-1$ sont des nombres taupins, ce que l'on peut indiquer dans le tableau (lorsque le nombre est inférieur à n).
4. `taupin[n]` donne la réponse.

Exercice III.21 — Test de taupinalité
Écrire une fonction `est_taupin(n)` qui renvoie `True` ou `False` selon que n est ou non un nombre taupin.

On aurait aimé pouvoir suivre la généalogie d'un nombre taupin, en calculant le nombre taupin du parrain. malheureusement il peut exister des nombres qui sont obtenus de plusieurs manières. Par exemple
$1 \to 2 \to 4 \to 8 \to 16 \to 32$ et $1 \to 2 \to 4 \to 11 \to 32$.
On peut remarquer que toutes les suites commencent par $1 \to 2$ pour les deux calculs possibles.

Exercice III.22 — Nombre de filières
Écrire une fonction `filieres(n)` qui renvoie le nombre de manières (qui peut être 0) d'obtenir k à partir de 2 avec les transformations $k \mapsto 2k$ et $k \mapsto 3k-1$ pour tout k compris entre 0 et n.

Exercice III.23 — 3 filières
Déterminer le premier nombre taupin qui peut peut l'être avec 3 chemins (il est inférieur à 30000)

3.2 Calcul mental

Exercice III.24 — Fonction

1. Importer la fonction `randint` du module `random`.
2. Importer le module `time` et chercher ce que fait le programme suivant (répondre avant de le tester).

```
def addition1():
    a = randint(1,10)
    b = randint(1,10)
    print(a,'+',b,'=')
    time.sleep(4)
    return a+b
```

3. Construire une fonction `calculMental` qui propose de chercher au hasard une addition de nombres entiers à 3 chiffres ou un produit de nombres entiers à 2 chiffres et laisse un temps raisonnable pour la recherche de la réponse.
4. Construire une fonction `challenge` qui propose 3 lancements successifs de `calculMental`

Exercice III.25 — Entrée-sortie

1. Commenter les instructions suivantes et leur résultats (obtenus dans la console).

```
>>> a=input()
4
>>> type(a)
<class 'str'>
>>> b=int(a)
>>> type(b)
<class 'int'>
>>>
```

2. Modifier `calculMental` en utilisant `input` pour faire entrer la réponse par l'utilisateur, la machine dira si c'est correct ou non ou si le temps est dépassé. Modifier aussi la fonction `challenge` afin qu'elle renvoie le pourcentage de réussite.

3.3 Coefficients binomiaux

On veut calculer les coefficients binomiaux $\binom{n}{p}$ pour $0 \leqslant p \leqslant n \leqslant k$ avec k donné comme paramètre. On attend les résultats sous la forme d'une liste de listes ; par exemple, pour $k = 4$, on veut obtenir `[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]`
Pour créer une liste de listes on peut utiliser les instructions

```
coef = [[]]*(k + 1)
for n in range(k + 1):
    coef[n] = [0]*(n + 1)
```

On peut simplifier en

```
coef = [[0]*(n + 1) for n in range(k + 1)]
```

Exercice III.26 — Triangle de Pascal

Écrire une fonction `triangle(k)` qui calcule les coefficients comme indiqué en n'utilisant que des additions à l'aide de la formule $\binom{n+1}{p+1} = \binom{n}{p} + \binom{n}{p+1}$.

Exercice III.27 — Dernière ligne

Écrire une fonction `coef_bin(n)` qui calcule les coefficients $\binom{n}{p}$ pour $0 \leqslant p \leqslant n$ en n'utilisant qu'une liste simple de taille $n + 1$ et des additions.

4 Solutions

Solution de l'exercice III.1 -

a	f1(a)	f2(a)	f3(a)	f4(a)	f5(a)	f6(a)
20	2	0	2	2	2	2
12	3	3	2	2	2	3
105	3	3	3	3	3	3
7	error	0	0	0	0	0

Il vaut mieux éviter de faire des fonctions qui devraient répondre à plusieurs questions.

Solution de l'exercice III.2 - On suit la définition en commençant par les exceptions

```
def bissextile(annee):
    if annee%400 == 0:
        return True
    elif annee%100 == 0:
        return False
    elif annee%4 == 0
        return True
    else:
        return False
```

On peut comprendre la règle sous la forme : les années bissextile sont celles multiples de 4 mais non multiple de 100 ou les années multiples de 400.

```
def bissextile(annee):
    return (annee%4 == 0 and not(annee%100 == 0)) or (annee
       %400 == 0)
```

Solution de l'exercice III.3 -

```
def jours(mois, annee):
    if mois == 2:
        if bissextile(annee):
            return 29
        else:
            return 28
    elif mois == 4 or mois == 6 or mois == 9 or mois == 11:
        return 30
    else:
        return 31
```

Solution de l'exercice III.4 -

```
def racinesReelles(a, b, c):
    """Entrees : 3 flottants representant
                 le polynome aX**2 + bX + c
       Sortie : la liste des racines reelles"""
    delta = b**2 - 4*a*c
    if delta > 0:
        return [(-b + delta)/2*a, (-b - delta)/(2*a)]
    elif delta == 0:
        return [-b/(2*a)]
    else:
        return []
```

Solution de l'exercice III.5 - Ne pas oublier d'initialiser le produit à 1 et non à 0.

```
from random import random

def approx_pi(n):
    dedans = 0
    for i in range(n):
        x = random()
        y = random()
        if x**2 + y**2 < 1:
            dedans = dedans + 1
    return dedans/n*4
```

Solution de l'exercice III.6 -

```
def approximation(x,n):
      """Entrées : un réel x et un entier positif n
         Sortie  : un couple (a,b) tel que a/b est la
             meilleure
                    approximation de x avec 1<=b<=n"""
    ecart = 1
    for denom in range(1,n+1):
        p = math.floor(x*denom)
        for num in range(p,p+2):
            if abs(x-num/denom) < ecart:
                ecart = abs(x-num/denom)
                approx = (num,denom)
    return approx
```

Solution de l'exercice III.7 -

1. Si on calcule la fonction avec les bonnes coordonnées, la condition `ligne == ligne0 or colonne == colonne0` va être évaluée en `True` donc la fonction va retourner "En vue" au lieu de "Coulé".
2. Il suffit d'inverser les deux premières conditions.

```
def bataille(ligne,colonne):
    if ligne == ligne0 and colonne == colonne0:
        return "Coulé"
    elif ligne == ligne0 or colonne == colonne0:
        return "En vue"
    else :
        return "À l'eau"
```

3. On peut imbriquer les conditions.

```
def bataille(ligne,colonne):
    if ligne == ligne0:
        if colonne == colonne0:
            return "Coulé"
        else:
            return "En vue"
    else:
        if colonne == colonne0:
            return "En vue"
        else:
            return "À l'eau"
```

4. Si les points proches contiennent la diagonale alors ils forment un carré.

```
def bataille2(ligne,colonne):
    if ligne == ligne0 and colonne == colonne0:
        return "Coulé"
    elif     abs(ligne - ligne0) <= 1
         and abs(colonne - colonne0) <= 1:
        return "En vue"
    else:
        return "À l'eau"
```

Sinon il faut une égalité d'une coordonnée et un écart de 1 au plus pour l'autre donc une somme des écarts majorée par 1.

```
def bataille2(ligne,colonne):ff
    if ligne == ligne0 and colonne == colonne0:
        return "Coulé"
    elif  (abs(ligne-ligne0) + abs(colonne-colonne0) <= 1):
        return "En vue"
    else :
        return "À l'eau"
```

Solution de l'exercice III.8 -

```
def pythagoriciens(n):
    nombre = 0
    for a in range(1, n+1):
        for b in range(i, n+1):
            for c in range(j, n+1):
                if a**2 + b**2 == c**2:
                    nombre = nombre + 1
    return nombre
```

Pour $n = 1000$, ce programme est trop lent.
On peut obtenir la valeur avec

```
def pythagoriciens1(n):
    nombre = 0
    for a in range(1, n+1):
        for b in range(a, n+1):
            r = (a**2 + b**2)**0.5
            if r == int(r) and r <= n:
                    nombre = nombre + 1
    return nombre
```

Solution de l'exercice III.9 - Ne pas oublier d'initialiser le produit à 1 et non à 0.

```
def produit(liste):
    """Entrée : une liste
       Sortie : le produit des termes de la liste"""
    prod = 1          # On initialise le produit
    for x in liste:   # Chaque élément de la liste
        prod = prod*x # est multiplié
    return somme
```

Solution de l'exercice III.10 - L'initialisation est différente.

1. Si on cherche à savoir si tous les éléments sont positifs la réponse est `True` jusqu'à ce qu'on trouve un élément négatif

```
def tout_positif(liste):
    """Entrée : une liste de nombre
      Sortie : True si tous les termes sont positifs
               False sinon"""
    resultat = True
    for x in liste:
        resultat = resultat and (x >= 0)
    return resultat
```

2. Ici la réponse est `False` jusqu'à preuve du contraire.

```
def existe_positif(liste):
    """Entrée : une liste de nombre
      Sortie : True si au moins un terme est positif
               False sinon"""
    resultat = False
    for x in liste:
        resultat = resultat or (x >= 0)
    return resultat
```

Solution de l'exercice III.11 -

1. On peut écrire la liste terme-à-terme en lisant la liste initiale à l'envers.

```
def listeInverse(liste):
    """Entrée : une liste
       Sortie : une liste avec les termes inversés"""
    n = len(liste)
    etsil = [0]*n # liste à l'envers :)
      for i in range(n):
          etsil[i] = liste[n - 1- i]
      return etsil
```

2. Ce n'est pas la même question, on modifie la liste "en place".
   On utilise la fonction `echange` de ce chapitre.

```
def retourner(liste):
  """Entrée : une liste
      Sortie : la liste est retournée"""
  n = len(liste)
  for i in range(n//2):
      echange(liste,i,n-1-i)
```

Il ne faut pas faire des échanges du début jusqu'à la fin car sinon on remet en place les éléments retournés. On doit échanger les éléments d'indice 0 à $p-1$ si $n = 2p$ ou $n = 2p+1$ (le milieu reste en place).

Solution de l'exercice III.12 -

```
from random import randint

def tiragesDe(n):
    """Entrée : un entier positif
       Sortie : la liste de la distribution des valeurs
                d'un tirage de n dés"""
    resultat = [0]*6
    for i in range(n):
        k = randint(1, 6)
        resultat[k - 1] += 1
    return resultat
```

Solution de l'exercice III.13 -

```
def moyenne(liste):
    """Entrée : une liste
       Sortie : la moyenne des éléments"""
    n = len(liste)  # Le nombre d'éléments
    som = 0         # On initialise la somme
    for i in range(n): # Chaque terme
        som = som + liste[i] # est ajouté
    return somme/n # La moyenne
```

```
def variance(liste):
    """Entrée : une liste
       Sortie : l'écart-type des éléments de la liste"""
    moy = moyenne(liste)
    n = len(liste)
    som = 0
    for i in range(n):
        som = som + (liste[i] - moy)**2
    return somme/n
```

Avec les raccourcis de python

```
def variance(liste):
    """Entrée : une liste
       Sortie : l'écart-type des éléments de la liste"""
       moy = moyenne(liste)
    return moyenne([(x-moy)**2 for x in liste])
```

Solution de l'exercice III.14 -

```
def lissage(liste):
    n = len(liste)
    lisse = [0]*n
    lisse[0] = (liste[0] + liste[1])/2
    for i in range(1, n-1):
        lisse[i] = (liste[i-1] + 2*liste[i] + liste[i+1])/4
    lisse[n-1] = (liste[n-2] + liste[n-1])/2
    return lisse
```

Solution de l'exercice III.15 -

```
def catalan(n):
    """Entrée : un entier positif
       Sortie : la liste des nombres de catalan de C(0) à C(n)
          """
    res = [0]*(n+1)
    for k in range(n): # On calcule C(k+1)
        somme = 0
        for i in range(k+1): # i de 0 à k : on applique
            somme = somme + res[i]*res[k-i] # la définition
        res[k+1] = somme
    return res
```

Solution de l'exercice III.16 - le plus simple est d'utiliser les extractions.
On rappelle que `liste[:i]+liste[i:]` reconstitue la liste.

```
def inserer(x,i,liste):
    """Entrée : un élément x, un entier i et une liste
                on doit avoir 0 <= i <= len(liste)
       Sortie : la liste avec x inséré à la position i"""
    return liste[:i]+[x]+liste[i:]
```

Solution de l'exercice III.17 -

```
def supprimer_place(i,liste):
    """Entrée : un entier i et une liste, 0 <= i < len(liste)
       Sortie : la liste moins le i-ième élément"""
    return liste[:i]+liste[i+1:]
```

Solution de l'exercice III.18 -

```
def maxListe(liste):
      """Entrée : une liste non vide
         Sortie : la valeur du maximum de la liste"""
  maximum = liste[0] # Un maximum provisoire
  n = len(liste)
  for i in range(n):
      if maximum < liste[i]:
          maximum = liste[i]
  return maximum
```

On compare inutilement le premier terme avec lui-même pour $i = 0$. On pouvait écrire `for i in range(1,n)`, dans le cas $n = 1$ aucune itération n'est calculée.

On pouvait aussi se passer de l'indice :

```
def maxListe(liste):
    """Entrée : une liste non vide
         Sortie : la valeur du maximum de la liste"""
  maximum = liste[0] # Un maximum provisoire
  for x in liste:
      if maximum < x:
          maximum = x
  return maximum
```

Solution de l'exercice III.19 -

1. Le résultat est certain si on a trouvé une valeur négative : dans ce cas la réponse est `False`. Si toute la boucle est parcourue sans sortie c'est que toutes les valeurs sont positives.

```
def tout_positif(liste):
    """Entrée : une liste de nombre
       Sortie : True si tous les termes sont positifs
                False sinon"""
    for x in liste:
        if x < 0:
            return False
    return True
```

2. Ici le cas est symétrique, une valeur positive permet de conclure `True`

```
def existe_positif(liste):
    """Entrée : une liste de nombre
       Sortie : True si au moins un terme est positif
                False sinon"""
    for x in liste:
        if x >= 0:
            return True
    return False
```

Solution de l'exercice III.20 -

```
def cesaro(liste):
    """Entrée : une liste de nombre
       Sortie : la suite transformée de Cesaro"""
    n = len(liste)
    ces = [0]*n
    somme = 0
    for k in range(n):
        somme = somme + liste[k]
        ces[k] = somme/(k+1)
    return ces
```

Solution de l'exercice III.21 -

```
def est_taupin(n):
    """Entrée : une entier positif
       Sortie : True si n est un nombre taupin
                False sinon"""
    taupin = [False]*(n+1)
    taupin[1] = True
    for k in range(1, n+1):
        if taupin[k]:
            if 2*k <= n:
                taupin[2*k] = True
            if 3*k - 1 <= n:
                taupin[3*k - 1] = True
    return taupin[n]
```

Solution de l'exercice III.22 -

```
def filieres(n):
    """Entrée : une entier positif
       Sortie : le nombres de filiation taupines de 2 à n"""
    nombres = [0]*(n+1)
    nombres[1] = 1
    nombres[2] = 1
    for k in range(2, n+1):
        if 2*k <= n:
            nombres[2*k] += nombres[k]
        if 3*k - 1 <= n:
            nombres[3*k - 1] += nombres[k]
    return nombres
```

Solution de l'exercice III.23 -

```
liste = filieres(30000)
for i in range(30000):
    if liste[i] == 3:
        print(i)
```

La valeur calculée est 20480. Voici les 3 chemins pour y arriver, on notera que deux de ces chemins passent par 32 puis sont identiques.

```
[1, 2, 4, 11, 32, 95, 190, 569, 1138, 2276, 6827, 20480],
[1, 2, 4, 8, 16, 32, 95, 190, 569, 1138, 2276, 6827, 20480],
[    1,     2,     5,    10,    20,    40,    80,   160,
   320,   640,  1280,  2560,  5120, 10240, 20480]]
```

Solution de l'exercice III.24 -

```
from random import randint
import time

def calculMental():
    op = randint(1,2)
    if op == 1:
        a= randint(100, 999)
        b= randint(100, 999)
        print("{} + {} = ?".format(a, b))
        time.sleep(6)
        return a + b
    else:
        a = randint(10, 99)
        b = (10, 99)
        print("{} * {} = ?".format(a, b))
        time.sleep(13)
        return a*b

def challenge():
    print(calculMental())
    print(calculMental())
    print(calculMental())
```

Solution de l'exercice III.25 -

```
def calculMentalbis():
    op = randint(1,2)
    if op==1:
        a= randint(100, 999)
        b= randint(100, 999)
        print("{} + {} = ?".format(a, b))
        delai = 6
        resultat = a + b
    else:
        a=random.randint(9,99)
        b=random.randint(9,99)
        print(a,'*',b,'=')
        delai = 13
        resultat = a * b
    t = time.time()
    res = input("Taper votre reponse avant {} secondes".format
       (delai))
    res = int(res)
    dt = time.time()-t
    if res == resultat and dt <= delai:
        print('Bravo')
    elif dt > delai and res == resultat:
        print('Vous avez depasse le temps imparti mais le
           resultat est correct')
    else:
        print("La bonne réponse est {}".format(resultat))
```

Solution de l'exercice III.26 -

```
def triangle(k):
    coef = [[]]*(k + 1)
    for n in range(k + 1):
        coef[n] = [0]*(n + 1)
    coef[0][0] = 1
    for n in range(1, k+1):
        coef[n][0] = 1
        for p in range(1, n):
            coef[n][p] = coef[n-1][p-1] + coef[n-1][p]
        coef[n][n] = 1
    return coef
```

Solution de l'exercice III.27 - L'astuce est de calculer les nouveaux coefficients en partant des derniers ; on ne remplace alors pas un terme qui devrait servir ensuite.

```
def coef_bin(n):
    coef = [1]*(n + 1)
    for m in range(1, n+1):
        for p in range(m-1, 0, -1): #coef[m][m] vaut déjà -1
            coef[p] = coef[p-1] + coef[p]
    return coef
```

TP IV

Jeux de chiffres

Résumé *Dans ce T.P. nous allons manipuler les nombres entiers positifs à partir de leur écriture en base 10.*

1 Outils

Le chiffre des unité d'un nombre entier n est le reste de la division de n par 10, que l'on calcule en python par `n % 10`. Si on veut le chiffre des dizaine, il suffit de diviser n par 10 et de prendre le chiffre des unités ; on devra effectuer la division entière : `n//10`.
Le chiffre de rang p d'un entier n est le p-ième chiffre à partir de la droite dans l'écriture de n : le chiffre de rang 3 de 1478 est 4, son chiffre de rang 5 est 0.

Exercice IV.1
Écrire une fonction `p_ieme_chiffre(n, p)` *qui renvoie le chiffre de rang p dans l'écriture de n.*

Exercice IV.2
En déduire une fonction `chiffres(n, p)` *qui renvoie la liste des chiffres de n* **dans l'ordre**.
Si n a moins de p chiffres ($n < 10^{p-1}$) les premiers termes de la liste seront des 0.
Si n a plus de p chiffres ($n \geqslant 10^p$) les premiers chiffres de n seront oubliés.
`chiffres(458, 3)` doit renvoyer `[4, 5, 8]`
`chiffres(458, 5)` doit renvoyer `[0, 0, 4, 5, 8]`
`chiffres(458, 2)` doit renvoyer `[5, 8]`

Exercice IV.3 — Amélioration facultative
Écrire une fonction `chiffres(n, p)` *qui renvoie la liste des chiffres de n sans utiliser l'exponentiation.*

Bien entendu nous aurons besoin de l'opération inverse : déterminer un nombre à partir de la liste de ses chiffres. Le calcul mathématique est simple : si n est représenté par `chf` de taille p alors $n =$ `chf[p-1]` $+$ `chf[p-2]`$.10 +$ `chf[p-3]`$.100 + \cdots +$ `chf[0]`$.10^{p-1}$.

Exercice IV.4
Écrire une fonction `nombre(chf)` *qui renvoie un nombre à partir de la liste de ses chiffres.*

On aura besoin de trier les chiffres obtenus.
Les algorithmes de tri seront étudiés en seconde année.
On emploiera la méthode `sort` pour trier une liste dans l'ordre croissant.
Pour `l = [3, 1, 4, 1, 5]`, l'instruction `l.sort()` transforme la liste en `l = [1, 1, 3, 4, 5]`.

Un des objectifs de la décomposition est de pouvoir "retourner" un nombre : $7453 \to 3547$.
Une étape intermédiaire est de retourner une liste.

Exercice IV.5
Écrire une fonction `retourner(liste)` *qui renvoie une liste qui a les mêmes éléments que la liste initiale mais écrits dans le sens inverse.*

Pour retourner un nombre il suffit alors d'en calculer la liste des chiffres, de retourner cette liste puis de transformer la liste ainsi obtenue en un nombre.
On retournera un nombre en le considérant comme ayant p chiffres. Par exemple, sur 3 chiffres, 1 est en fait 001 donc le résultat de son retournement est en 100.

Exercice IV.6
Écrire une fonction `miroir(n, p)` *qui renvoie l'entier retourné de n.*

2 Exercices tirés du projet Euler

Exercice IV.7 — Projet Euler 20
Que vaut la somme des chiffres de 100 ! ?
100 ! n'a pas plus que 200 chiffres et Python manie les entiers sans limitation.
On peut remarquer que 145 =!1+!4+!5. Il existe un second entier vérifiant cette propriété d'être la somme des factorielles de ses chiffres (en dehors des cas triviaux 1 =!1 et 2 =!2).

Exercice IV.8 — Projet Euler 34
Déterminer l'entier $n > 145$ qui est la somme des factorielles de ses chiffres.
On pourra admettre qu'il a 5 chiffres.

Exercice IV.9 — Projet Euler 56
Quelle est la somme des chiffres maximale pour les entiers de la forme a^b avec a et b entiers, $a < 100$ et $b < 100$. Ces entiers sont strictement inférieurs à 100^{100} donc ont 200 chiffres au plus.

3 Un tour

"*Pensez à un nombre de 3 chiffres.*
Faites la différence entre le nombre et son retourné (le grand moins le petit).
À partir du nombre retourné que l'on considère comme ayant 3 chiffres faites la somme avec son retourné. Vous obtenez 1089"
Exemples : pour $n = 452$, 254, $|452 - 254| = 198$, $198 + 891 = 1089$.
$n = 100$. $|100 - 001| = 099$, $099 + 990 = 1089$.

Exercice IV.10
Écrire une fonction `tour(n)` *qui, à partir d'un entier compris entre 100 et 999, effectue les opérations ci-dessus*
On utilisera `miroir(k, 3)`.
En fait le résultat n'est pas vrai pour tous les nombres.

Exercice IV.11
Écrire, dans la partie principale les instructions qui permettent d'afficher tous les entiers entre 100 et 999 qui ne donnent pas 1089.
Combien y-a-t-il d'exceptions ? Peut-on les caractériser ?

4 Nombre de Kaprekar

La fonction de Kaprekar consiste à associer à un nombre quelconque n un autre nombre $K(n)$ généré de la façon suivante.

- On forme le nombre n_1 en arrangeant les chiffres du nombre n dans l'ordre croissant
- On forme le nombre n_2 en arrangeant les chiffres de n dans l'ordre décroissant,
- on pose $K(n) = n_2 - n_1$.

On va étudier la suite des itérés par ce calcul **en conservant le même nombre de chiffres le long du calcul**.
Par exemple $K(4582) = 8542 - 2458 = 6084$, si on itère on obtient
$6084 \to 8172 \to 7443 \to 3996 \to 6264 \to 4176 \to 6174 \to 6174 \to \cdots$
$1121 \to 999 \to 8991 = 9990 - 999 \to 8082 \to 8532 \to 6174 \to 6174 \to \cdots$.
Si on teste plusieurs nombres il semble qu'on arrive à 6174 pour un nombre de départ à 4 chiffres.

Exercice IV.12
Écrire une fonction `K` *telle que* `K(n)` *calcule $K(n)$ pour un entier $n \leqslant 9999$ écrit sur 4 chiffres.*

On définit, pour tout entier p, la suite (k_n) par $k_0 = p$ et $k_{n+1} = K(k_n)$.

Exercice IV.13
Prouver que, pour entier p à 4 chiffres à l'exception de 9 nombres particuliers, la suite (k_n) vérifie qu'il existe un entier n_0 tel que $k_n = 6174$ pour $n \geqslant n_0$.

Pour un entier p, sa longueur de Kaprékar est le premier entier n tel que $k_n = 6174$.
Pour les entiers exceptionnels on a $k_n = 0$ pour $n \geqslant 1$, leur longueur de Kaprekar sera 1.
La longueur de Kaprekar est donc le premier entier n tel que $k_n = k_{n+1}$.

Exercice IV.14
Écrire une fonction `longueurK` *qui calcule la longueur de Kaprékar d'un entier à 4 chiffres.*

Exercice IV.15
Quelle la plus grande longueur de Kaprékar pour un entier à 4 chiffres ?

Exercice IV.16
Calculer le nombre d'entiers qui ont une longueur de Kaprékar de i pour $i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$.

Exercice IV.17
Que donne la fonction de Kaprekar pour les nombres à 3 chiffres ?

5 Nombres de Lychrel

Un nombre est un palindrome s'il est égal à son miroir.
On définit une suite (u_n) par son premier terme, u_0, et par la relation de récurrence :
u_{n+1} est la somme de u_n et du miroir de u_n.
Par exemple si on pose $u_0 = 79$ alors $u_1 = 79 + 97 = 176$, $u_2 = 176 + 671 = 847$, $u_3 = 847 + 748 = 1595$, $u_4 = 1595 + 5951 = 7546$, $u_5 = 7546 + 6457 = 14003$ et $u_6 = 14003 + 30041 = 44044$.
On remarque que u_6 est un palindrome.
Si on part de $u_0 = 28$ alors $u_1 = 28 + 82 = 110$, $u_2 = 110 + 11 = 121$ qui est aussi un palindrome.
En fait la suite parvient souvent à un nombre palindrome rapidement : les nombres qui ne semblent pas aboutir à un palindrome sont appelés nombres de **Lychrel**. Il y a 13 nombres de Lychrel candidats inférieurs à 1000, le plus petit étant 196.
On admettra que pour $n \leqslant 1000$, n est un nombre de Lychrel si la suite des u_k pour $0 \leqslant k \leqslant 25$ ne contient pas de palindrome.
On a besoin de retourner un nombre en n'ajoutant pas de 0 en tête, il faut donc calculer le nombre de chiffres d'un entier.

Exercice IV.18
Écrire une fonction `longueur(n)` *qui renvoie le nombre de chiffres de n.*
En déduire une fonction `miroir1(n)` *qui retourne le miroir de n.*

Exercice IV.19
Écrire une fonction `palindrome196(u)` *qui renvoie le premier nombre palindrome de la suite des (u_n) définie ci-dessus avec $u_0 = u$.*
u doit être un entier non nul inférieur ou égal à 195.

Exercice IV.20
Écrire une fonction `longueur196(u)` *qui renvoie le premier entier n tel que u_n est un palindrome où u est la suite définie ci-dessus avec $u_0 = u$.*
u doit être un entier non nul inférieur ou égal à 195.

Exercice IV.21
Déterminer le premier entier entre 1 et 195 pour lequel cette longueur est maximale.

Exercice IV.22
Combien y-a-t-il de nombres de Lychrel inférieurs à 10000 ?
Cette question ressemble à celle du projet Euler 55 mais les nombre de Lychrel demandés ne doivent pas donner un nombre palindrome (égal à son miroir) après la première étape.

6 Solutions

Solution de l'exercice IV.1 -

```
def p_ieme_chiffre(n, p):
    m = n//10**(p-1)
    return m%10
```

Solution de l'exercice IV.2 - Attention : la fonction ci-dessus calcules les chiffres en les numérotant de 1 à p.

```
def chiffres(n, p):
    chf = [0]*p
    for k in range(p):
        chf[k] = p_ieme_chiffre(n, p-k)
    return chf
```

Solution de l'exercice IV.3 -

```
def chiffres(n, p):
    chf = [0]*p
    for k in range(p):
        chf[p-1-k] = n%10
        n = n//10
    return chf
```

Solution de l'exercice IV.4 -

```
def nombre(chf):
    p = len(chf)
    n = 0
    for i in range(p):
        n = n + chf[i]*10**(p-1-i)
    return n
```

Solution de l'exercice IV.5 -

```
def retourner(liste):
    n = len(liste)
    resultat = [0]*n
    for i in range(n):
        j  = n - 1 - i
        resultat[j] = liste[i]
    return resultat
```

On peut aussi utiliser `for j in range(n-1, -1, -1)`.
On pouvait simplifier la fonction en utilisant une extraction.

```
def retourner(liste):
    return liste[ : :-1]
```

Solution de l'exercice IV.6 -

```
def miroir(n, p):
    chf = chiffres(n, p)
```

```
    fhc = retourner(chf)
    return nombre(fhc)
```

Solution de l'exercice IV.7 -

```
def facto(n):
    f = 1
    for i in range(n):
        f = f*(i+1)
    return f

l = chiffres(facto(100), 200)
somme = 0
for x in l:
    somme = somme + x
print("La somme des chiffres de 100! est",somme)
```

Solution de l'exercice IV.8 -

```
fact = [1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880]
for n in range(10000, 100000):
    l = chiffres(n, 5)
    s = 0
    for x in l:
        s = s + fact[x]
    if n ==  s:
        print(n)
        break
```

Solution de l'exercice IV.9 -

```
def sum_c(n):
    l = chiffres(n,201)
    s = 0
    for x in l:
        s = s + x
    return s

max = 1
a_max = 1
b_max = 1
for a in range(100):
    for b in range(101):
        s = sum_c(a**b)
        if s > max:
            print(a, b, s)
            max = s
            a_max = a
            b_max = b
print(a_max, b_max, max)
```

Solution de l'exercice IV.10 -

```
def tour(n):
    n1 = miroir(n, 3)
```

```
    n2 = abs(n -n1)
    n3 = miroir(n2,3)
    return n2 + n3
```

Solution de l'exercice IV.11 -

```
exceptions = 0
for i in range(100,1000):
    if tour(i) != 1089:
        exceptions += 1
        print(i)

print("Il y a",exceptions,"exceptions")
```

Les 90 exceptions sont les entiers égaux à leur miroir.

Solution de l'exercice IV.12 -

```
def K(n):
    l1 = chiffres1(n, 4)
    l1.sort()
    l2 = retourner(l1)
    return nombre(l1) - nombre(l2)
```

Solution de l'exercice IV.13 -

```
for p in range(1000,10000):
    k = p
    k1 = K(p)
    while k != k1:
        k = k1
        k1 = K(k)
    if n != 6174:
        print(i)
```

Solution de l'exercice IV.14 -

```
def longueurK(p):
    k = p
    k1 = K(p)
    long = 0
    while k != k1:
        k = k1
        k1 = K(k)
        long = long + 1
    return long
```

Solution de l'exercice IV.15 -

```
longMax = 0
nb = 6174
for p in range(1000,10000):
    long = longueurK(p)
    if long > longMax:
        longMax = k
        nb = p
```

```
print(iterMax)
print(nb)
```

La longueur 7 est maximale ; par exemple pour $p = 1004$.

Solution de l'exercice IV.16 -

```
histo = [0]*8
for p in range(1000,10000):
    long = longueurK(p)
    histo[long] += 1
```

La longueur 7 est maximale (pour 1980 valeurs de n).

Solution de l'exercice IV.17 - On aboutit à 495 sauf pour les multiples de 111.

Solution de l'exercice IV.18 - La fonction a été vue en cours, on cherche le premier p tel que $n < 10^p$

```
def longueur(n):
    p = 0
    puiss = 1
    while puiss <= n:
        p = p + 1
        puiss = puiss*10
    return p

def miroir1(n):
    p = longueur(n)
    return miroir(n, p)
```

Solution de l'exercice IV.19 -

```
def palindrome196(u):
    while u != miroir1(u):
        u = u + miroir1(u)
    return u
```

Solution de l'exercice IV.20 -

```
def longueur196(u):
    n = 0
    while u != miroir1(u):
        n = n +1
        u = u + miroir1(u)
    return n
```

Solution de l'exercice IV.21 -

```
iterMax = 0
nb = 1
for u in range(1,196):
    n = longueur196(u)
    if n > iterMax:
        iterMax = n
        nb = u
print(nb,"a une longueur196 maximale :",iterMax)
```

On trouve que 89 a une longueur de 24.

Solution de l'exercice IV.22 -

```
nb = 0
for i in range(10000):
    n = i + 1
    for k in range(50):
        if n == miroir1(n):
            n = n + miroir1(n)
        else:
            nb = nb + 1
            break
print(nb)
```

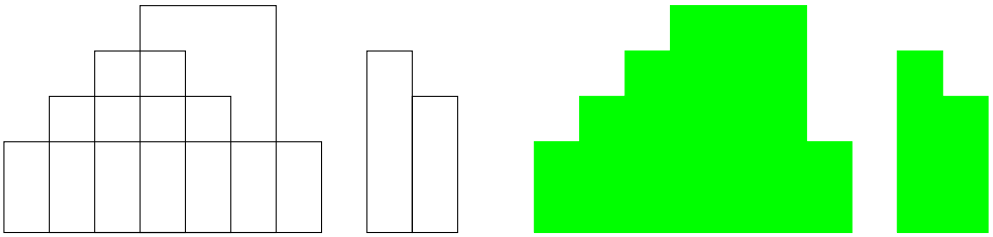
TP V

Lignes d'horizon

Résumé

Ce T.P. est adapté de l'épreuve du concours Polytechnique 2005. Avant 2015, il n'y avait pour enseignement de l'informatique qu'une initiation succincte, correspondant aux premiers mois du cours de première année en informatique commune.

On cherche à calculer la ligne d'horizon engendrée par n beaux bâtiments modernes ($n > 0$), assimilés à des parallélépipèdes verticaux. Pour simplifier, on se place dans l'espace à deux dimensions ; nos bâtiments sont de simples rectangles verticaux ; la ligne d'horizon est représentée par une suite de segments horizontaux formés par les sommets des bâtiments.



Dans cet exemple les 6 immeubles sont représentés par 6 rectangles :

1. un rectangle de hauteur 2 et de largeur 7 entre les abscisses 0 et 7,
2. un rectangle de hauteur 3 et de largeur 4 entre les abscisses 1 et 5,
3. un rectangle de hauteur 4 et de largeur 2 entre les abscisses 2 et 4,
4. un rectangle de hauteur 5 et de largeur 3 entre les abscisses 3 et 6,
5. un rectangle de hauteur 4 et de largeur 1 entre les abscisses 8 et 9,
6. un rectangle de hauteur 3 et de largeur 1 entre les abscisses 9 et 10.

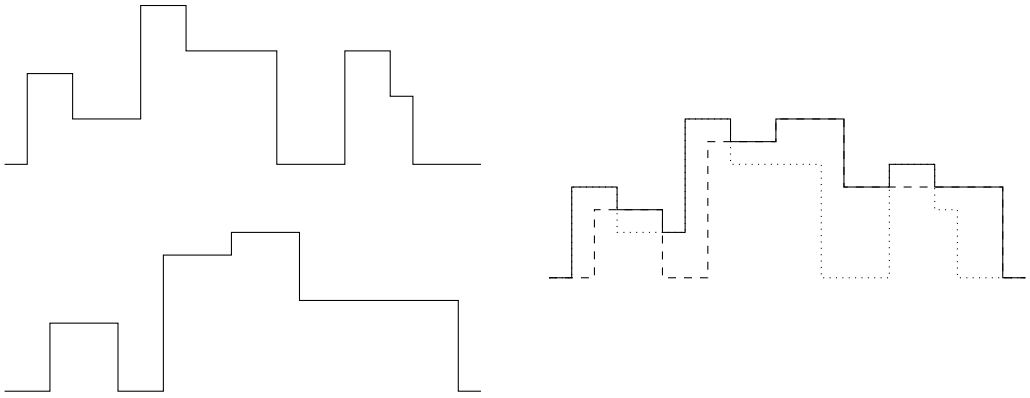
On suppose que les abscisses sont positives.

Les rectangles seront donnés par des triplets (g, h, d) où

1. g est l'abscisse de gauche,
2. d est l'abscisse de droite,
3. h est la hauteur. On rappelle que les rectangles ont leur base sur l'axe des abscisses.

On codera l'ensemble des rectangles par 3 listes `G`, `D` et `H` de même longueur (le nombre de rectangles) telles que le rectangle d'indice i est défini par $g =$`G[i]`, $d =$`D[i]` et $h =$`H[i]`.
Dans l'exemple ci-dessus, les listes peuvent valoir respectivement
`G = [0, 1, 2, 3, 8, 9]`, `D = [7, 5, 4, 6, 9, 10]` et `H = [2, 3, 4, 5, 4, 3]`.

Un objectif de ce problème est de fusionner deux lignes d'horizon comme illustré ci-dessous



1 Coordonnées entières

On suppose dans cette partie que les abscisses des rectangles sont entières (et positives) comme dans l'exemple ci-dessus.
La ligne d'horizon sera donc définie par la liste `Hor` des valeurs des hauteurs des segments $[k;k+1]$.
Dans l'exemple ci-dessus, cette liste est `[2, 3, 4, 5, 5, 5, 2, 0, 4, 3]`.
On a besoin de déterminer la largeur construite pour déterminer la longueur de la liste.

Exercice V.1
Écrire une fonction `largeur(D)` *qui renvoie l'abscisse de droite maximale à partie de la liste des abscisses de droite d'un ensemble de rectangles.*
La liste définissant la ligne d'horizon sera donc de longueur `M = largeur(D)`.

L'exercice suivant n'est utile que comme étape préliminaire à la question suivante, la fonction ne sera pas utilisée dans la question V.3.

Exercice V.2
Écrire une fonction `un_immeuble(M, g, h, d)` *qui calcule (et renvoie) la liste représentant la ligne d'horizon de longueur M quand il y a un seul immeuble défini par g, d et h (avec $0 \leqslant g < d \leqslant M$).*

Exercice V.3
Écrire une fonction `horizon(G, D, H)` *qui calcule (et renvoie) la ligne d'horizon à partir des listes qui caractérisent les rectangles.*

Exercice V.4
Écrire une fonction `fusion1(hor1, hor2` *qui détermine la ligne d'horizon obtenue à partir de la fusion des deux lignes d'horizon* `hor1` *et* `hor2` *supposées de même longueur.*

Exercice V.5
En utilisant `fusion1` *et* `un_immeuble` *déterminer une nouvelle fonction qui calcule la ligne d'horizon à partir des listes qui caractérisent les rectangles, comme à la question V.3.*
Quel est son inconvénient ?

Exercice V.6
Écrire une fonction `fusion(hor1, hor2)` *qui détermine la ligne d'horizon obtenue à partir de la fusion des deux lignes d'horizon* `hor1` *et* `hor2` *sans supposer qu'elles sont de même longueur.*

2 Coordonnées réelles

Si les coordonnées sont réelles, il n'est plus possible de définir la ligne d'horizon comme ci-dessus. Même dans le cas de coordonnées entières, lorsque les tailles des immeubles sont grandes par rapport au nombre d'immeubles, la méthode précédente utilise des listes de taille importantes.
On détermine une autre représentation de la ligne d'horizon sous la forme d'une liste `hor` de longueur impaire $2p+1$ contenant une succession d'abscisses et de hauteurs : `hor[2k]` et `hor[2k+2]` déterminent les abscisses de début et de fin d'un segment de ligne d'horizon à hauteur `hor[2k+1]`.
Ainsi, dans l'exemple ci-dessus, la ligne d'horizon peut être représentée par le tableau
`[0, 2, 1, 3, 2, 4, 3, 5, 6, 2, 7, 0, 8, 4, 9, 3, 10]`.

On impose dans la suite les conditions suivantes :

1. `hor[2k] <= hor[2k+2]` pour tout $k < p$: la ligne d'horizon est parcourue de la gauche vers la droite,
2. la première abscisse, `hor[0]`, sera toujours nulle,
   la derniere, `hor[0]`, vaudra toujours 10.

Les questions seront illustrées par le résultat attendu pour la ligne d'horizon

```
hor0 = [0.0, 0, 1.6, 3, 3.7, 0, 5.8, 4, 7.3, 6, 9.2, 2, 10]
```

On a choisit des abscisses réelles mais on garde des hauteurs entières.

2.1 Calculs

Dans cette partie, on supposera que les abscisses d'une ligne d'horizon sont distinctes.

Exercice V.7
Écrire une fonction `maximum(hor)` *qui calcule la hauteur maximale atteint par une ligne d'horizon.*
`maximum(hor0)` doit renvoyer 6

Exercice V.8
Écrire une fonction `bati(hor)` *qui calcule la largeur totale des bâtiments une ligne d'horizon, c'est la somme des largeurs des intervalles pour lesquels la hauteur est non nulle.*
`bati(hor0)` renvoie `6.300000000000001`, tout calcul flottant est faux.

Exercice V.9
Écrire une fonction `surface(hor)` *qui calcule la surface totale des bâtiments une ligne d'horizon, c'est la somme des aires des rectangles déterminés par la ligne d'horizon.*
`surface(hor0)` renvoie `25.299999999999997`.

2.2 Fusion

Pour calculer la fusion de deux lignes d'horizon, on peut remarquer qu'une ligne d'horizon de longueur n vérifie que n est impair et que le nombre d'intervalles est $p = \frac{n-1}{2}$, qu'on peut calculer par `n//2`. Il y a ainsi $p-1$ valeurs intermédiaires.
On en déduit que la fusion de deux lignes d'horizon de longueurs n_1 et n_2, de nombres d'intervalles respectifs p_1 et p_2, devra avoir $(p_1-1)+(p_2-1)$ valeurs intermédiaires donc p_1+p_2-1 intervalles : sa longueur sera $2(p_1+p_2-1)+1 = n_1+n_2-3$.
On supposera, lors de la fusion de deux lignes d'horizon que les valeurs des abscisses des deux lignes d'horizon sont distinctes à l'exception des abscisses 0 et 10, communes par définition.

Exercice V.10
Écrire une fonction `fusion(hor1, hor2)` *qui calcule la ligne d'horizon fusion de* `hor1` *et* `hor2`.
Pour `hor1 = [0, 0, 2.5, 3, 6.3, 0, 10]` et `hor2 = [0, 0, 4.7, 5, 8.6, 0, 10]`
`fusion(hor1, hor2)` renvoie `[0, 0, 2.5, 3, 4.7, 5, 6.3, 5, 8.6, 0, 10]`.

2.3 Forme canonique

La représentation `hor` d'une ligne d'horizon est **canonique** si elle vérifie les deux conditions supplémentaires suivantes dans lesquelles $n = 2p + 1$ est la longueur de `hor`.

1. `hor[2k] < hor[2k+2]` pour tout $k < p - 1$, il n'y a pas d'intervalles de largeur nulle.
2. `hor[2k-1] != hor[2k+1]` pour tout $k < p - 1$, un segment ne peut pas être découpé en plusieurs segments de même hauteur.

On peut prouver qu'une ligne d'horizon admet une unique représentation canonique.

Ces deux conditions peuvent être contredites lors de la fusion de deux lignes d'horizon, même si celles-ci sont sous forme canonique.

1. Si une abscisse de `hor1` et une abscisse de `hor2` (en dehors de 0 et de 10) sont égales, la fusion va sans doute créer un intervalle de largeur nulle.
2. L'exemple de la question V.10 montre qu'un grand immeuble peut engendrer dplusieurs segments de même hauteur.

Exercice V.11
Écrire une fonction `canonique(hor)` *qui calcule la représentation canonique de la ligne d'horizon représentée par* `hor`
Pour `hor3 = [0, 0, 1.2, 5, 1.2, 0, 1.8, 4, 1.8, 3, 2.4, 0, 10]`
`canonique(hor3)` renvoie `[0, 0, 1.8, 3, 2.4, 0, 10]`

3 Solutions

Solution de l'exercice V.1 -

```
def largeur(D):
    M = D[0]
    n = len(D)
    for i in range(1, n):
        if D[i] > M:
            M = D[i]
    return M
```

Solution de l'exercice V.2 -

```
def un_immeuble(M, g, h, d):
    hor = [0]*M
    for k in range(g, d):
        hor[k] = h
    return hor
```

Solution de l'exercice V.3 -

```
def horizon(G, D, H):
    n = len(G)
    M = largeur(D)
    hor = [0]*M
    for i in range(n):
        g = G[i]
        d = D[i]
        h = H[i]
        for k in range(g, d):
            hor[k] = max(hor[k], h)
    return hor
```

Solution de l'exercice V.4 -

```
def fusion1(hor1, hor2):
    M = len(hor1)
    hor = [0]*M
    for i in range(M):
        hor[i] = max(hor1[i], hor2[i])
    return hor
```

Solution de l'exercice V.5 -

```
def horizon_autre(G, D, H):
    n = len(G)
    M = largeur(D)
    hor = [0]*M
    for i in range(n):
        hor = fusion(hor, un_immeuble(M, G[i], D[i], H[i])
    return hor
```

La fonction crée beaucoup de listes, cela occupe de la mémoire.

Solution de l'exercice V.6 -

```
def fusion(hor1, hor2):
    M1 = len(hor1)
    M2 = len(hor2)
    m = min(M1, M2)
    M = max(M1, M2)
    hor = [0]*M
    for i in range(m):
        hor[i] = max(hor1[i], hor2[i])
    if M1 < M2:
        for i in range(m, M):
            hor[i] = hor2[i]
    else:
        for i in range(m, M):
            hor[i] = hor1[i]
    return hor
```

On peut aussi utiliser la fonction précédente.

```
def fusion(hor1, hor2):
    M1 = len(hor1)
    M2 = len(hor2)
    m = min(M1, M2)
    h1 = hor1[ :m]
    h2 = hor2[ :m]
    return fusion1(h1, h2) + hor1[m: ] + hor2[m: ]
```

Solution de l'exercice V.7 - Les hauteurs sont données par `hor[2p+1]` avec $2p+1 < n$,

```
def maximum(hor):
    n = len(hor)
    max = hor[1]
    for k in range(1, n, 2):
        if hor[k] > max:
            max = hor[k]
    return max
```

Solution de l'exercice V.8 - Les intervalles sont $[$`hor[2p]`$;$`hor[2p+2]`$]$ avec $2p+2 < n$,

```
def bati(hor):
    n = len(hor)
    largeur = 0
    for k in range(0, n-2, 2):
        if hor[k+1] > 0:
            largeur = largeur + hor[k+2] - hor[k]
    return largeur
```

Solution de l'exercice V.9 -
Les surfaces sont $($`hor[2p+2]` $-$ `hor[2p]`$]).$`hor[2p+1]` avec $2p+2 < n$,

```
def surface(hor):
    n = len(hor)
    surf = 0
    for k in range(0, n-2, 2):
        surf = surf + (hor[k+2] - hor[k])*hor[k+1]
    return surf
```

Solution de l'exercice V.10 -
Une idée possible est de remplir la ligne fusion pas-à-pas en suivant les abscisses suivantes, aux indices k_1 et k_2, de chaque ligne et en choisissant la plus proche. La hauteur sera le maximum des hauteurs des intervalles précédant les indices k_1 et k_2.

```
def fusion(hor1, hor2):
    n = len(hor1) + len(hor2) - 3
    fus = [0]*(n)
    k1 = 2
    k2 = 2
    for i in range(1, n, 2):
        h = max(hor1[k1-1], hor2[k2-1])
        fus[i] = h
        if hor1[k1] < hor2[k2]:
            fus[i+1] = hor1[k1]
            k1 = k1 + 2
        else:
            fus[i+1] = hor2[k2]
            k2 = k2 + 2
    return fus
```

Solution de l'exercice V.11 - L'idée est de supprimer les abscisses qui sont égales à la précédentes ou qui ne changent pas la hauteur.

- On ajoute à chaque étape une hauteur, notée `h` et une abscisse `hor[p]` avec p pair, c'est la boucle principale, de la ligne 7 à la ligne 14. `hor[p+2]` est alors différente de `hor[p]` et la hauteur `hor[p+1]` est différente de h (sinon on aurai choisi $p+2$).
  1. La nouvelle hauteur à ajouter sera donc `hor[p+1]` (lignes 8 et 9).
  2. Pour trouver l'abscisse suivante on incrémente l'indice 2 par 2 tant qu'on a la même hauteur ou que l'abscisse ne change pas (lignes 10 à 12).
  3. On peut alors ajouter l'abscisse (ligne 13).
  4. On passe alors à la position suivante (ligne 14).
- Pour déterminer la première hauteur donc la première abscisse, on doit avancer. Il faut donc éliminer les abscisses égales à l'abscisse initiale (lignes 4 à 6)

```
1  def canonique(hor):
2      n = len(hor)
3      can = [hor[0]]
4      p = 2
5      while p < n and hor[p] == hor[0]:
6          p = p + 2
7      while p < n:
8          h = hor[p-1]
9          can.append(h)
10         while p + 2 < n and (   hor[p+2] == hor[p]
11                              or hor[p+1] == h):
12             p = p + 2
13         can.append(hor[p])
14         p = p + 2
15     return can
```

TP V

Retours

Résumé *Ce T.P. reprend des exercices des T.P. 2 à 4, non faits lors des précédents T.P.*

1 Suite de Muller (1993)

La suite de Muller est définie par $u_0 = \frac{5}{2}$, $u_1 = \frac{17}{5}$ et $u_{n+2} = 30 - \frac{129}{u_{n+1}} + \frac{100}{u_n.u_{n+1}}$ pour $n \geqslant 0$.

Exercice V.1 — Calcul des termes
Écrire une fonction `muller(n)` qui calcule le terme u_n de cette suite.
On pourra s'inspirer de la fonction qui calcule les termes de la suite de Fibonacci.
Calculer les 50 premières valeurs de u_n. Quelle semble être la limite ?

Exercice V.2 — Étude mathématique
Prouver, par récurrence, que $u_n = \frac{1+4^{n+1}}{1+4^n}$. En déduire la limite de u_n.
On voit apparaître les limites du calcul avec les nombres flottants ; on y reviendra.

Les valeurs de la suites sont des fractions rationnelles. Pour éviter les erreurs d'arrondi on peut utiliser le module **`fractions`**. Il ne contient qu'une fonction, `Fraction`, qui permet de définir une variable sous forme de fraction.

```
from fractions import Fraction
a = Fraction(2, 6)
print(a)
print(float(a))
----------------------
1/3
0.3333333333333333
```

On remarquera que la fraction est simplifiée.
On peut effectuer les opérations usuelles avec les fractions.

Exercice V.3 — Usage des fractions
Écrire une fonction `muller_frac(n)` qui calcule le terme u_n de la suite de Muller en utilisant les fractions. Calculer les 50 premières valeurs de u_n ; quelle semble être la limite ?

2 Polynômes de degré 3

Le but de ces exercices est de déterminer les 3 racines (complexes) d'un polynôme de degré 3 : $P(X) = X^3 + aX^2 + +bX + c$. Même dans le cas d'un polynôme à coefficients réels et dont les 3 racines sont réelles ont doit calculer des valeurs intermédiaires complexes.
Python permet le calcul algébrique des nombres complexes sous la forme `a+bj` pour $a + ib$.

- i est noté `1j`
- $2 + 2i$ est noté `2+2j`
- Lorsque les coefficients doivent être calculés, il sera nécessaire d'utiliser le constructeur `complex` qui prend deux réels en arguments.

```
>>> z = complex(1+3, 2*3)
 4 + 6j
```

- Les parties réelle et imaginaire sont accessibles par les méthodes `real` et `imag`.

```
>>> z = 2+3j
>>> z.real
2.0
>>> z.imag
3.0
```

- On peut calculer la somme, le produit, le quotient de deux nombres complexes.
- On peut calculer **une** racine n-ième : `(-2+2j)**(1/3)` renvoie `(1.0000000000000002+1j)` ; on notera l'erreur d'arrondi.
- On peut définir un nombre complexe à partir de son module r et de son argument φ à l'aide de la fonction `rect(r, phi)` du module `cmath`.

```
import cmath as c
-------------------------
>>> c.rect(2,c.pi/3)
(1.0000000000000002+1.7320508075688772j)
```

On commence par le cas simple $X^3 - u$.

Exercice V.4
Écrire une fonction `racines0(u)` *qui reçoit un complexe u et renvoie ses 3 racines dans $\mathbb{C}$.*

Dans le cas du polynôme $P(X) = X^3 + pX + q$ on démontre (exercice de mathématique) que si on pose $X = a + b$ en imposant $ab = -\frac{p}{3}$ alors $a^3 + b^3 = -q$. Comme on a aussi $a^3b^3 = -\frac{p^3}{27}$ on voit que a^3 et b^3 sont les racines de $Q(X) = X^2 + qX - \dfrac{p^3}{27}$.
On obtient alors 2 valeurs pour a^3 ce qui donne 6 valeurs pour a, cependant on démontre (autre exercice mathématique) que

- si on choisit une des racines de Q, α,
- si on attribue à a prend successivement les 3 racines de α comme valeur,
- et si la valeur de b est déterminée par $ab = -\frac{p}{3}$,

on obtient alors les trois racines de P en calculant $a + b$.

Exercice V.5
Écrire une fonction `racines1(p, q)` *qui reçoit les deux paramètres d'un polynôme $X^3 + pX + q$ et qui en renvoie les 3 racines.*
On supposera p non nul

Exercice V.6
Dans le cas général, $P(X) = X^3 + aX^2 + +bX + c$, montrer que u est racine de P si et seulement si $u + \frac{a}{3}$ est racine d'un polynôme de la forme $P_1(X) = X^3 + pX + q$ dont on déterminera les coefficients p et q en fonction de a, b et c.
En déduire une fonction `racines(a, b, c)` *qui reçoit les trois paramètres d'un polynôme $X^3 + aX^2 + bX + c$ et qui en renvoie les 3 racines.*
On traitera à part le cas $p = 0$ en utilisant une instruction conditionnelle.

On suppose maintenant que les coefficients sont réels.

Exercice V.7
En raison des erreurs d'arrondi les racines réelles peuvent être calculées avec un partie imaginaires non nulle : on conviendra qu'un complexe est réel si sa partie imaginaire est inférieure à 10^{-10}.
Modifier `racines(a, b, c)` *en* `racinesR(a, b, c)` *de telle manière que les racines retournées soient sous forme réelle si ce sont des réels.*
On peut montrer (en étudiant les variations de $f(x) = x^3 + px + q$) que le polynôme $P_1(X) = X^3 + pX + q$ admet 3 racines réelles si et seulement si le discriminant de $Q(X) = X^2 + qX - \frac{p^3}{27}$ est négatif.

3 Nombres taupins

Dans les temps anciens, chaque taupin avait un numéro secret, son **nombre taupin**.
Les premiers étudiants avaient le numéro 1.
Pour les étudiants suivant le nombre était attribué par le parrain.

- Si le parrain, de numéro n, est carré il attribue le numéro $2n$ à son filleul.
- Si le parrain est cube (de numéro n) il attribue le numéro $3n - 1$ à son filleul : un cube sait faire 2 opérations à la fois.

On peut remarquer que tous les entiers ne sont pas attribués : par exemple 7 ne peut pas être de la forme $2n$ ni de la forme $3n - 1$.
La question se pose de savoir si un entier est un nombre taupin possible : par exemple 1000 est obtenu par la suite $1 \to 2 \to 5 \to 14 \to 28 \to 56 \to 167 \to 500 \to 1000$.
Pour déterminer si un entier n est un nombre taupin on propose l'algorithme suivant :

1. on crée une liste de booléens, `taupin` de taille $n + 1$, initialisée à `False`,
2. on initialise, 1 est un nombre taupin,
3. pour chaque k de 1 à n, on admet que la valeur de `taupin[k]` détermine si k et un nombre taupin ; dans ce cas $2k$ et $3k - 1$ sont des nombres taupins, ce que l'on peut indiquer dans le tableau (lorsque le nombre est inférieur à n).
4. `taupin[n]` donne la réponse.

Exercice V.8 — Test de taupinalité
Écrire une fonction `est_taupin(n)` qui renvoie `True` ou `False` selon que n est ou non un nombre taupin.

On aurait aimé pouvoir suivre la généalogie d'un nombre taupin, en calculant le nombre taupin du parrain. malheureusement il peut exister des nombres qui sont obtenus de plusieurs manières. Par exemple
$1 \to 2 \to 4 \to 8 \to 16 \to 32$ et $1 \to 2 \to 4 \to 11 \to 32$.
On peut remarquer que toutes les suites commencent par $1 \to 2$ pour les deux calculs possibles.

Exercice V.9 — Nombre de filières
Écrire une fonction `filieres(n)` qui renvoie le nombre de manières (qui peut être 0) d'obtenir k à partir de 2 avec les transformations $k \mapsto 2k$ et $k \mapsto 3k - 1$ pour tout k compris entre 0 et n.

Exercice V.10 — 3 filières
Déterminer le premier nombre taupin qui peut peut l'être avec 3 chemins (il est inférieur à 30000)

4 Nombre de Kaprekar

La fonction de Kaprekar consiste à associer à un nombre quelconque n un autre nombre $K(n)$ généré de la façon suivante.

- On forme le nombre n_1 en arrangeant les chiffres du nombre n dans l'ordre croissant
- On forme le nombre n_2 en arrangeant les chiffres de n dans l'ordre décroissant,
- on pose $K(n) = n_2 - n_1$.

On va étudier la suite des itérés par ce calcul **en conservant le même nombre de chiffres le long du calcul**.
Par exemple $K(4582) = 8542 - 2458 = 6084$, si on itère on obtient
$6084 \to 8172 \to 7443 \to 3996 \to 6264 \to 4176 \to 6174 \to 6174 \to \cdots$
$1121 \to 999 \to 8991 = 9990 - 999 \to 8082 \to 8532 \to 6174 \to 6174 \to \cdots$.
Si on teste plusieurs nombres il semble qu'on arrive à 6174 pour un nombre de départ à 4 chiffres.

Exercice V.11
Écrire une fonction `K` *telle que* `K(n)` *calcule $K(n)$ pour un entier $n \leqslant 9999$ écrit sur 4 chiffres.*

On définit, pour tout entier p, la suite (k_n) par $k_0 = p$ et $k_{n+1} = K(k_n)$.

Exercice V.12
Prouver que, pour entier p à 4 chiffres à l'exception de 9 nombres particuliers, la suite (k_n) vérifie qu'il existe un entier n_0 tel que $k_n = 6174$ pour $n \geqslant n_0$.

Pour un entier p, sa longueur de Kaprékar est le premier entier n tel que $k_n = 6174$.
Pour les entiers exceptionnels on a $k_n = 0$ pour $n \geqslant 1$, leur longueur de Kaprekar sera 1.
La longueur de Kaprekar est donc le premier entier n tel que $k_n = k_{n+1}$.

Exercice V.13
Écrire une fonction `longueurK` *qui calcule la longueur de Kaprékar d'un entier à 4 chiffres.*

Exercice V.14
Quelle la plus grande longueur de Kaprékar pour un entier à 4 chiffres ?

Exercice V.15
Calculer le nombre d'entiers qui ont une longueur de Kaprékar de i pour $i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$.

Exercice V.16
Que donne la fonction de Kaprekar pour les nombres à 3 chiffres ?

5 Solutions

Solution de l'exercice V.1 -

```
def muller(n):
    u = 5/2
    u_suivant = 17/5
    for i in range(n):
        u_avant = u
        u = u_suivant
        u_suivant = 30 - 129/u + 100/u/u_avant
    return u
```

Dans la partie principale.

```
for k in range(50):
    print("u{} vaut {}".format(k, muller(k)))
```

La suite semble converger vers 25 alors qu'elle semblait s'approcher de 4 au début.

Solution de l'exercice V.2 - $\frac{1+4^1}{1+4^0} = \frac{5}{2} = u_0$, $\frac{1+4^2}{1+4^1} = \frac{17}{5} = u_1$,
si $u_n = \frac{1+4^{n+1}}{1+4^n}$ et $u_{n+1} = \frac{1+4^{n+2}}{1+4^{n+1}}$ alors
$u_{n+2} = 30 - \frac{129(1+4^{n+1})}{1+4^{n+2}} + \frac{100(1+4^n)}{1+4^{n+2}} = \frac{30(1+4^{n+2}) - 129(1+4^{n+1}) + 100(1+4^n)}{1+4^{n+2}}$
$u_{n+2} = \frac{(30-129+100) + 4^n(30.16 - 129.4 + 100)}{1+4^{n+2}} = \frac{1+4^n.64}{1+4^{n+2}} = \frac{1+4^{n+3}}{1+4^{n+2}}$
On peut écrire $u_n = \frac{4^{-n}+4}{4^{-n}+1}$ donc la limite est 4.

Solution de l'exercice V.3 -

```
def muller_frac(n):
    u = Fraction(5, 2)
    u_suivant = Fraction(17, 5)
    for i in range(n):
        u_avant = u
        u = u_suivant
        u_suivant = 30 - 129/u + 100/u/u_avant
    return u

for k in range(50):
    print("u{} vaut {}, {} sous forme décimale".format(k,
       muller_frac(k), float(muller_frac(k))))
```

Solution de l'exercice V.4 -

```
def racines3(u):
    """Entrées : un complexe
       Sortie  : les 3 racines de u"""
    v1 = u**(1/3)
    w = c.rect(1,2*c.pi/3)
    v2 = v1*w
    v3 = v2*w # ou v1/w
    return v1, v2, v3
```

Solution de l'exercice V.5 -

```
def racines1(p, q):
    """Entrées : les coefficients p et q de X**3+pX+q
       Sortie  : les 3 racines du polynôme"""
    delta  = q**2 + 4*p**3/27
    alpha = (-q + delta**(1/2))/2
    a1, a2, a3 = racines0(alpha)
    b1 = -p/3/a1
    b2 = -p/3/a2
    b3 = -p/3/a3
    return a1+b1, a2+b2, a3+b3
```

Solution de l'exercice V.6 - $\left(u+\frac{a}{3}\right)^3 = u^3 + a.u^2 + \frac{a^2}{3}u + \frac{a^3}{27}$ or $u^3 + a.u^2 = -b.u - c$ car u est racine.
Ainsi $\left(u+\frac{a}{3}\right)^3 = \left(\frac{a^2}{3} - b\right)u + \frac{a^3}{27} - c = \left(\frac{a^2}{3} - b\right)\left(u+\frac{a}{3}\right) + \frac{a^3}{27} - c - \frac{a^3}{9} + \frac{ab}{3}$.
$p = b - \frac{a^2}{3}$ et $q = c - \frac{ab}{3} + \frac{2a^3}{27}-$.

```
def racines(a, b, c):
    """Entrées : les coefficients de X**3+aX**2+bX+c
       Sortie  : les 3 racines du polynôme"""
    p = b - a**2/3
    q = c - a*b/3 + 2*a**3/27
    if p == 0:
        u1, u2, u3 = racines0(-q)
    else:
        u1, u2, u3 = racines1(p, q)
    return u1 - a/3, u2 - a/3, u3 - a/3
```

Solution de l'exercice V.7 -

```
def racinesR(a, b, c):
    """Entrées : les coefficients de X**3+aX**2+bX+c
       Sortie  : les 3 racines du polynôme"""
    p = b - a**2/3
    q = c - a*b/3 + 2*a**3/27
    u1, u2, u3 = racines1(p,q)
    v1 = u1 - a/3
    v2 = u2 - a/3
    v3 = u3 - a/3
    if abs(v1.imag) < 1e-10:
        v1 = v1.real
    if abs(v2.imag) < 1e-10:
        v2 = v2.real
    if abs(v3.imag) < 1e-10:
        v3 = v3.real
    return v1, v2, v3
```

Solution de l'exercice V.8 -

```
def est_taupin(n):
    """Entrée : une entier positif
       Sortie : True si n est un nombre taupin
                False sinon"""
    taupin = [False]*(n+1)
    taupin[1] = True
    for k in range(1, n+1):
        if taupin[k]:
            if 2*k <= n:
                taupin[2*k] = True
            if 3*k - 1 <= n:
                taupin[3*k - 1] = True
    return taupin[n]
```

Solution de l'exercice V.9 -

```
def filieres(n):
    """Entrée : une entier positif
       Sortie : le nombres de filiation taupines de 2 à n"""
    nombres = [0]*(n+1)
    nombres[1] = 1
    nombres[2] = 1
    for k in range(2, n+1):
        if 2*k <= n:
            nombres[2*k] += nombres[k]
        if 3*k - 1 <= n:
            nombres[3*k - 1] += nombres[k]
    return nombres
```

Solution de l'exercice V.10 -

```
liste = filieres(30000)
for i in range(30000):
    if liste[i] == 3:
        print(i)
```

La valeur calculée est 20480. Voici les 3 chemins pour y arriver, on notera que deux de ces chemins passent par 32 puis sont identiques.

```
[1, 2, 4, 11, 32, 95, 190, 569, 1138, 2276, 6827, 20480],
[1, 2, 4, 8, 16, 32, 95, 190, 569, 1138, 2276, 6827, 20480],
[    1,     2,     5,    10,    20,    40,    80,   160,
   320,   640,  1280,  2560,  5120, 10240, 20480]]
```

Solution de l'exercice V.11 -

```
def K(n):
    l1 = chiffres1(n, 4)
    l1.sort()
    l2 = retourner(l1)
    return nombre(l1) - nombre(l2)
```

Solution de l'exercice V.12 -

```
for p in range(1000,10000):
    k = p
    k1 = K(p)
    while k != k1:
        k = k1
        k1 = K(k)
    if n != 6174:
        print(i)
```

Solution de l'exercice V.13 -

```
def longueurK(p):
    k = p
    k1 = K(p)
    long = 0
    while k != k1:
        k = k1
        k1 = K(k)
        long = long + 1
    return long
```

Solution de l'exercice V.14 -

```
longMax = 0
nb = 6174
for p in range(1000,10000):
    long = longueurK(p)
    if long > longMax:
        longMax = k
        nb = p
print(iterMax)
print(nb)
```

La longueur 7 est maximale ; par exemple pour $p = 1004$.

Solution de l'exercice V.15 -

```
histo = [0]*8
for p in range(1000,10000):
    long = longueurK(p)
    histo[long] += 1
```

La longueur 7 est maximale (pour 1980 valeurs de n).

Solution de l'exercice V.16 - On aboutit à 495 sauf pour les multiples de 111.

TP VI

Polynômes

1 Premières fonctions

On choisit de représenter le polynôme $P = \sum_{k=0}^{n} a_k X^k$ où les a_k sont des réels
sous la forme d'une liste $[a_0, a_1, \cdots a_n]$ d'objets de type `float`.

Exercice VI.1 — Degré
Écrire une fonction `degre(P)` *qui renvoie le degré d'un polynôme donné en paramètre.*
Par convention, elle renverra -1 *pour le polynôme nul.*

Exercice VI.2 — Réduction
Écrire une fonction `reduire` *qui renvoie la représentation de longueur minimale (de longueur égale au degré plus un) d'un polynôme.*
Le polynôme nul devra donc être représenté par la liste vide `[]`.

Dans la suite on pourra supposer que les polynômes sont réduits.
Toute fonction qui renvoie un polynôme devra renvoyer un polynôme réduit.

Exercice VI.3 — Derivation
Écrire une fonction `derive(P)` *qui renvoie le polynôme dérivé formel (sous forme de liste aussi) d'un polynôme donné en paramètre.*

Exercice VI.4 — Évaluation
Écrire une fonction `calculNaif(x, P)` *qui renvoie la valeur en un réel* x *d'un polynôme donné* P.
Cette fonction utilisera `x**k` *pour calculer* x^k.
Pour $P = \sum_{k=0}^{n} a_k X^k$ et x réel, on observe que

$$P(x) = \sum_{k=0}^{n} a_k x^k = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + x(a_n)) \cdots)))$$

Cette façon de représenter $P(x)$ est l'algorithme de Hörner. Elle peut aussi s'écrire

$$P(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + x(a_n + x.0)) \cdots)))$$

Exercice VI.5
Écrire (sur un papier !) la forme de Hörner de $P(x) = 1 + 2x + 3x^2 + x^4 - x^5$.

Exercice VI.6
Écrire une fonction `horner(x, P)` *qui calcule* $P(x)$ *en utilisant l'algorithme de Hörner.*

Les termes calculés dans la méthode de Hörner apparaissent dans un autre contexte.
Si P est un polynôme alors α est une racine de $P - P(\alpha)$ donc il existe un polynôme Q [1] tel que $P - P(\alpha) = (X - \alpha).Q$, $P = (X - \alpha).Q + P(\alpha)$.
Si on pose $P = \sum_{k=0}^{d} a_k X^k$ et $Q = \sum_{k=0}^{d-1} b_k X^k$ on obtient

$$a_d = b_{d-1},\ a_k = b_{k-1} - \alpha b_k \text{ pour } 1 \leqslant k \leqslant d-1 \text{ et } a_0 = -\alpha b_0 + P(\alpha)$$

On remarque que les valeurs successives $b_{d-1} = a_d$ et $b_{k-1} = a_k + \alpha b_k$ correspondent aux valeurs successives calculées dans la méthode de Hörner.

Exercice VI.7
Écrire, en utilisant ces relations, une fonction `divEuc(P, a)` *qui renvoie une liste représentant le polynôme* $Q = \dfrac{P}{X - a}$ *lorsque* a *est une racine de* P.

Remarque : si on dérive $P = (X - \alpha)Q + \beta$ alors on a $P'(\alpha) = Q(\alpha)$: l'algorithme ci-dessus permet donc de calculer $P'(\alpha)$ sans utiliser la dérivée de P.

2 Recherche d'une racine

2.1 Plus grande racine

On se place dans cette partie dans le cas d'un polynôme $P = \sum_{k=0}^{d} a_k X^k$ scindé à racines simples réelles qui admet, de plus, une unique racine de valeur absolue maximale. Si on note $r_1, r_2, \ldots, r_d$ ses racines on a, par exemple, $|r_k| < |r_d|$ pour tout k appartenant à $\{1, 2, \ldots, d-1\}$.
On définit alors une suite $(u_n)_{n\in\mathbb{N}}$ en choisissant u_0, u_1, …, u_{d-1} puis en calculant, par récurrence,

$$u_{n+d} = \frac{-1}{a_d} \sum_{k=0}^{d-1} a_k u_{n+k}$$

On démontre alors qu'on a $u_n = \sum_{k=1}^{d} A_k r_k^n$ avec A_1, A_2, …, A_d constantes réelles.
Si les valeurs initiales sont bien choisies on a $A_d \neq 0$ donc la suite $\left(\dfrac{u_{n+1}}{u_n}\right)$ converge vers r_d.

2.2 Exemple

On prend $P = X^3 - 4X^2 - 7X + 10$; P vérifie les hypothèses ci-dessus (un graphe rapide permet de s'en convaincre).
On définit alors une suite $(u_n)_{n\in\mathbb{N}}$ par

$$u_0 = 1,\ u_1 = 2,\ u_2 = 1 \text{ et } \forall n > 2, u_{n+3} = 4u_{n+2} + 7u_{n+1} - 10u_n$$

Exercice VI.8
Écrire une fonctions `termesU(n)` *qui renvoie la listes des valeurs de* u_k *pour* $0 \leqslant k \leqslant n$.
On pourra supposer qu'on a $n \geqslant 2$.

1. C'est le résultat obtenu en effectuant la division euclidienne de P par $X - \alpha$.

Exercice VI.9
Écrire une fonction `quotients(n)` *qui renvoie la liste des quotients* $\frac{u_{k+1}}{u_k}$ *pour* k *variant de* 0 *à* $n-1$.

Exercice VI.10
Représenter graphiquement les valeurs de la suite des quotients (`plt.plot(Y)`*) pour* $n = 15$.
Que semble être la limite ?

Exercice VI.11
On note $q_k = \frac{u_{k+1}}{u_k}$, *représenter graphiquement les valeurs* $\left(\frac{q_{k+1}-5}{q_k-5}\right)$ *pour* $0 \leqslant k \leqslant 13$.
Quel semble être son comportement ?
Que peut-on en déduire ?

2.3 Cas général

Si 1 n'est pas une racine de P, on pourra choisir 1 pour les valeurs initiales de la suite $(u_n)_{n\in\mathbb{N}}$.

Exercice VI.12
Écrire une fonction `racine` *qui calcule la plus grande racine (en valeur absolue) d'un polynôme* P *vérifiant les hypothèses ci-dessus et n'admettant pas 1 pour racine.*

2.4 Méthode de Laguerre

Il existe de nombreux algorithmes de recherche de valeur approchée d'une racine d'un polynôme. Nous allons en présenter un, particulièrement rapide.
On suppose que P est un polynôme réel ayant au moins une racine réelle a.
On note d le degré de P et H la fonction définie par

$$H(x) = (d-1)^2 P'(x)^2 - d(d-1)P(x)P''(x)$$

On définit alors la suite des valeurs (x_n) en choisissant x_0 réel puis

$$\forall k \geqslant 0,\ x_{k+1} = x_k - \frac{d.P(x_k)}{P'(x_k) + \varepsilon\sqrt{H(x_k)}}$$

où ε est choisi dans $\{-1, 1\}$ de même signe que $P'(x_k)$.
Comme $\sqrt{H(x_k)}$ est positif, ε a pour effet de maximiser la valeur absolue du dénominateur.
Le principal inconvénient de cette méthode est le calcul des valeurs de la dérivée seconde.
Par contre, en théorie, elle est très rapide.

Exercice VI.13
Écrire une fonction `H(x, P)` *qui prend en paramètre* x *et* P *et qui donne* $H(x)$.

Exercice VI.14
Écrire une fonction `maxV(x, P)` *qui prend en paramètre* x *et* P *et qui donne, parmi les deux valeurs* $P'(x) + \sqrt{H(x)}$ *et* $P'(x) - \sqrt{H(x)}$, *celle qui a la plus grande valeur absolue.*

Exercice VI.15
Écrire une fonction `algo(P, x0, n)` *qui renvoie* x_n *pour le polynôme* P, *la valeur initiale* x_0 *et l'entier naturel* n.
Tester `algo` *pour* $P = X(X-1)(X+2)(X-5) = X^4 - 4X^3 - 7X^2 + 10X$ *et différents choix de* n *et de graine* x_0.

On admet que plus $\dfrac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}$ est petit, plus la vitesse de convergence est grande.

Exercice VI.16
Récrire `algo` *pour obtenir* `algoV` *qui renvoie en plus la valeur de* $\dfrac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}$ *qui devrait nous donner une idée de la vitesse de convergence.*
Tester `algoV` *pour* $P = X^4 - 4X^3 - 7X^2 + 10X$ *puis pour* $Q = XP$.
Observer que la vitesse de convergence est plus lente quand on approche une racine multiple.

2.5 Autres racines

Pour itérer l'algorithme, si on a trouvé une racine α pour un polynôme P, on peut réutiliser la même méthode appliquée au polynôme $Q = \dfrac{P}{(X - \alpha)}$.
On a calculé le quotient dans l'exercice VI.7.

Exercice VI.17
Écrire une fonction qui calcule les racines d'un polynôme P.
On peut supposer que toutes les racines de P *sont réelles et de valeurs absolues distinctes.*
Appliquer à $X^4 - 10X^2 + 4X + 8$.

3 Compléments

3.1 Amplification des erreurs

On définit une suite récurrente $(u_n)_{n\in\mathbb{N}}$ par $u_0 = \frac{1}{12}$ et $\forall n \in \mathbb{N}, u_{n+1} = 13u_n - 1$.
Une récurrence immédiate montre que la suite est constante égale à $\frac{1}{12}$.

Exercice VI.18
Écrire une fonction `suite1(n)` *qui prend un entier* n *en paramètre et qui renvoie la valeur de* u_n.
Expérimenter pour différentes valeurs de n. *Expliquer.*

On définit une suite récurrente $(u_n)_{n\in\mathbb{N}}$ par

$$u_0 = -18,\ u_1 = 22 \text{ et } \forall n \in \mathbb{N},\ u_{n+2} = \frac{10}{3}u_{n+1} - u_n - 56$$

On peut prouver qu'il existe λ et μ tels que $u_n = \lambda\left(\frac{1}{3}\right)^n + \mu 3^n + 42$ pour tout n . Les conditions initiales imposent $\mu = 0$ et $\lambda = -60$ donc $\lim\limits_{n\to+\infty} u_n = 42$.

Exercice VI.19
Écrire une fonction `suite1(n)` *qui renvoie la valeur de* u_n.
Quelle semble être la limite de la suite ?
Proposer une explication

3.2 Racines complexes

Exercice VI.20
Reprendre la méthode d'approximation d'une racine pour un polynôme à racine complexes.
On aura alors parfois $H(x_k)$ *négatif et on devra en tenir compte et remplacer* $\sqrt{H(x_k)}$ *par une des racines complexes de* $H(x_k)$.

4 Solutions

Solution de l'exercice VI.1 -

```
def degre(P):
    """Entrée : une liste qui représente un polynome
       Sortie : le degré du polynôme, -1 pour le polynome nul
          """
    deg = -1
    for k in range(len(P)):
        if P[k] != 0:
            deg = k
    return deg
```

Solution de l'exercice VI.2 - Si d est le degré, le polynôme réduit est celui où on ne garde que les termes d'indices 0 à d.

```
def reduire(P):
    """Entrée : une liste qui représente un polynome
       Sortie : une liste qui représente le même polynôme
                et dont le dernier terme est non nul"""
    d = degre(P)
    return P[0:d+1] # On doit garder P[d]
```

Solution de l'exercice VI.3 -

```
def derive(P):
    """Entrée : une liste qui représente un polynome
       Sortie : une liste qui représente la dérivée"""
    n = len(P)
    deri=[]
    for k in range(1, n):
        deri.append(k*P[k])
    return reduire(deri)
```

Solution de l'exercice VI.4 -

```
def calculNaif(x,  P):
    """Entrée : un nombre
                et une liste représentant un polynome
       Sortie : la valeur de P(x)"""
    n = len(P)
    val = 0
    for k in range(n):
        val = val + P[k]*x**(k)
    return val
```

Solution de l'exercice VI.5 -
$1 + x(2 + x(3 + x(0 + x(1 - x))))$

Solution de l'exercice VI.6 -

```
def horner(x, P):
    """Entrée : un nombre
                et une liste représentant un polynome
       Sortie : la valeur de P(x)"""
    n = len(P)
    res = 0
    for k in range(1, n+1):
        res= x*res + P[-k]
    return res
```

Solution de l'exercice VI.7 - On remarque que, si on pose $b_n = 0$, on a $b_{k-1} = a_k + \alpha b_k$ pour tout k

```
def divEuc(P, a):
    """Entrée : un nombre
                et une liste représentant un polynome
       Requis : P(a) = 0
       Sortie : une liste représentant P/(X-a)"""
    n = len(P)
    Q = [0]*(n-1)
    b = 0
    for k in range(n - 2, -1, -1):
        b= a*b + P[k+1]
        Q[k] = b
    return Q
```

Solution de l'exercice VI.8 -

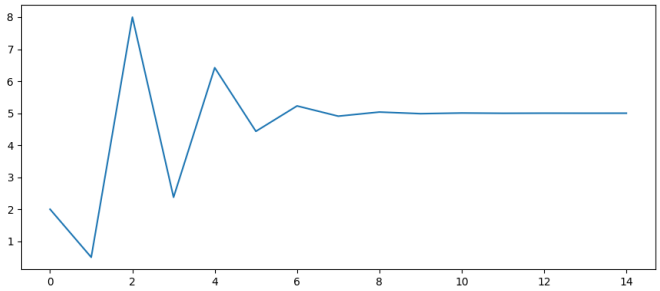
```
def termesU(n):
    """Entrée : un entier positif
       Sortie : la liste des n+1 premiers termes
                de la suite u"""
     U = [1, 2, 1]
     for k in range(3, n+1):
         a = U[k-3]
         b = U[k-2]
         c = U[k-1]
         u = 4*c + 7*b -10*a
         U.append(u)
    return U
```

Solution de l'exercice VI.9 -

```
def quotients(n):
    """Entrée : un entier positif
       Sortie : la liste des n premiers termes
                de la suite u(k+1)/u(k)"""
    U = termesU(n)
    Q = []
    for k in range(n):
        q = U[k+1]/U[k]
        Q.append(q)
    return Q
```

Solution de l'exercice VI.10 -

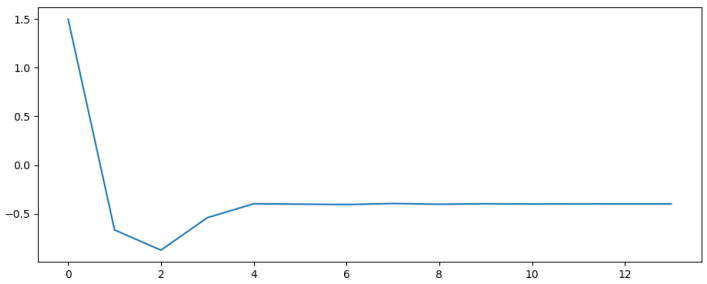
```
plt.plot(quotients(15))
plt.show()
```



La suite semble converger vers 5.

Solution de l'exercice VI.11 -

```
Q = quotients(15) # quotients pour 0 <= k < 15
T = []
for i in range(14):
    t = (Q[i+1] - 5)/(Q[i] - 5)
    T. append(t)
plt.plot(T)
plt.show()
```



On observe une convergence vers une limite environ égale à -0.4.
La convergence est géométrique ; le caractère négatif signifie que la suite passe alternativement au dessus et en dessous de 5.

Solution de l'exercice VI.12 -

```
def racines(P):
    """Entrée : une liste représentant un polynome
       Requis : le polynome est scindé sur R, P(1) non nul
       Sortie : la plus grande racine"""
    n = len(P) - 1
    coef = []
    for i in range(n):
        c = -P[i]/P[-1]
        coef.append(c)
    U = [1]*(n)
    for k in range(n, 100):
        u = 0
        for i in range(n):
            u = u + U[k-n+i]*coef[i]
        U.append(u)
    return U[-1] /U[-2]
```

Solution de l'exercice VI.13 -

```
def H(x, P):
    dP = derive(P)
    ddP = derive(dP)
    d = degre(P)
    Px = horner(x, P)
    dPx = horner(x, dP)
    ddPx = horner(x, ddP)
    res = (d-1)**2*dPx**2 - d*(d-1)*Px*ddPx
    return res
```

Solution de l'exercice VI.14 -

```
def MaxV(x, P):
    dP = derive(P)
    A = horner(x, dP)
    B = H(x, P)**0.5
    if A > 0:
        return A + B
    else:
        return A - B
```

Solution de l'exercice VI.15 -

```
def algo(P, x0, n):
    d = degre(P)
    x = x0
    for i in range(n):
        x = x - d*horner(x, P)/MaxV(x, P)
    return x
```

Solution de l'exercice VI.16 - On doit avoir $n \geqslant 2$.

```
def algoV(P, x0, n):
    d = degre(P)
    x_2 = x0
    x_1 = x0 - d*horner(x0, p)/MaxV(x0, P)
    x = x_1 - d*horner(x_1, p)/MaxV(x_1, P)
    for i in range(n-2):
        x_2 = x_1
        x_1 = x
        x = x - d*horner(x, P)/MaxV(x, P)
    return x, (x - x_1)/(x_1 - x_2)
```

Solution de l'exercice VI.17 -

```
def racines(P):
    """Entrée : une liste représentant un polynome
       Requis : le polynome est scindé sur R
       Sortie : la liste des racines"""
    R = []
    while horner(1, P) == 0:
        R.append(1)
        P = divEuc(P, 1)
    while len(P) >  1:
        a = racine(P)
        R.append(a)
        P = divEuc(P, a)
    return R
```

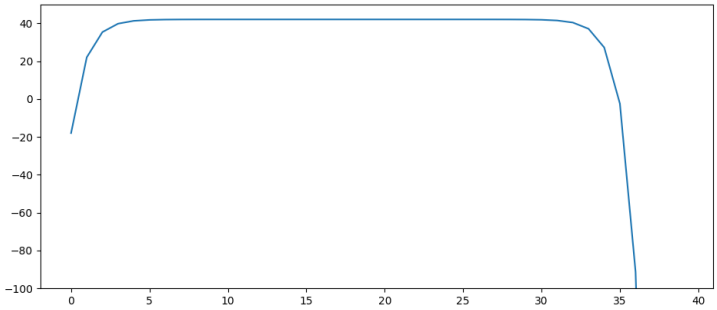
Solution de l'exercice VI.18 -

```
def suite1(n):
    u = 1/12
    for i in range(n):
        u = 13*u-1
    return u
```

(u_n) ne semble pas converger vers $\frac{1}{12}$.
Si $u_k = \frac{1}{12} + \varepsilon$, alors $u_{k+1} = \frac{1}{12} + 13.\varepsilon$ donc les u_k s'écartent de $\frac{1}{12}$ quand on n'a pas une valeur exacte de $\frac{1}{12}$. Or cette valeur n'existe pas dans la représentation des réels : elle est approchée, d'où la divergence.

Solution de l'exercice VI.19 -

```
def suite2(n):
    u = -18
    u_next = 22
    for i in range(n):
        u_new = 10*u_next/3 - u - 56
        u = u_next
        u_next = u_new
    return u
```



La suite se rapproche de 42 puis tend vers $-\infty$.
Les raisons sont les mêmes qu'à l'exercice ci-dessus.

Solution de l'exercice VI.20 - Il suffit de changer `MaxV`

```
def MaxV(x, P):
    dP = derive(P)
    A = horner(x, dP)
    B = H(x, P)**0.5
    if abs(A+B) > abs(A-B):
        return A + B
    else:
        return A - B
```

TP VII

La force avec nous

Nous allons utiliser la méthode la plus directe pour résoudre certains problèmes : on énumère tous les cas possibles. On parle de recherche exhaustive ou d'algorithme en force brute.

Ce type d'approche peut intervenir dans plusieurs cas :

- *on peut avoir besoin de calculer toutes les valeurs possibles, par exemple le produit de deux matrices demande de calculer tous les coefficients,*
- *on ne connaît pas de manière simple de déterminer les cas qui répondent à un critère, on teste alors pour chaque cas s'il vérifie la condition,*
- *on ne connaît pas de manière simple de déterminer un extremum, on doit alors comparer chaque cas avec un extremum provisoire.*

Souvent cette méthode demandera d'énumérer des objets combinatoires :

- *tous les choix possibles dans une suite de décisions,*
- *toutes les parties d'un ensemble,*
- *toutes les permutation d'un ensemble (bijections de E vers lui-même),*
- *toutes les parties à p éléments d'un ensemble,*
- *toutes les applications d'un ensemble vers un autre ...*

Nous allons étudier quelques exemples qui correspondent aux trois premiers cas.
Un outil utile pour décrire toutes les suites finies de décision est d'utiliser la décomposition en base 2 des nombres qui permettent d'énumérer toutes les suites de 0 et 1. On leur fera correspondre toutes les suites de choix possibles : oui/non, dedans/dehors (pour les sous-ensembles) ...

1 Écriture binaire

Exercice VII.1
Écrire une fonction Python `binaire(n,p)` *qui renvoie la liste* $[a_{p-1}, a_{p-2}, \ldots, a_1, a_0]$ *telle que* $n = \sum_{k=0}^{p-1} a_k 2^k$ *pour* $n < 2^p$.
`binaire(50,8)` devra renvoyer `[0, 0, 1, 1, 0, 0, 1, 0]`.

Exercice VII.2
Écrire une fonction Python `entier(liste)` *qui renvoie l'entier dont la représentation en base 2 est la liste fournie en paramètre :*
pour `liste = [`a_{p-1}`,`a_{p-2}`,...,`a_1`,` a_0`]` *la fonction renvoie* $\sum_{k=0}^{p-1} a_k 2^k$.
On pourra écrira une fonction qui n'effectue que p multiplications et p additions pour une liste de taille p.
`entier([0, 0, 1, 0, 1, 0, 1, 0]` devra renvoyer `42`.

2 Le virelangue de Gaston Lagaffe

Dans une page célèbre Gaston emploie une comptine pour calmer le fils de sa voisine. Nous allons essayer de la reproduire.
En papouasie il y a des habitants :
- certains sont papous, d'autres ne le sont pas,
- certains sont papas, pas d'autres,
- les habitants peuvent avoir (ou pas) des poux
- ces poux peuvent être papas.

On veut donc écrire les phrases du type
"En papouasie, il y a des papous"
"En papouasie, il y a des pas papous papas"
"En papouasie, il y a des papous papas à poux papas"
"En papouasie, il y a des pas papous pas papas pas à poux pas papas"

Il y a donc jun certain nombre de caractéristiques (ici de 1 à 4) qui peuvent être chacune niées en les faisant précéder par `pas`.
On suppose données les variables globales

```
intro = "En papouasie, il y a des "
listeMots = ["papous ", "papas ", "à poux ","papas "]
```

On peux ajouter des caractéristiques : époux, poupins, pompiers ...

Exercice VII.3
Écrire une fonction `phrase(condition)` *où* `condition` *est une liste de p termes (avec* $p \leqslant 4$*) valant 0 ou 1 et qui renvoie la phrase de la comptine avec p caractéristiques. La caractéristique d'indice* `i` *est niée ou non selon que* `condition[i]` *vaut 1 ou 0.*
`phrase([0, 1, 1])` devra renvoyer *"En papouasie il y a des papous pas papas pas à poux".*

Exercice VII.4
Écrire une fonction `tout(p)` *qui écrit toutes les phrases possibles avec p caractéristiques.*
p doit être un entier compris entre 1 et la longueur de la liste `listeMots`.
`tout(2)` doit écrire à l'écran

```
En papouasie, il y a des papous papas
En papouasie, il y a des papous pas papas
En papouasie, il y a des pas papous papas
En papouasie, il y a des pas papous pas papas}
```

Exercice VII.5
Écrire les instructions dans la partie principale qui permettent d'écrire la comptine sous la forme

```
En papouasie, il y a des papous
En papouasie, il y a des pas papous
En papouasie, il y a des papous papas
...
En papouasie, il y a des pas papous pas papas pas à poux pas
   papas}
```

3 Les héritiers

Un riche propriétaire meurt.
Dans son testament il stipule que ses deux enfants doivent se partager les différents biens immobilier sans les revendre. Il faut donc partager les maisons de telle manière que la différence des valeurs soit la plus petite possible.
Par exemple si les valeurs sont 17, 36, 45, 46, 51 et 61 de total 256 le partage pourrait être $17 + 51 + 61 = 129$ pour l'un et $36 + 45 + 46 = 127$ pour l'autre.
Pour 17, 36, 45, 46, 51 et 93 il y a un partage équitable avec $17 + 36 + 45 + 46 = 144 = 51 + 93$.

Exercice VII.6
Écrire une fonction `totalPartiel(valeurs, choix)` *où* `valeurs` *est une liste d'entiers et* `choix` *est une liste de même longueur de termes valant 0 ou 1 et qui renvoie la somme des termes de la liste des valeurs pour les indices* `i` *tels que* `choix[i] = 1`.
`totalPartiel([12, 15, 21],[0, 1, 1])` devra renvoyer `15 + 21 = 36`.

Exercice VII.7
En déduire une fonction `partage(valeurs)` *où* `valeurs` *est une liste d'entiers qui renvoie la valeur la plus proche par défaut de la moitié de la somme totale des valeurs ainsi qu'une liste des choix qui permet de l'atteindre.*
`partage([12, 15, 21])` devra renvoyer `21, [0, 0, 1]`.
On évitera d'essayer la fonction avec une liste de taille supérieure à 20.

Exercice VII.8 — Facultatif et plus difficile
Proposer une solution pour un héritage à partager en 3.

4 Sur la route

Le groupe de musique **Green Metal** veut organiser une tournée.
Elle partira de Paris, visitera les 9 plus grosses agglomérations en France avant de se terminer par un dernier concert à Paris.
Les villes seront données dans une liste avec leur latitude et leur longitude (en degrés).

```
villes0  = [["Paris"     , 48.856614, 2.352222],
            ["Lyon"      , 45.764043, 4.835659],
            ["Marseille" , 43.296482, 5.369780],
            ["Lille"     , 50.629250, 3.057256],
            ["Nice"      , 43.710173, 7.261953],
            ["Toulouse"  , 43.604652, 1.444209],
            ["Bordeaux"  , 44.837789,-0.579180],
            ["Nantes"    , 47.218371,-1.553621],
            ["Toulon"    , 43.124228, 5.928000],
            ["Douai-Lens", 50.420087, 2.947280]]
```

Le nom de la ville d'indice `i` est défini par `villes0[i][0]` sa latitude et sa longitude sont les résultats respectifs de `villes0[i][1]` et `villes0[i][2]`.

Comme son nom l'indique, le groupe est soucieux de la planète et il veut donc que la longueur de la tournée soit la plus courte possible.

- La distance selon l'axe nord-sud est calculée par la différence des latitudes (en radians) multipliée par le rayon de la terre qu'on estime à 6371 km : $\Delta_y = (\text{lat}_B - \text{lat}_A).R$.
- La distance selon l'axe est-ouest est approchée par la différence des longitudes (en radians) multipliée par le rayon de la terre le tout multiplié par le cosinus de la latitude moyenne : $\Delta_x = (\text{long}_B - \text{long}_A).R.\cos\left(\dfrac{\text{lat}_B + \text{lat}_A}{2}\right)$.
- La distance sera calculée à l'aide du théorème de Pythagore : $d = \sqrt{\Delta_x^2 + \Delta_y^2}$.

Exercice VII.9
Écrire une fonction `distance(i, j, villes)` *qui renvoie la distance entre les villes d'indices* `i` *et* `j` *dans la liste* `villes`.
`distance(0, 3, villes0)` devra renvoyer `203.51...`

Une tournée sera définie par la liste des villes hors de Paris dans l'ordre de passage. Par exemple la tournée Paris, Bordeaux, Douai-Lens, Lille, Lyon, Marseille, Nantes, Nice, Toulon, Toulouse, Paris sera représentée par `[6, 9, 3, 1, 2, 7, 4, 8, 5]`.

Exercice VII.10
Écrire une fonction `tour(liste, villes)` *qui renvoie la distance totale parcourue par la tournée déterminée par* `liste`.
`tour([1, 2, 3, 4, 5, 6, 7, 8, 9], villes0)` devra renvoyer `5054.59...`

Toutes les tournées seront définies par les permutations de cette dernière liste.
Pour calculer les permutations d'une liste on peut utiliser le module `itertools`.
La boucle `for perm in itertools.permutations(liste):` permet de visiter toutes les permutations possibles de la liste.

```
>>> for x in itertools.permutations([1,5,5]):
...     print(x, end = " ")
(1, 5, 5) (1, 5, 5) (5, 1, 5) (5, 5, 1) (5, 1, 5) (5, 5, 1)
```

Il faut remarquer que `perm` est alors un tuple et non une liste, si on veut utiliser une liste on peut le convertir par `list(perm)`.

Exercice VII.11
Calculer la tournée minimale et afficher la liste des villes dans l'ordre.
Même pour une liste si courte le temps de calcul est assez long.

5 Complément : énumérer les permutations

Nous allons ici étudier un algorithme connu depuis longtemps [1] qui permet de calculer la permutation suivante dans l'ordre croissant.
L'ordre est celui qui énumère les permutations de `[1, 2, 3, 4]` en
`[1, 2, 3, 4]`, `[1, 2, 4, 3]`, `[1, 3, 2, 4]`, `[1, 3, 4, 2]`, `[1, 4, 2, 3]`,
`[1, 4, 3, 2]`, `[2, 1, 3, 4]`, `[2, 1, 4, 3]`, `[2, 3, 1, 4]`, `[2, 3, 4, 1]`,
`[2, 4, 1, 3]`, `[2, 4, 3, 1]`, `[3, 1, 2, 4]`, `[3, 1, 4, 2]`, `[3, 2, 1, 4]`,
`[3, 2, 4, 1]`, `[3, 4, 1, 2]`, `[3, 4, 2, 1]`, `[4, 1, 2, 3]`, `[4, 1, 3, 2]`,
`[4, 2, 1, 3]`, `[4, 2, 3, 1]`, `[4, 3, 1, 2]`, `[4, 3, 2, 1]`

Pour en comprendre le fonctionnement on peut considérer la permutation

`[2, 1, 6, 5, 8, 7, 4, 3]`

et en chercher le successeur.
On cherche une permutation qui garde le maximum de termes initiaux.
Les 4 derniers termes forment une suite décroissante donc on ne peut les ré-ordonner pour obtenir une suite plus petite.
On va donc déterminer le successeur de `[5, 8, 7, 4, 3]`.
On doit augmenter le 5 et la plus petite valeur possible est 7 : en échangeant on aboutit à `[7, 8, 5, 4, 3]`. On remarque que les 4 derniers termes sont toujours dans l'ordre décroissant ; il suffit alors de les "retourner" pour trouver la plus petite permutation possible qui succède à la permutation initiale, c'est-à-dire le successeur :

`[2, 1, 6, 7, 3, 4, 5, 8]`

L'algorithme pour déterminer l'élément suivant d'une liste `perm` peut s'énoncer ainsi.

1. n est la longueur de la liste.
2. On cherche le dernier indice j tel que `perm[j] < perm[j+1]`.
3. S'il n'existe pas on renvoie la liste vide (il n'y a pas de successeur).
4. S'il existe on a `perm[j] < perm[j+1] >= perm[j+2] >= ... >= perm[n-1]`.
   On cherche le dernier indice $k > j$ tel que `perm[j] < perm[k]`.
5. On échange les termes d'indices k et j dans `perm`.
6. On retourne la liste `perm` entre $j+1$ et $n-1$.

Exercice VII.12
Écrire une fonction `suivant(liste)` *qui implémente cet algorithme.*
On retournera une nouvelle liste sans modifier la liste passée en paramètre.

1. La première trace écrite date du XIV-ième siècle, en Inde

6 Solutions

Solution de l'exercice VII.1 -

```
def binaire(n, p):
    """Entrée : deux entiers
       Requis : 0 <= n < 2**p
       Sortie : la liste des p coefficients de l'écriture en
          base 2
                de n commençant le bit de poids fort"""
    bin = [0]*p
    for i in range(p):
        bin[p-1-i] = n%2
        n = n//2
    return bin
```

Solution de l'exercice VII.2 -

```
def entier(liste):
    """Entrée : une liste de 0 et 1
       Sortie : l'entier  représenté en base 2 par la liste,
                la représentation commence par le bit de poids
                    fort"""
    p = len(liste)
    n = 0
    for i in range(p):
        n = 2*n + liste[i]
    return n
```

Solution de l'exercice VII.3 -

```
def phrase(condition):
    """Entrée : une liste de 0 et 1, de longueur 4 au plus
       Sortie : la phrase de la comptine correspondante"""
    ch = intro
    p = len(condition)
    for i in range(p):
        if condition[i] == 1:
            ch = ch + "pas "
        ch = ch + listeMots[i]
    return ch
```

Solution de l'exercice VII.4 -

```
def tout(p)
    N = 2**p
    for n in range(N):
        print(phrase(binaire(n, p))
```

Solution de l'exercice VII.5 -

```
for p in range(1,5):
    tout(p)
```

Solution de l'exercice VII.6 -

```
def totalPartiel(valeurs,choix):
    """Entrée : deux listes de même longueur
       Sortie : le produit scalaire des vecteurs"""
    n = len(valeurs)
    total = 0
    for i in range(n):
        total = total + valeurs[i]*choix[i]
    return total
```

Solution de l'exercice VII.7 -

```
def somme(liste):
    s = 0
    for x in liste:
        s = s + x
    return s
```

```
def part(valeurs):
    """Entrée : une liste de nombres positifs
       Sortie : la plus grande somme partielle inférieure à la
           moitié
                de la somme totale et le choix qui la donne"""
    N = len(valeurs)
    tout = somme(valeurs)
    presque = 0
    choix0 = [0]*N
    for n in range(2**N):
        choix = binaire(n,N)
        part = totalPartiel(valeurs,choix)
        if part <= (tout/2) and part > presque:
            choix0 = choix
            presque = part
    return presque, choix0
```

Solution de l'exercice VII.8 -

```
def ternaire(n,p):
    """Entrée : deux entiers
       Requis : 0 <= n < 3**p
       Sortie : la liste des p coefficients de l'écriture en
          base 3
                de n commençant le bit de poids fort"""
    ter = [0]*p
    for i in range(p):
        ter[p-1-i] = n%3
        n = n//3
    return ter
```

```
def sommes3(valeurs,choix):
    """Entrée : deux listes de même longueur,
                la seconde de termes 0, 1 ou 2
       Sortie : les 3 sommes des valeurs correspondant
                respectivement aux 0, 1 et 2"""
    s = [0, 0, 0]
    n = len(valeurs)
    for i in range(n):
        k = choix[i]
        s[k] = s[k] + valeurs[i]
    return s
```

```
def parts3(valeurs):
    """Entrée : une liste de nombres positifs
       Sortie : les parts s'approchant le plus d'un partage à
          3
                et la liste des choix le permettant"""
    N = len(valeurs)
    tout = somme(valeurs)
    ecartMin = 2*tout
    choix0 = [0]*N
    for n in range(3**N):
        choix = ternaire(n,N)
        parts = sommes3(valeurs,choix)
        ecart = abs(tout/3-parts[0]) + abs(tout/3-parts[1]) +
           abs(tout/3-parts[2])
        if ecart < ecartMin:
            choix0 = choix
            ecartMin = ecart
    return sommes3(valeurs,choix0), choix0
```

Solution de l'exercice VII.9 -

```
def distance(i, j, villes):
    """Entrée : 2 entiers et une listes de triplets
                (nom, latitude, longitude)
       Sortie : la distance entre les villes d'indices i et j
          """
    ltA = villes[i][1]*2*pi/360
    ltB = villes[j][1]*2*pi/360
    lgA = villes[i][2]*2*pi/360
    lgB = villes[j][2]*2*pi/360
    ltM = (ltA+ltB)/2
    deltaX = (lgB-lgA)*rTerre*cos(ltM)
    deltaY = (ltB-ltA)*rTerre
    return sqrt(deltaX**2 + deltaY**2)
```

Solution de l'exercice VII.10 -

```
def tour(liste, villes):
    """Entrée : une liste d'indices et une listes de triplets
                (nom, latitude, longitude)
       Sortie : la distance totale de la tournée
                représentée par la liste"""
    long = distance(0,liste[0])
    n = len(liste)
    for i in range(n-1):
        long = long + distance(liste[i],liste[i+1])
    long = long + distance(0,liste[n-1])
    return long
```

Solution de l'exercice VII.11 -

```
l0 = list(range(1,len(villes0)))
dist0 = tour(l0)
distMin = dist0
cheminMin = l0
for liste in itertools.permutations(l0):
    d = tour(liste, villes0)
    if d < distMin:
        distMin = d
        cheminMin = liste
print("La distance minimale est :",distMin)
print("Elle correspond au trajet")
print(villes0[0][0])
for k in cheminMin:
    print(villes0[k][0])
print(villes0[0][0])
```

```
La distance minimale est : 2381.165121342024
Elle correspond au trajet
Paris
Lyon
Nice
Toulon
Marseille
Toulouse
Bordeaux
Nantes
Lille
Douai-Lens
Paris
```

Solution de l'exercice VII.12 -

```
def echanger(liste,i,j):
    """Entrée : une liste et deux entiers
       Requis : 0 <= i,j < len(liste)
       Sortie : les termes d'indices i et j ont été échangés
          dans la liste"""
    temp = liste[i]
    liste[i] = liste[j]
    liste[j] = temp
```

```
def retourner(liste,i,j):
    """Entrée : une liste et deux entiers
       Requis : 0 <= i,j < len(liste)
       Sortie : les termes d'indices i à j ont été retournés
          dans la liste"""
    c = (i+j-1)//2
    for k in range(i,c+1):
        echanger(liste,k,i+j-k)
```

```
def suivant(liste):
    """Entrée : une liste
       Sortie : la liste permutée suivante"""
    l = deepcopy(liste)
    n = len(l)
    j = n-2
    while j >= 0 and l[j] >= l[j+1]:
        j = j-1
    if j == -1:
        return []
    else:
        k = n-1
        while l[k] <= l[j]:
            k = k-1
        echanger(l,j,k)
        retourner(l,j+1,n-1)
        return l
```

TP VIII

Modélisation d'un amortisseur

1 Présentation

Le laboratoire de sciences industrielles dispose du matériel qui permet d'étudier une suspension de moto avec la possibilité d'effectuer des mesures.

Lors de la manipulation un capteur (un accéléromètre) calcule les accélérations d'un point de la partie mobile et enregistre celles-ci dans un fichier texte.
On souhaite traiter ce fichier afin de corréler l'expérience avec un modèle de connaissance du type oscillant amorti.

Il s'agit donc d'étudier le mouvement vertical oscillant amorti d'un solide en translation verticale dans un champ gravitationnel vertical dirigé vers le bas.
Le solide est guidé par une liaison de direction verticale possédant :

- une raideur qui crée une force, en $[N]$, proportionnelle au déplacement vertical relatif par rapport à la position d'équilibre en $[m]$ et qui s'oppose au déplacement,
- un amortissement visqueux qui crée une force verticale en $[N]$ qui s'oppose à la vitesse de déplacement relative entre le solide et la référence en $[m \cdot s^{-1}]$.

1.1 Création du fichier

Le fichier est créé selon le protocole suivant.

- On part de la position d'équilibre statique du système.
- On déplace manuellement vers le bas de $100mm$ le châssis de la moto.
- On lance une acquisition pour $6s$ avec une fréquence d'échantillonnage de $200Hz$.
- On lâche le châssis sans condition initiale de vitesse.
- On sauvegarde le fichier relatif à l'expérience sous le nom `moto1.txt` en demandant une indexation des points.
- Le fichier `moto.txt` est une suite de caractères générée par le logiciel d'acquisition.
- Il comporte 1200 lignes car l'acquisition se fait à $200Hz$ pour une acquisition de 6 secondes.
- Chaque ligne est l'enregistrement de 4 nombres :

```
   1  +0.0000000  +7.3876953  +1.9042969
   2  -0.0048828  +7.3925781  +1.8994141
   3  -0.0097656  +7.3974609  +1.8994141
 ...         ...         ...         ...
1200  -0.0048828  +7.3925781  +1.5283203
```

- La première colonne donne le numéro de la mesure, ici de 1 à 1200.
- Les trois dernières colonnes correspondent aux mesures de l'accéléromètre, ce sont les composantes de l'accélération dans une base orthonormée : à l'équilibre, elles mesurent l'accélération de la pesanteur. La composante verticale est écrite dans la deuxième coordonnée, c'est-à-dire la troisième colonne. C'est cette mesure que nous étudierons.
- Dans le fichier, chaque ligne est donc composée de quatre éléments.
  Ces éléments sont séparés par des caractères de tabulation, `"\t"`.
  Chaque ligne est terminée par un caractère de changement de ligne `"\n"`.

On peut ouvrir le fichier dans un éditeur pour en lire la structure.
Dans un éditeur classique les tabulations et les retours à la lignes seront traduits et on obtient un alignement vertical comme dans l'exemple ci-dessus.

1.2 Objectifs

Première phase : convertir le fichier texte `moto1.txt` en une structure de données utilisable par python.

Deuxième phase : transformer ces données pour obtenir une représentation de la vitesse et de la position de l'accéléromètre.

Troisième phase : rechercher les grandeurs caractéristiques associées à un modèle mécanique permettant son identification.

1.3 Rappels et compléments sur les fichiers

Pour utiliser un fichier on doit ouvrir une connexion avec celui-ci par l'instruction `open`. Celle-ci a deux arguments : le nom du fichier (une chaîne de caractères) et le type qui peut être `"r"` pour la lecture, `"w"` pour l'écriture d'un nouveau fichier et `"a"` pour l'ajout à un fichier existant.
On peut importer le fichier

- comme une grande chaîne de caractères : `read`
- comme une liste de lignes : `readlines`
- ligne à ligne : `readline`

Une ligne est une portion du fichier délimitée par le caractère de retour à la ligne `"\n"`, ce caractère est conservé en fin de ligne.
On obtient donc la liste des lignes du fichier d'acquisition avec

```
## Importations
nom = 'moto.txt'
fichier = open(nom, 'r')
listeBrute = fichier.readlines()
fichier.close()
```

On rappelle l'importance de fermer la communication : `fichier.close()`.

La méthode ci-dessus ne permet de charger le fichier que si le répertoire de travail est celui qui contient le fichier.
Si ce n'est pas le cas on peut donner le chemin complet d'accès au fichier.
On peut aussi changer le répertoire de travail.
Python contient un module, `os`, qui fournit des instructions qui permettent de gérer les fichiers. Nous utiliserons la fonction `chdir` qui permet de se déplacer dans l'arborescence du fichier. La méthode la plus simple pour l'utiliser est de donner le nom complet du répertoire comme argument.

```
## Importations
import os

os.chdir("/home/profs/detrez.eric/Travail/2018-2019") # Linux
# os.chdir("H://Travail//2018-2019") # Sous Windows
```

On notera le dédoublement du séparateur `"//"` sous Windows.
Ce nom est lu, par exemple, dans le **file browser** de Pyzo.

2 Traitement initial

On peut voir la structure non traduite des lignes dans la console ; la fonction `print` interprète les caractères spéciaux donc produit un affichage différent..

```
>>> listeBrute[55]
'        56\t-0.0097656\t+7.3974609\t+1.8945313\n'

>>> print(listeBrute[55])
        56   -0.0097656   +7.3974609   +1.8945313
```

Exercice VIII.1
Écrire une fonction `traduction(ch)` *qui transforme une chaîne de caractères semblable aux lignes du fichiers en une liste de nombre.*
On pourra utiliser

- une extraction `ch[:-1]` pour enlever le dernier caractère (`"\n"`)
- la méthode `split(c)` qui transforme une chaîne de caractères en une liste de chaînes de caractères qui sont les morceaux découpé par le paramètre. Par exemple

```
>>> ch = "anticonstitutionnellement"
>>> ch.split('t')
['an', 'icons', 'i', 'u', 'ionnellemen', '']
>>> ch.split('ti')
['an', 'cons', 'tu', 'onnellement']
>>> ch.split('n')
['a', 'tico', 'stitutio', '', 'elleme', 't']
```

- la fonction de conversion de type, ici `float`, pour transformer une chaîne en flottant.

Exercice VIII.2
Écrire les instructions (dans la partie principale) qui permettent de traduire `listeBrute` *en une liste* `liste` *dont les éléments sont des listes de 4 nombres.*
Le premier élément de la liste devra donc être `[1.0, 0.0000000, 7.3876953, 1.9042969]`.

Exercice VIII.3
Écrire une fonction `extraire(k,liste)` *où k est un entier et* `liste` *est une liste dont les éléments sont des listes de longueur k au moins et qui renvoie la liste des k-ièmes composantes.*
`extraire(1, [[1, 2, 3], [4, 5, 6]])` *doit renvoyer* `[2, 5]`
N.B. La k-ième composante correspond à l'indice $k-1$.

On peut maintenant visualiser les différentes colonnes.

```
import matplotlib.pyplot as plt

plt.plot(extraire(1,liste)) # 0 ou 1 ou 2 ou 3
plt.show()
```

- Pour la première colonne les valeurs sont affines par rapport à l'indice.
- La deuxième colonne semble être du bruit
- La troisième et la quatrième colonnes montrent des oscillations, un peu bruitées.

3 Traitement des listes

Dans cette section on se propose
- de déterminer l'origine temporelle des variations,
- de convertir les résultats signifiants en unité connues,
- de calculer puis retirer l'accélération gravitationnelle,
- de calculer les vitesses puis les déplacement à partir de l'accélération.

On définit, après les instructions de la **question VIII.2**, les listes particulières :

```
tempsBrut = extraire(0, liste)
accBrute = extraire(2, liste)
```

3.1 Origine et échelles

Le tracé de `accBrute` (la troisième composante ci-dessus) montre que l'accélération n'a pas varié immédiatement.

Exercice VIII.4
Écrire une fonction `rechZero(liste, ecart)` *qui recherche le premier indice pour lequel la valeur de la liste s'écarte d'au moins* `ecart` *de la valeur initiale.*

On choisit dans la suite un écart assez petit [1] pour déterminer l'origine temporelle. On ajoutera, par exemple, l'instruction suivante dans la partie principale.

```
i0 = rechZero(accBrute,1e-2) - 1
```

Avec la fréquence d'échantillonnage, on connaît le temps entre deux mesures : $\Delta t = \frac{1}{f} = 0.005\,s$.
On notera `Delta_t` une variable qui prend la valeur Δt dans la suite.

Exercice VIII.5
Écrire les instructions (dans la partie principale) qui calculent la liste des temps en secondes, notée `temps`*, en commençant avec $t = 0$ pour l'origine du phénomène (le point d'indice* `i0`*).*
La liste sera plus courte que la liste originale.

Pour déterminer la valeur de la gravitation que l'appareil mesure on **admet** [2] qu'elle est la moyenne des mesures de `accBrute`.

Exercice VIII.6
Écrire une fonction `moy(liste)` *qui calcule la valeur moyenne de la liste.*

L'accéléromètre mesure l'accélération a (sous la forme d'une tension V) dans le référentiel terrestre, dans lequel le mouvement est la composition d'une chute libre et des oscillations amorties.
On admet que, dans le référentiel qui «tombe» avec le système, l'accélération vaut $a' = a - g$: c'est cette accélération que l'on cherche à obtenir.
On prend pour mesure de la gravitation $g = 9.81\,m/s^2$.
On admet aussi qu'il existe une relation affine entre a' (en m/s^2 et V (en volt) :

$$a' = a - g = \alpha V + \beta$$

Cette loi est étalonnée avec les données suivantes :
- On a mesuré $V = V_0 = 4,8291$ quand le système est au repos dans le référentiel terrestre. Cela correspond à $a = 0$ donc $a' = -g$.
- La moyenne des accélérations sur toute la durée des mesure correspond à $a = g$ donc $a' = 0$.

1. Mais pas trop petit pour éviter les variations aléatoires.
2. On pouvait aussi choisir la moyenne des mesures avant les oscillations, ce qui donne un résultat très proche.

Exercice VIII.7
Écrire les instructions (dans la partie principale) qui calculent la liste des accélérations en m/s^2, notée `acc`*, en commençant avec $t = 0$ pour l'origine du phénomène.*
`acc` *sera de même longueur que la liste* `temps`*.*

On a maintenant une liste dont les valeurs sont les temps, régulièrement espacés, en lesquels le système a mesuré une accélération. Si on note t_i la valeur de `liste[i]` alors la valeur `acc[i]` est la valeur de l'accélération au temps t_i.
On peut alors représenter cette fonction accélération les unités.

```
plt.title("Accélération") # Ajout d'un titre
plt.xlabel("Temps en secondes") # Légende des abscisses
plt.ylabel("Accélération en m/s^2") # Légende des ordonnées
plt.grid(True) # Ajout d'une grille
plt.plot(temps,acc,
         color="magenta", # couleur de la ligne
         linewidth=2.5,   # épaisseur de la ligne
         linestyle="-",   # type de ligne
         label="Accélération") # nom de la fonction tracée
plt.legend(loc='upper right') # placement de ce nom
plt.show()
```

`matplotlib` permet d'ajouter beaucoup de personnalisations aux graphes.

3.2 Vitesses et positions

On cherche ensuite à évaluer les vitesses et les positions aux temps t_i.
Si on connait la dérivée d'une fonction en t_i et si t_i et t_{i+1} sont suffisamment proches on peut approcher le taux d'accroissement par la dérivée : $\dfrac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i} \sim f'(t_i)$.
On peut alors calculer les valeurs de f de proche en proche : $f(t_{i+1}) \sim f(t_i) + (t_{i+1} - t_i).f'(t_i)$.
On remarque qu'on a besoin d'une valeur initiale $f(t_0)$.
Comme les temps sont régulièrement espacés on a besoin uniquement du pas entre deux temps : $\Delta t = t_{i+1} - t_i$

Exercice VIII.8
Écrire une fonction `primitive(listeDer, dt, valeurInit)` *où* `listeDer` *est une liste représentant les valeur de $f'(t_i)$,* `dt` *est le pas de temps et* `valeurInit` *est la valeur initiale de la fonction en t_0 et qui renvoie une liste, de même longueur que* `listeDer`*, contenant des valeurs approchées de $f(t_i)$.*
On remarque qu'on n'utilise pas la dernière valeur de `listeFonction`.

Exercice VIII.9
Calculer et tracer le graphe des vitesses et des positions.
La vitesse initiale est 0 $m.s^{-1}$, la position initiale est -0,10 m.

Exercice VIII.10
Enregistrer la suite des valeurs des positions dans un fichier `moto1.txt`*.*

4 Identification des caractéristiques

La réponse du système est identifiable à la solution de l'équation différentielle suivante :

$$\frac{1}{\omega_0^2}x''(t) + \frac{2\xi}{\omega_0}x'(t) + x(t) = 0$$

- où $x(t)$ est la position verticale repérée
- ξ est le coefficient d'amortissement,
- ω_0 est la pulsation propre du système

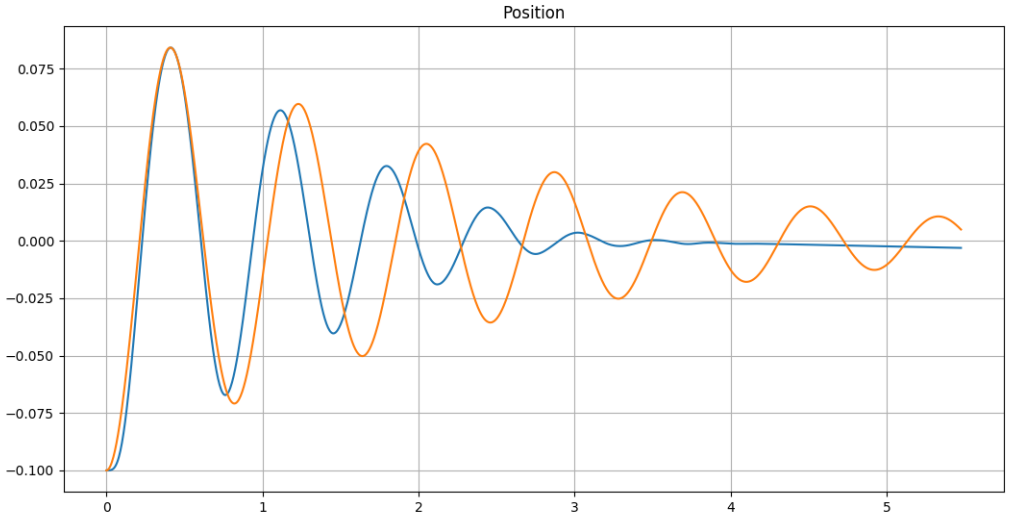
La solution telle que $x(0) = -e_0 = -0.1$ et $x'(0) = 0$ est

$$x(t) = -e_0 e^{-\xi\omega_0 t}\left(\cos(\omega t) + \frac{\xi\omega_0}{\omega}\sin(\omega t)\right) \text{ avec } \omega = \omega_0\sqrt{1-\xi^2}$$

Exercice VIII.11
Prouver que $x'(t) = \dfrac{e_0\omega_0}{\sqrt{1-\xi^2}}e^{-\xi\omega_0 t}\sin(\omega t)$.
En déduire les valeurs de t *et de* x *aux extremums.*
On va donc calculer les valeurs du temps, t_1, et de la position au premier extremum après l'origine, e_1, on remarque que cela correspond au maximum de la courbe.
On a alors $\omega_0 = \dfrac{\pi}{t_1\sqrt{1-\xi^2}}$ et $\dfrac{\xi}{\sqrt{1-\xi^2}} = \dfrac{-1}{\pi}\ln\left(\dfrac{e_1}{e_0}\right)$.

Exercice VIII.12
Écrire une fonction qui calcule l'indice (le premier) du maximum d'une liste.
En déduire les valeurs de t_1 *et* e_1 *puis de* ξ *et* ω_0.
`pi` *et* `log` *(pour* ln*) seront importés du module* `math`.

Voici le résultat obtenu :



Exercice VIII.13 — Question ouverte
Comment améliorer les résultats ?

5 Solutions

Solution de l'exercice VIII.1 -

```
def traduction(ch):
    """Entrée : une chaîne de caractères représentant
                une suite de nombres séparés par des
                   tabulations
                et terminant par '\n'
       Sortie : la liste de ces nombres"""
    ch1 = ch[:-1] # On enlève le caractère '\n' final
    l1 = ch1.split('\t') # On découpe
    l2 = []
    n = len(l1)
    for i in range(n):          # Les éléments
        l2.append(float(l1[i])) # sont sont convertis en
           flottants
    return l2
```

On peut aussi faire le travail à la main en testant s'il y a un entier ou un flottant

```
def traduction(ch):
    """Entrée : une chaîne de caractères représentant
                une suite de nombres séparés par des
                   tabulations
                et terminant par '\n'
       Sortie : la liste de ces nombres"""
    l = []
    morceau = ''
    for car in ch: # On lit les caractères
        if car == '\t' or car == '\n': # fin d'un morceau
            l.append(float(morceau)) # on covertir en nombre
            morceau = '' # on re-initialise le morceau
        else:
            morceau = morceau + car # on ajoute le caractère
    return l
```

Solution de l'exercice VIII.2 -

```
liste = []
n = len(listeBrute)
for i in range(n):
    liste.append(traduction(listeBrute[i]))
```

ou `liste = [traduction(ch) for ch in listeBrute]`

Solution de l'exercice VIII.3 -

```
def extraire(k,liste):
    """Entrée : un entier et une liste
       Requis : les éléments de la liste sont des listes
                de longueur au moins k
       Sortie : la liste des k-ièmes composantes"""
    resultat  = []
    for l in liste:
        resultat.append(l[k-1])
    return resultat
```

Solution de l'exercice VIII.4 - On choisit de renvoyer la longueur de la liste si elle ne diffère jamais de plus d'écart depuis sa position initiale.

```
def rechZero(liste,ecart):
    """Entrée : une liste et un nombre
       Sortie : le premier indice en lequel
                la liste diffère de liste[0]
                d'au moins ecart"""
    init = liste[0]
    n = len(liste)
    for i in range(n):
        if abs(liste[i] -init) >= ecart:
            return(i)
    return n
```

Solution de l'exercice VIII.5 -

```
f = 200
Delta_t = 1/f
x0 = tempsBrut[i0]
n = len(tempsBrut)
temps = []
for i in range(i0, n):
    x = tempsBrut[i]
    t = (x-x0)*Delta_t
    temps.append(t)
```

La boucle peut être remplacée par `temps = [(x-x0)*Delta_t for x in tempsBrut[i0:]]`

Solution de l'exercice VIII.6 -

```
def moy(liste):
    """Entrée : une liste non vide
       Sortie : la moyenne des termes"""
    n = len(liste)
    som = 0
    for i in range(n):
        som = som + liste[i]
    return som/n
```

Solution de l'exercice VIII.7 -

```
vg = moy(accBrute)
v0 = 4.8291
g = 9.81
n = len(accBrute)
acc = []
for i in range(i0, n):
    acc_V = accBrute[i]
    acc_ms2 = (acc_V-vg)*g/(vg-v0)
    acc.append(acc_ms2)
```

ou, en utilisant la définition de listes par compréhension,

```
vg = moy(accBrute)
v0 = 4.8291
g = 9.81
acc = [(x-vg)*g/(vg-v0) for x in accBrute[i0:]]
```

Solution de l'exercice VIII.8 -

```
def primitive(listeDer, dt, valeurInit):
    """Entrée : une liste de nombres et deux nombres
       Sortie : la liste des primitives"""
    listef = [valeurInit]
    n = len(listeDer)
    for i in range(n-1): # il reste n-1 valeurs à calculer
        new_f = listef[i] + dt*listeDer[i]
        listef.append(new_f)
    return listef
```

Solution de l'exercice VIII.9 -

```
vit = primitive(acc, Delta_t, 0)
pos = primitive(vit, Delta_t, -0.1)

plt.subplot(3,1,1)
plt.title("Accélération")
plt.grid(True)
plt.plot(temps,acc)
plt.subplot(3,1,2)
plt.title("Vitesse")
plt.grid(True)
plt.plot(temps,vit)
plt.subplot(3,1,3)
plt.title("Position")
plt.grid(True)
plt.plot(temps,pos)
plt.show()
```

Solution de l'exercice VIII.10 -

```
fichier = open("moto1.txt", "w')
n = len(pos)
for i in range(n):
    x = str(pos[i]) + "\n"
    fichier.write(x)
fichier.close()
```

Solution de l'exercice VIII.11 -

$x'(t) = -e_0 e^{-\xi\omega_0 t}\left(-\xi\omega_0\cos(\omega t) + \dfrac{-\xi^2\omega_0^2}{\omega}\sin(\omega t) - \omega\sin(\omega t) + \xi\omega_0\cos(t)\right)$.

Or $\dfrac{\xi^2\omega_0^2}{\omega} + \omega = \dfrac{\xi^2\omega_0^2 + \omega^2}{\omega} = \dfrac{\xi^2\omega_0^2 + \omega_0^2(1-\xi^2)}{\omega} = \dfrac{\omega_0^2}{\omega} = \dfrac{\omega_0}{\sqrt{1-\xi^2}}$.

On voit donc que les extremums sont atteints aux temps $t_k = \dfrac{k\pi}{\omega}$ et leurs valeurs sont de la forme $-(-1)^k e_0 e^{-\xi\omega_0 t_k}$.

Solution de l'exercice VIII.12 -

```
def indMax(liste):
    indMax = 0
    n = len(liste)
    for k in range(n):
        if liste[k] > liste[indMax]:
            indMax =k
    return indMax
```

```
i1 = indMax(pos)
t1 = temps[i1]
e1 = pos[i1]
q = -log(e_1/e_0)/pi
xi = q/(pi**2+q**2)**0.5
omega0 = pi/t1/(1-xi**2)**0.5
```

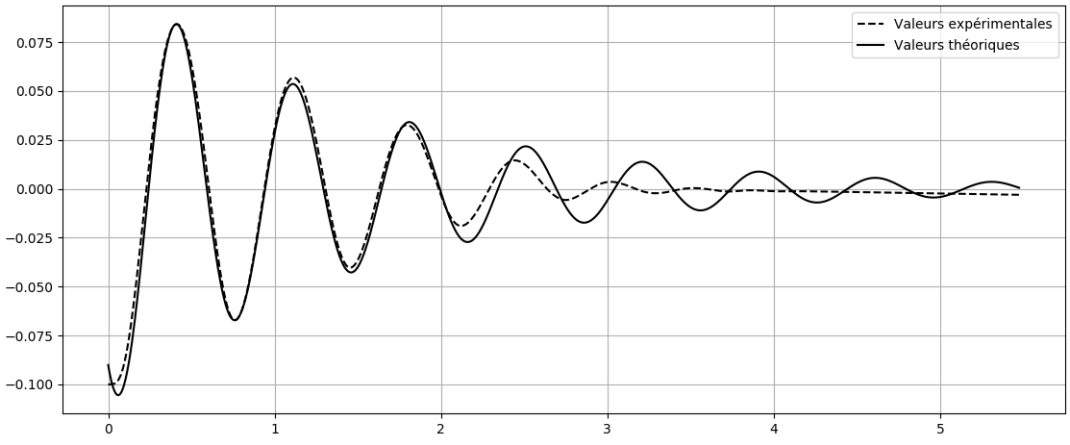
Solution de l'exercice VIII.13 - On peut penser que le calcul de l'origine n'est pas idéal et espérer calculer les constantes à l'aide des deux premiers extremums. t_1 est remplacé par le temps entre les deux extremums et on calcule le quotient entre les valeurs extrémales.

On change aussi la fonction solution pour qu'elle prenne la valeur e_1 en t_1.

```
i1 = indMax(pos)
i2 = indMinDepuis(pos, i1)
t1 = temps[i1]
t2 = temps[i2]
t = (t2-t1)
omega = pi/t
e1 = abs(pos[i1])
e2 = abs(pos[i2])
q = log(e1/e2)/pi
xi = q/(1+q**2)**0.5
omega0 = pi/t/(1-xi**2)**0.5

sol = [solution(t) for t in temps]
plt.plot(temps,pos,linestyle = "dashed", color = "black",label
   ="Valeurs expérimentales")
plt.plot(temps, sol,color = "black",label="Valeurs théoriques"
   )
plt.grid()
plt.legend()
plt.show()
```

C'est moins pire


Valeurs expérimentales
Valeurs théoriques
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
0
1
2
3
4
5

TP IX

Exploitation de données

1 Introduction

Le but de ce T.P. est de d'utiliser le langage SQL pour exploiter une base de données.
Un base de données simple est une ensemble de données regroupées

- par ligne, ce sont les données d'un objet (ici les communes de France continentale)
- par colonnes, ce les différents type de données (population, département, ...)

Nous allons utiliser des données fournies par l'INSEE, résultat des recensements.
Ce fichier est relativement simple mais très long : plus de 35000 lignes.
On pourrait le traiter à l'aide d'un tableur (c'est son format d'origine).
Cependant même les questions simples :

- quelles sont les villes de plus de 200 000 habitants ?
- quelle est la commune la plus étendue (`surf` en km^2) ?
- quelles sont les communes dont la population a plus que doublé depuis 1968 ?

demanderaient des manipulations qui seraient longues à effectuer.
Le format choisi pour ces T.P. est `SQL Lite` : d'autres formats sont possibles (`MySQL`, `postgreSQL`, `Oracle`, ...) mais tous partagent le cœur des fonctions `SQL`.

1.1 sqlitebrowser

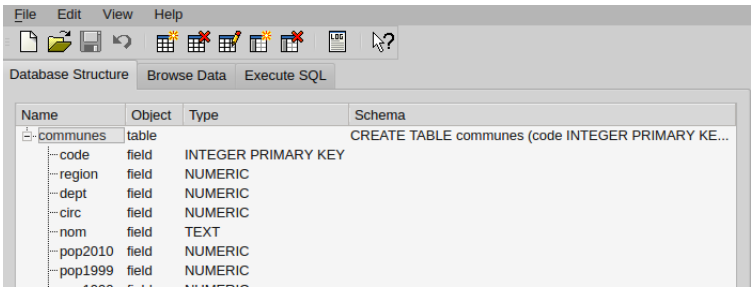
En général l'intérêt d'une base de données est qu'elle peut être utilisée dans un réseau : la mise en place est assez lourde mais le partage d'une même base par plusieurs utilisateurs est un gros avantage.
Nous allons utiliser un logiciel plus léger pour l'apprentissage du langage SQL : chaque poste exploitera sa base de données. Le logiciel choisi est SQLITEBROWSER.
Il faut commencer par copier la base de données sur le bureau (Windows) ou dans le répertoire local à votre nom (Linux). La base disponible dans le dossier de votre classe dans le disque `Public` : `communes.sqlite`.
On peut ensuite lancer le logiciel :

Windows	**Linux**
Copier le logiciel SQLITEBROWSER depuis le disque `P:` du réseau vers le bureau et le lancer	Ouvrir SQLITEBROWSER dans le menu des application, sous-menu `programmation`.

Le logiciel est en anglais.
Ouvrir la base de données `communes.sqlite` depuis le menu `Files`, item `Open Database`.

Le logiciel présente plusieurs onglets, nous en utiliserons 3.

- L'onglet par défaut, `Database Structure`, permet de voir l'agencement des différentes tables. Notre premier TP se contente d'une seule table. C'est ici que l'on peut s'assurer du nom des variables.
- L'onglet `Browse Data` permet de voir toutes les données comme dans un tableur : en général il n'est pas utilisable car les tables auront une grande taille.
- L'onglet `Execute SQL` est l'endroit où nous allons travailler.
  On entre la commande dans la fenêtre `SQL` ou `SQL String` et on la fait exécuter en cliquant sur la flêche (ou sur `Execute Query`).



2 Extraction

L'exploitation d'une base de données consiste à poser des question, appelées **requêtes**, compréhensibles pas le langage afin d'extraire les informations qui nous intéressent.
La structure de base d'une requête est donnée par l'exemple

```
select nom, POP2010     /* intitulés des colonnes demandées */
from communes           /* tables utilisées */
where POP2010 > 200000 /* Conditions filtrant les lignes */
```

Ici on demande le nom et la population (en 2010) des villes de plus de 200.000 habitants.

On peut améliorer la présentation en triant les résultats.
Pour cela on ajoute une ligne `order by variable` où `variable` est le nom de l'attribut ou de la variable calculée qui sert de clé de tri. On peut remplacer le nom de la colonne par son numéro (commençant par 1) dans la liste des variables affichées.
Les résultats sont affichés selon l'ordre croissant, si on veut l'ordre inverse on suit la commande par `desc` .

Pour faire afficher tous les attributs on peut écrire `select *`.

Exercice IX.1
Trouver tous les renseignement d'une ville que vous choisissez.
Attention : le nom d'une ville (par exemple) est une chaîne de caractères, il faut l'encadrer par des guillemets.

Les conditions seront souvent basées sur des comparaisons :
les opérateurs sont `=, != (ou <>), <, >, <=, >=`.

Exercice IX.2
Trouver les noms des communes de moins de 10 habitants (< 10) avec leur département et leur population. Il y en a 22
Si les valeurs sont des chaînes de caractères on peut filtrer sur une partie du nom :

1. `_` représente un caractère quelconque,
2. `%` représente une suite (éventuellement vide) de caractères.
3. L'opérateur s'écrit alors `like` et non =.

Par exemple les villes dont le nom contient *truc* sont recherchées par `nom LIKE "%truc%"`

Exercice IX.3
Trouver les communes dont le nom contient `kerque` *et leur département ; il y en a 10.*

Les recherches peuvent être composées par "ou" (`OR`) ou "et" (`AND`).

Exercice IX.4
Trouver les noms des communes de plus de 30000 habitants et de surface inférieure à 5 km^2 avec leur département, leur population et leur surface.
(19, toutes en région parisienne)

1. Les tests et les résultats montrés peuvent contenir des calculs.
2. On peut, pour en donner la signification ou pour l'utiliser ensuite, nommer un résultat avec le mot-clé `as`. Par exemple on peut obtenir et nommer la variation de population avec `pop2010 - pop1968 AS deltaPop`.
3. Si on effectue une division entre des entiers on a la division euclidienne, à quotient entier. Pour obtenir la division réelle classique on doit imposer qu'un nombre soit réel. On peut, par exemple, écrire `(a + 0.0)/b`.
4. Les valeurs affichées peuvent présenter un nombre déterminé de décimales avec `round(x,k)` qui affiche x avec k décimales.

Exercice IX.5
Trouver les noms des communes dont la densité de population est supérieure à 20000 habitants par km^2 avec leur département, leur population et leur surface.
(7 communes, toutes dans la région parisienne)

Exercice IX.6
Trouver les noms des communes dont la densité de population est inférieure à 1 habitant par km^2 avec leur département, leur population, leur surface et leur densité. (23 communes)

Exercice IX.7
Trouver les noms des communes avec leur département et leur population dans l'ordre décroissant de population.

Exercice IX.8
Trouver les 10 communes dont le taux d'augmentation de population a été le plus important entre 1999 et 2010.
On pourra se restreindre aux villes de plus de 100.000 habitants.

Exercice IX.9
Quelle commune a le nom le plus long ? Le nom le plus court ?
On donnera aussi le département.
La longueur d'une chaîne de caractères s'obtient par `length(ch)`.
On peut n'afficher que les p premiers résultats en ajoutant l'instruction `limit p`.

Exercice IX.10
La base n'est pas cohérente : la population totale (`pop2010`*) n'est pas toujours égale à la somme des populations par genre. Déterminer les communes dans lesquelles il y a une erreur. Il y en a 373.*

3 Statistiques

On peut compter le nombre d'éléments avec `count()` ; il ne peut pas y avoir d'autres éléments affichés car il n'y a qu'une valeur à afficher pour toutes les données.

Exercice IX.11
Combien y-a-t-il de communes dans la base ? (36205)
D'autres fonctions sont disponibles, elles calculent un résultat en fonction d'une valeur pour chaque ligne. On pourra utiliser la moyenne d'une donnée (`AVG`), le maximum (`MAX`), le minimum (`MIN`), la somme (`SUM`).

Exercice IX.12
Quelle est la moyenne de population des communes du Nord ? (3964.2...)

Exercice IX.13
Quelle est la somme des superficies des communes du Nord ? (5734 km^2)

Exercice IX.14
Quelle est la différence entre le nombre de femmes et le nombre d'hommes recensés ?
1 971 799 femmes de plus !
On peut aussi compter par catégorie.

1. On choisit un critère, par exemple `dept`. On l'indique en ajoutant, après les lignes classiques (`select, from, where`) une ligne `group by dept`.
2. La base est alors regroupée selon ce critère, un paquet par valeur.
3. On peut alors calculer une fonction statistique par paquet. On indique le critère de regroupement et la fonction calculée dans la ligne `select`.

Exercice IX.15
Calculer le nombre de communes par département.
Quels sont les 3 départements qui comportent le plus grand nombre de communes ? (62,2,80)

Exercice IX.16
Quels sont les 3 départements les plus peuplés ? (59,75,13)

Exercice IX.17
Déterminer la densité de population par département.

Exercice IX.18
Quels est le nom de commune le plus courant ?
14 communes s'appelent Sainte Colombe

On peut avoir besoin de filtrer selon un résultat du regroupement. Pour cela l'instruction de test s'écrit dans une ligne commençant par `having` située en dessous de la ligne `group by`.

Exercice IX.19
Quels sont les département de population supérieure à 1.500.000 ? (59, 75, 13, 69, 92, 93)

Exercice IX.20
Quels sont les départements dont la population de la ville la plus peuplée est au moins un tiers de la population totale ? (13, 31, 75, 87, 90)

Exercice IX.21
Quels sont les départements dont la population de plus de 60 ans représente plus de 30% de la population totale ? (10 départements)

4 Requêtes imbriquées

Si le résultat d'une requête est un tableau on peut l'utiliser pour une autre requête.
Par exemple après avoir calculé la moyenne de la population par arrondissement,

```
select dept, arr, sum(POP2010 )as s
from communes
group by arr
```

on peut calculer la moyenne des habitants par arrondissement pour chaque département.

```
select dept, avg(s) as moyenne
from (select dept, arr, sum(POP2010 )as s
      from communes
      group by arr)
group by dept
order by moyenne desc
```

Selon les versions du logiciel on peut enregistrer le résultat d'une requête sous la forme d'une vue (`view`) et l'utiliser plusieurs fois sans avoir besoin de la ré-écrire.

Si le résultat d'une requête est un nombre (`count`, `avg`, `max`, ...) on peut l'utiliser dans un test ou un calcul.

Exercice IX.22
Après avoir calculé la population par département, déterminer la moyenne de la population par département. (664 420,7)

Exercice IX.23
Déterminer les départements qui ont une population inférieure au quart de la moyenne de la population par département. (48, 23, 5, 90, 15, 9, 4)

Exercice IX.24
Déterminer le nombre d'arrondissements par département.

Exercice IX.25
Déterminer le nombre de département qui ont 1, 2, 3, ..., 9 arrondissements.

Exercice IX.26
Quels sont les département dans lesquels sont situés les communes ayant le nom le plus courant (question IX.18) ?.

Exercice IX.27
Déterminer le département dont la surface est la plus proche de la surface moyenne.
On pourra trier selon la valeur absolue de la différence de la surface avec la moyenne et ne garder qu'un résultat (limit 1). Le département est 15.

On veut classer les villes par leur taille en les regroupant sous la forme 1 à 9 habitants, 10 à 99 habitants, etc Pour cela la fonction `length(n)`, qui renvoie le nombre de chiffres de n, est utile.

Exercice IX.28
Déterminer le nombre de personnes habitant dans une ville comportant 1 à 9 habitants, 10 à 99 habitants, ...

5 Résumé

Voici l'ordre des commandes possibles dans une requête SQL.

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT
```

Les deux premières lignes sont obligatoires (pour `SQL lite`), les suivantes sont facultatives mais doivent être placées dans l'ordre indiqué et n'apparaître qu'une fois au plus.

6 Requêtes sous Python

On peut utiliser des requêtes SQL dans Python ; on pourra ainsi utiliser les résultats.
Voici un exemple qui calcule la liste de la population par département

```
1  import os
2  os.chdir("/home/ericd13/Documents/2018-2019/IPT Sup/TP/TP07")
3
4  import sqlite3
5
6  connexion = sqlite3.connect('communes.sqlite')
7  curseur = connexion.cursor()
8  curseur.execute("select  dept, sum(POP2010 )as population from
       communes group by dept")
9  resultat = curseur.fetchall()
10 connexion.close()
```

- Les lignes 1 et 2 permettent de changer le répertoire de travail.
- La ligne 4 importe le module d'interface avec sqlite.
- La ligne 6 établit la liaison avec la base de données .
- La ligne 7 définit l'objet qui reçoit les requêtes et les réponses.
- La ligne 8 envoie la requête.
- La ligne 9 transmet le résultat sous forme de liste à une variable.
- La ligne 10 ferme la liaison.

La requête doit être sous la forme d'une chaîne de caractères. Lorsqu'elle est longue il vaut mieux l'enregistrer sous la forme d'un fichier texte et la charger depuis Python.

```
import os
os.chdir("/home/ericd13/Documents/2018-2019/IPT Sup/TP/TP07")

import sqlite3

fichier = open("requete.sql")
requete = fichier.read()
fichier.close()

connexion = sqlite3.connect('communes.sqlite')
curseur = connexion.cursor()
curseur.execute(requete)
resultat1 = curseur.fetchall()
connexion.close()
```

7 Solutions

Solution de l'exercice IX.1 -

```
select *
from communes
where nom = "Lille"
```

Solution de l'exercice IX.2 -

```
select nom, dept, POP2010
from communes
where POP2010 <10
```

Solution de l'exercice IX.3 -

```
select nom, dept
from communes
where nom like "%kerque%"
```

Solution de l'exercice IX.4 -

```
select nom, dept, POP2010, surf
from communes
where POP2010 > 30000 and surf <= 5
```

Solution de l'exercice IX.5 -

```
select nom, dept, POP2010, surf
from communes
where POP2010/surf > 20000
```

Solution de l'exercice IX.6 -

```
select nom, dept, POP2010, surf, round(POP2010/(0.0+surf),2)
   as densite
from communes
where densite < 1
```

Solution de l'exercice IX.7 -

```
select nom, dept, POP2010
from communes

order by POP2010 desc /* ou order by POP2010 desc */
```

Solution de l'exercice IX.8 -

```
select nom, dept, POP2010, POP1999, (POP2010 - POP1999)*100.0/
   POP1999 as taux
from communes
where POP2010 > 30000
order by taux desc
limit 10
```

Solution de l'exercice IX.9 -

```
select dept, nom, length(nom)
from communes

order by 3 desc
limit 1
```

Solution de l'exercice IX.10 -

```
select nom, femmes+hommes-pop2010 as erreur
from communes
where erreur != 0
```

Solution de l'exercice IX.11 -

```
select count()
from communes
```

Solution de l'exercice IX.12 -

```
select avg(pop2010)
from communes
where dept = 59
```

Solution de l'exercice IX.13 -

```
select sum(surf)
from communes
where dept = 59
```

Solution de l'exercice IX.14 -

```
select sum(femmes)-sum(hommes)
from communes
```

Solution de l'exercice IX.15 -

```
select dept, count() as nombreCommunes
from communes
group by dept
order by 2 desc
limit 3
```

Solution de l'exercice IX.16 -

```
select dept, sum(POP2010) as pop
from communes
group by dept
order by 2 desc
limit 3
```

Solution de l'exercice IX.17 -

```
select dept, sum(POP2010)/sum(surf) as densité
from communes
group by dept
```

Solution de l'exercice IX.18 -

```
select nom, count() as nombre
from communes
group by nom
order by 2 desc
limit 1
```

Solution de l'exercice IX.19 -

```
select dept, sum(POP2010) as population
from communes
group by dept
having population > 1 500 000
order by 2 desc
```

Solution de l'exercice IX.20 -

```
select dept, max(pop2010) as popMax, sum(pop2010) as pop
from communes
group by dept
having pop < 3*popMax
```

Solution de l'exercice IX.21 -

```
select dept, 100*(sum(age60_74)+sum(age75+0.0))/sum(pop2010)
   as taux_vieux
from communes
group by dept
having taux_vieux>30
```

Solution de l'exercice IX.22 -

```
select  dept, sum(POP2010 )as population
from communes
group by dept
```

```
select round(avg(population),1)
from (select  dept, sum(POP2010 ) as population
      from communes
      group by dept)
```

Solution de l'exercice IX.23 -

```
select dept, population
from (select  dept, sum(POP2010 ) as population
      from communes
      group by dept)
where population <  (select  avg(pop)
                     from (select  dept, sum(POP2010 ) as pop
                           from communes
                           group by dept))/4.0
order by 2 desc
```

Solution de l'exercice IX.24 -

```
select dept, count() as nbArr
from (select dept, arr,sum(pop2010) as popArr
      from communes
      group by arr)
group by dept
```

Solution de l'exercice IX.25 -

```
select nbArr, count() as nb
from (select dept, count() as nbArr
      from (select dept, arr,sum(pop2010) as popArr
            from communes
            group by arr)
      group by dept)
group by nbArr
```

Solution de l'exercice IX.26 -

```
select dept,  nom
from communes
where nom =(select nom
            from (select nom, count() as nombre
                  from communes
                  group by nom
                  order by 2 desc
                  limit 1
                 )
           )
```

Solution de l'exercice IX.27 -

```
select dept, surface
from (select dept, sum(surf) as surface
      from communes group by dept)
order by abs(surface - (select avg(surface)
                        from (select dept, sum(surf) as
                            surface
                               from communes group by dept
                              )
                       )
            )
limit 1
```

Si on a sauvé en une vue `surfDept` le résultat de

```
select dept, sum(surf) as surface
from communes
group by dept)
```

on peut simplifier la requête en

```
select dept, surface
from surfDept
order by abs(surface - (select avg(surface) from surfDept))
limit 1
```

Solution de l'exercice IX.28 -

```
select sum(POP2010) as population, count() as nombre,  rang
from (select nom, dept, POP2010, length(POP2010) as rang
      from communes)
group by rang
```