INF101 - INF131 - INF104
Algorithmique et programmation en Python
Cahier d'exercices : TD, TP, annales

Responsable d'UE : Carole Adam
carole.adam@imag.fr

Septembre 2021

Polycopié conçu à partir du cours INF204 2016 créé par :
Lydie du Bousquet, Aurélie Lagoutte, Julie Peyre, Florent Bouchez Tichadou, Amir Charif, Grégoire Cattan, Emna Mhiri

Table des matières

Chapitre 1

Exercices de TD

1.1 Les bases

1.1.1 Échange des valeurs de deux variables

On considère le programme ci-dessous :

```
print("Entrez deux reels :")
x = float(input())
y = float(input())
print("Vous avez saisi ",x," et ", y)
x = y
y = x
print("Apres echange des variables : ",x , " et ",y)
```

1. Dire ce qui s'affiche quand on exécute ce programme en donnant 1.23 et -17.5 en entrée.
2. Ce programme fait-il bien ce qu'il est censé faire et sinon comment le corriger ?
3. Compléter ce programme pour lire, échanger et afficher les valeurs de 3 variables.

1.1.2 Affichages

Écrire un programme qui demande à l'utilisateur 2 entiers A et B, et qui les affiche de la manière suivante :

1. Un par ligne
2. Sur la même ligne séparés par une virgule : A,B
3. Avec un message donnant le nom et la valeur : A=... et B=...
4. En damier

```
A B A
B A B
A B A
```

5. Sur la même ligne séparés par des tirets et avec un point d'exclamation final : A-B-A-B-!

1.1.3 Entrées, sorties, calculs

Écrire un programme qui demande à l'utilisateur 2 réels x et y, calcule leur somme et leur produit, et les affiche dans un message explicite.

Exemple d'exécution :

```
x=? 3.2
y=? 7
3.2 + 7 = 10.2
3.2 * 7 = 22.4
```

1.1.4 Bonjour

Écrire un programme qui demande à l'utilisateur son prénom et affiche un message de salutations personnalisé.
Exemple d'exécution :

```
Comment t'appelles-tu ?   Toto
Bonjour Toto !
```

1.1.5 Âge

Écrire un programme qui demande à l'utilisateur son année de naissance, puis calcule et affiche son âge.
Exemple d'exécution :

```
Quelle est ton année de naissance ? 1998
En 2020 tu as 22 ans
```

1.1.6 Identité

Écrire un programme qui demande à l'utilisateur son prénom, son nom, puis affiche son identité complète.
Exemple d'exécution :

```
Quel est ton prénom ? Léo
Quel est ton nom ? Dupont
Identité : Léo Dupont
```

1.1.7 Emploi du temps

Écrire un programme qui demande à l'utilisateur son nombre d'heures de cours par semaine, puis le nombre de semaines de cours, puis calcule le nombre d'heures totales et l'affiche exactement comme dans l'exemple ci-dessous :

```
Combien d'heures par semaine ?   6
Combien de semaines de cours ?   12
Total : 72 heures
```

1.1.8 Rectangle

Écrire un programme qui demande à l'utilisateur la longueur et la largeur d'un rectangle, et calcule et affiche son aire exactement comme dans l'exemple ci-dessous :

```
Largeur ?   3
Longueur ?  7
L'aire du rectangle de largeur 3 et de longueur 7 vaut 21
```

1.1.9 Puissance de 2

Écrire un programme qui demande à l'utilisateur un entier n, puis calcule et affiche 2 à la puissance n, exactement comme dans l'exemple d'exécution ci-dessous :

```
Entier ? 3
2 puissance 3 = 8
```

1.2 Instruction conditionnelle `if` et booléens

1.2.1 Un peu de logique

Compléter ci-dessous avec le nombre de valeurs de l'entier A pour lesquelles les expressions suivantes sont vraies (et spécifier quelles valeurs), comme dans l'exemple.

```
A == 8 or A == 9
-> 2 (8 et 9)

A > 0 and A < 20
-> 19 (Les entiers compris entre 1 et 19 inclus)

A >= 3 and A < 5
-> ...

A >= 3 or A < 5
-> ...

A > 0 and A < 10 and A % 3 == 0 and A % 2 != 0
-> ...

A <= 0 and A >= 20
-> ...
```

1.2.2 Expressions booléennes

On considère qu'on a initialisé les variables a,b,c de type entier. Écrire des expressions booléennes représentant les faits suivants :

1. a est impair
2. (au moins) un des 3 entiers a,b,c est impair
3. (exactement) un seul des 3 entiers a,b,c est impair
4. a est multiple de b
5. a,b,c sont compris entre 0 (inclus) et 20 (exclus)
6. a est soit (strictement) négatif, soit positif pair
7. b est nul ou impair

1.2.3 Négations d'expressions booléennes

Donner la négation simplifiée (pas d'opérateur `not`) des expressions booléennes suivantes :

1. a<10 or a>15
2. a%3 != 0 and a>=20
3. a%b == 0 or b%a == 0
4. (a>=b or a>=c) and (a%2==0 or a%3==0)

Donner aussi la lecture en français de ces négations.

1.2.4 Comparaison de deux programmes

Les deux programmes suivants sont-ils semblables ? Justifier votre réponse.

```
rep = input("Etes-vous fumeur : ")

if (rep == "oui") or (rep == "OUI") :
    print("Vous devriez arreter")
else :
    print("Continuez ainsi")
    print("Merci de votre participation")
```

```
rep = input("Etes-vous fumeur : ")

if (rep == "oui") or (rep == "OUI") :
    print("Vous devriez arreter")
else :
    print("Continuez ainsi")
print("Merci de votre participation")
```

1.2.5 Comparaison de deux programmes (bis)

Les deux programmes suivants sont-ils semblables ? Justifier votre réponse.

```
n = int(input("Entrez n : "))

if n%2 == 1 :
    n = 3*n+1
else :
    n = n//2

print(n)
```

```
n = int(input("Entrez n : "))

if n%2 == 1 :
    n = 3*n+1
if n%2 == 0 :
    n = n//2

print(n)
```

1.2.6 Bonjour (v2)

Écrire un programme qui :

1. Demande à l'utilisateur son prénom
2. Demande à l'utilisateur sa langue (français ou anglais)
3. Si l'utilisateur parle français, lui affiche "bonjour ..." (avec son prénom)
4. Si l'utilisateur parle anglais, lui affiche "hello ..." (avec son prénom)
5. Si l'utilisateur saisit une autre langue, affiche un message d'erreur

1.2.7 Ou exclusif

1. Écrire l'expression booléenne correspondant au `ou_exclusif` entre 2 booléens `a` et `b`. On rappelle que le OU EXCLUSIF signifie que exactement l'un des 2 booléens est vrai (mais pas les 2 en même temps).
2. Faire sa table de vérité

1.2.8 Lois de De Morgan

Utiliser des tables de vérité pour prouver les lois de De Morgan (négation des conjonctions et disjonctions, cf cours).

1.2.9 [INF104] Compatibilité de valeurs

Lorsque l'on compare un modèle et une mesure expérimentale, il faut toujours tenir compte de l'incertitude sur cette mesure.

1. Écrire un programme qui demande à l'utilisateur la valeur calculée par un modèle, la valeur mesurée, et son incertitude (nombres réels), et affiche "`compatible`" si la valeur du modèle est à moins de deux fois l'incertitude de la valeur mesurée, ou "`incompatible`" sinon.
2. Modifier le programme de façon à ce qu'il affiche "`compatible`" lorsque l'écart est inférieur à une fois l'incertitude, "`incompatible`" s'il est supérieur à trois fois l'incertitude, "`incertain`" sinon.
3. Modifier le programme pour permettre cette fois la comparaison de deux valeurs expérimentales, sachant que $\Delta_{(a-b)} = \sqrt{\Delta_a^2 + \Delta_b^2}$. On considérera les valeurs compatibles lorsque leur différence est compatible avec zéro.

1.2.10 Maximum de trois nombres

Écrire un programme qui demande trois entiers et qui affiche le plus grand des trois.

- Version 1 : demander d'abord les 3 nombres, puis les comparer entre eux. Combien y a-t-il de cas à considérer (donc de comparaisons faites) ?
- Version 2 : garder en mémoire un maximum temporaire, et lui comparer chaque nouvel entier lu. Dans ce cas combien fait-on de comparaisons ?
- Bonus avec une boucle : lire n entiers et afficher le maximum et le minimum.

1.2.11 Année bissextile

Écrire un programme qui détermine si une année saisie par l'utilisateur est bissextile ou non en affichant "L'année XXX est bissextile" ou "L'année XXX n'est pas bissextile" selon le cas. On rappelle que : une année est dite bissextile si c'est un multiple de 4, sauf si c'est un multiple de 100 ; toutefois, elle est considérée comme bissextile si c'est un multiple de 400.

1.2.12 Jours, heures, minutes, secondes

1. Écrire un programme en Python qui lit un nombre de secondes (entier) au clavier et puis affiche cette durée en jours, heures, minutes et secondes au format de l'exemple suivant :

   ```
   Donner un nombre de secondes : 93901
   La durée saisie se décompose en : 1 jour(s), 2 heure(s), 5 minute(s), 1 seconde(s)
   ```

2. Ajouter la gestion des cas particuliers : si une valeur vaut 0 alors on n'affiche pas cet élément, si elle vaut 1 on l'affiche au singulier, sinon on l'affiche au pluriel.

   ```
   Donner un nombre de secondes : 93900
   La durée saisie se décompose en : 1 jour, 2 heures, 5 minutes
   ```

1.2.13 Quiz

On veut écrire un programme de quiz (voir fonctionnement attendu ci-dessous).

1. Commencer par écrire un programme qui pose uniquement la première question et teste la réponse de l'utilisateur. Attention : combien y a-t-il de cas à envisager ?
2. Compléter maintenant ce programme pour poser les 3 questions successivement.
3. Compléter ce programme pour compter le nombre de bonnes réponses et l'afficher à la fin.
4. Bonus avec boucle while : modifier ce programme pour qu'il propose à l'utilisateur de rejouer à chaque fin de partie ; tant que l'utilisateur accepte, le programme démarre un nouveau quiz avec 3 nouveaux entiers ; quand l'utilisateur refuse, le programme se termine.
5. Bonus avec fonctions : écrire une fonction par question (qui pose la question, lit la réponse, et renvoie le résultat de l'utilisateur (juste ou faux) ; les appeler dans le programme principal qui gère le quiz.

Fonctionnement attendu Le programme demande trois entiers a, b, c à l'utilisateur, puis lui pose successivement les questions suivantes :

- $a < b < c$?
- un seul nombre impair parmi a, b, c ?
- a, b, c distincts deux à deux ?

Pour chaque question, le programme doit contrôler la réponse, et doit afficher "réponse correcte, Bravo" ou "réponse incorrecte" selon le cas ; il doit aussi compter le nombre de bonnes réponses et, à la fin, afficher le score sous la forme : "x bonnes réponses sur 3".

Exemple d'exécution du programme (conformez vous strictement à ce modèle).

```
Donner trois entiers :
a=? 1 (Entrée)
b=? 4 (Entrée)
c=? 3 (Entrée)

Jeu du "Vrai ou Faux ?" (répondez en tapant V ou F)

Question 1.
1 < 4 < 3 ?
V (Entrée)
```

```
réponse incorrecte

Question 2.
y a-t-il un seul nombre impair parmi 1, 4, 3 ?
F (Entrée)
réponse correcte, Bravo

Question 3.
1, 4, 3 sont-ils distincts deux à deux ?
V (Entrée)
réponse correcte, Bravo

2 bonnes réponses sur 3
Le jeu est terminé : Au revoir !
```

1.2.14 Multiples

Écrire un programme qui demande à l'utilisateur un entier entre 1 et 100. Si l'entier ne respecte pas ces bornes, le programme se termine sur un message d'erreur. Sinon il :

1. affiche uniquement le premier entier entre 2 et 10 dont x est un multiple (plus petit diviseur)
2. affiche tous les entiers entre 2 et 10 dont x est un multiple

Quelle est la différence entre les 2 versions ? *Remarque : on verra bientôt comment écrire des boucles qui permettent d'éviter de telles répétitions de code.*

1.2.15 Dé pipé

On utilisera ici le module `random` vu en cours. Écrire un programme qui joue une partie de dés entre 2 joueurs :

- demande les prénoms des 2 joueurs
- pour le premier joueur, on utilise un vrai dé 6 : tirer un nombre au hasard entre 1 et 6 et l'afficher sous la forme "JJJ fait XXX" (en remplaçant JJJ par le prénom et XXX par le tirage)
- pour le 2e joueur, on utilise un dé pipé : tirer un nombre au hasard entre 1 et 3, l'afficher sous la même forme
- déterminer le vainqueur (celui qui a fait le plus gros tirage) et afficher son prénom sous la forme "JJJ gagne"

Extension avec d'autres dé pipés :

- Écrire une instruction qui ne tire que des faces impaires d'un dé 6
- Écrire une instruction qui tire 1 avec une probabilité de 30%, 2 avec une probabilité de 20%, 3 ou 4 avec une probabilité de 15% chacun, 5 ou 6 avec une probabilité de 10% chacun.

1.2.16 Encore des expressions booléennes

On suppose que toutes les variables nécessaires sont correctement initialisées : a et b sont des entiers, c et rep sont des chaînes de caractères, on suppose qu'on sait que c est de longueur 1. 1. Écrire les expressions booléennes qui représentent les assertions suivantes :

1. `a` est un entier pair
2. `a` est un multiple de 3 ou un multiple de 7
3. `b` est un entier strictement négatif multiple de 3
4. `c` est une lettre minuscule de l'alphabet
5. `c` est une lettre de l'alphabet (majuscule ou minuscule)
6. `c` est un caractère numérique
7. `rep` n'est ni la lettre `'o'` ni la lettre `'n'`
8. l'entier `b` n'est pas compris entre 0 et 20 inclus

2. Écrire la négation des expressions booléennes ci-dessus, et les simplifier autant que possible (c'est-à-dire qu'on ne veut pas utiliser l'opérateur `not`).

1.3 Itération conditionnelle `while`

Bases des itérations conditionnelles

1.3.1 Code à trous

Soit le programme incomplet suivant :

```
i = <A>
while <B>:
    print(i)
    i = <C>
```

Donner les expressions <A>, <B>, <C> pour que :

1. Le programme affiche les entiers allant de 0 à 10.
2. Le programme affiche les entiers allant de 10 à 0.
3. Le programme affiche les multiples de 3 strictement inférieurs à 30 (0 compris).
4. Le programme affiche les puissances de 3 strictement inférieures à 300 (1 compris).
5. Le programme affiche un entier saisi par l'utilisateur tant que celui-ci est strictement positif.
6. Le programme affiche une lettre saisie par l'utilisateur tant que celle-ci est une voyelle.
7. Le programme affiche une lettre saisie par l'utilisateur jusqu'à ce que celle-ci soit une voyelle.

1.3.2 Lecture de code

Combien de '*' les programmes suivants affichent-ils ? Justifier les réponses.

```
# (a)
i = 0
while i <= 15 :
    print('*')
    i = i + 1
```

```
# (b)
i=0
while i>10 :
    print('*')
    i=i+1
```

```
# (c)
i=20
while i<100 :
    if i%2!=0 :
        print('*')
    i=i+1
```

```
# (d)
i = 0
while i < 100 :
    if i % 2 == 0 :
        print('*')
        i = i + 1
```

```
# (e)
i=1
while i<=16 :
    print('*')
    i=i*2
```

```
# (f)
i=0
while i<10 :
    print('*')
    i=i-1
```

Bonus : le(s)quel(s) de ces programmes devrai(en)t être modifié(s), pourquoi, et comment ?

1.3.3 Fizz Buzz

Écrire un programme qui affiche tous les entiers entre 1 et 1000, mais qui remplace les multiples de 3 par `"fizz"`, les multiples de 5 par `"buzz"`, et les multiples à la fois de 3 et 5 par `"fizzbuzz"`. Exemple d'exécution (début) :

```
1
2
fizz
4
buzz
fizz
7
8
fizz
buzz
11
fizz
13
14
fizzbuzz
16
17
...
```

Boucle while pour filtrer les entrées

1.3.4 Dé pipé (le retour)

Écrire un programme qui tire un nombre au hasard jusqu'à obtenir un nombre pair. Il affiche alors le nombre pair tiré. Bonus : afficher aussi le nombre de tirages nécessaires.

1.3.5 Voyelle

Écrire un programme qui demande à l'utilisateur de saisir une lettre et qui continue jusqu'à ce que l'utilisateur saisisse une voyelle.

1. Version 1 : avec instruction `break`
2. Version 2 : sans instruction `break`

1.3.6 Mot de passe

1. Écrire un programme dans lequel vous définissez un mot de passe (par exemple `mdp = 'abcdef'`), puis qui demande à l'utilisateur de saisir un mot de passe pour accéder. Si l'utilisateur saisit le bon mot de passe, le programme affiche le message 'Acces autorise' et se termine, sinon il affiche 'Acces refuse' et redemande.
2. Modifier ce programme pour que l'utilisateur n'ait droit qu'à 5 essais pour saisir le mot de passe (après 5 erreurs, accès refusé).
3. Si vous avez utilisé `break` dans la 2e version, réécrire le programme sans l'utiliser.

Boucle while, accumulateurs et drapeaux

1.3.7 Accumulateurs simples

Écrire un programme qui :

1. Calcule et affiche la somme des n premiers entiers
2. Calcule et affiche la somme des n premiers entiers impairs
3. Calcule et affiche la somme de n entiers lus au clavier
4. Calcule et affiche le produit de n entiers lus au clavier
5. Lit n paires d'entiers, compare les entiers 2 à 2, et affiche à la fin un booléen indiquant si toutes les paires étaient en ordre croissant (cf exemple pour n=3).

```
Paire d'entiers #1
3
7
Paire d'entiers #2
2
12
Paire d'entiers #3
5
9
Toutes les paires sont en ordre croissant
...
Paire d'entiers #3
9
5
Toutes les paires ne sont pas en ordre croissant
```

1.3.8 Moyenne, minimum, maximum

Écrire un programme qui demande à l'utilisateur de saisir des nombres réels (float) correspondant à ses notes d'examen. L'utilisateur donnera une note invalide (non comprise entre 0 et 20) pour indiquer la fin de la saisie (cette note invalide ne sera pas comptabilisée). Après la fin de la saisie le programme doit afficher la moyenne des notes (valides) saisies (*attention* au cas spécial où aucune note valide n'est saisie).

Compléter ce programme pour afficher aussi la meilleure note et la pire note de l'utilisateur.

1.3.9 Factorielle

Écrire un programme qui lit un entier n saisi par l'utilisateur, et qui affiche n ! (factorielle de n).
Rappel : n ! = 1 x 2 x 3 x 4 x . . . x n
Indice : Attention à la valeur initiale.
Bonus avec une boucle imbriquée : proposer à l'utilisateur de rejouer avec une nouvelle valeur de n.

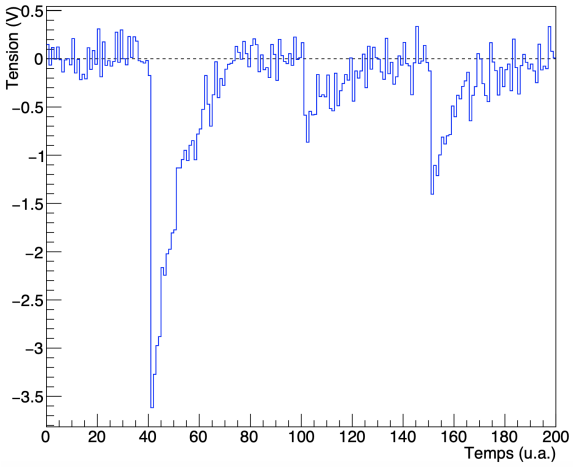
1.3.10 Séquence croissante

Écrire un programme qui lit une suite d'entiers au clavier jusqu'à ce que l'utilisateur saisisse un nombre strictement négatif. Le programme affichera alors 'Croissante' si la suite d'entiers était croissante, 'Décroissante' si la suite était décroissante, 'constante' si tous les entiers étaient égaux, et 'Ni croissante ni décroissante' si la suite n'est ni croissante ni décroissante. *Indice :* utiliser des drapeaux.

1.3.11 [INF104] Détection de particule

Mots-clés : *photomultiplicateur — plastique scintillant — convertisseur analogique-numérique — détection de pic*

Dans une expérience de détection de particules cosmiques, on souhaite ajouter un programme qui détecte automatiquement le signal correspondant au passage d'une particule, de façon à ne pas stocker de données inutiles. Le signal issu d'un photomultiplicateur, illustré dans la figure ci-contre, consiste en une brusque baisse de la tension, suivie d'une lente remontée, et se superpose aux fluctuations aléatoires dues au bruit de fond.
Le système d'acquisition inclut une transformation de ce signal électrique (analogique) en données numériques : un entier compris entre 0 et 2^{12} pour chaque intervalle de temps Δt. On supposera ici que 0 correspond à -4 V et 2^{12} à 1 V.



1. Écrire un programme qui lit séquentiellement cette série de nombres et utilise une condition sur l'amplitude du signal pour détecter l'instant de passage d'une particule. Le programme affichera par exemple "`déclenchement`" au moment de la détection.

2. Améliorer la discrimination entre le signal voulu et le bruit en imposant que le déclenchement ne se fasse que lorsque la tension descend rapidement, puis remonte lentement sous forme d'une suite de valeurs croissantes.

3. Bonus : améliorer cette méthode de détection en tenant compte des petites oscillations possibles sur la phase de remontée de la tension du fait du bruit.

Boucle while pour rejouer un programme

1.3.12 Nombres premiers (simple)

Écrire un programme qui demande à l'utilisateur de saisir un entier positif N, et qui vérifie si N est premier. (*Rappel :* un nombre est premier s'il n'a pas de diviseurs autres que 1 et lui même. Ce programme affiche le résultat, puis propose à l'utilisateur de recommencer avec un nouveau nombre, jusqu'à ce que celui-ci réponde 'non'. Exemple d'exécution :

```
Programme de test de nombres premiers
Entre un entier ? 7
7 est premier
Rejouer ? oui
Entre un entier ? 10
10 n'est pas premier
Rejouer ? non
Fin du programme
```

Question : combien de diviseurs faut-il tester avant de déterminer si N est premier ?

1.3.13 Dé pipé : multi-parties

On reprend l'exercice du dé pipé vu au TD précédent.

1. Écrire un programme qui oppose 2 joueurs, l'un lance un vrai dé 6, l'autre lance un dé pipé qui l'avantage (par exemple lancer 3 dés puis garder le maximum).

2. Modifier ce programme pour jouer N parties (par exemple 20) et afficher le nombre de victoires du joueur honnête (dé non pipé).

3. Modifier le premier programme pour jouer jusqu'à ce que le joueur honnête gagne, et afficher le nombre de parties qui auront été nécessaires.

Boucle while, suites et récurrence

1.3.14 Suite de Fibonacci

Soit la suite de Fibonacci donnée par

$$\begin{cases} f_0 = 1 \\ f_1 = 1 \\ f_n = f_{n-1} + f_{n-2} \end{cases}$$

1. Écrire un programme qui lit un entier n au clavier, puis calcule et affiche f_n

2. On peut estimer le nombre d'or en divisant un terme de la suite par le précédent. Plus n grandit et plus l'estimation est précise. Compléter ce programme pour afficher l'estimation du nombre d'or f_n/f_{n-1} (on affiche uniquement la dernière, pas les valeurs intermédiaires).

3. Bonus plus difficile : définir une constante $nbor = 1,618033988749895$ (approximation du nombre d'or). Modifier ce programme pour qu'il demande à l'utilisateur une précision souhaitée (distance maximale autorisée avec le nombre d'or, par exemple p = 0.01), puis il calcule les termes de la suite et l'estimation du nombre d'or *estimnbo* jusqu'à ce que la précision soit atteinte, c-à-d que la distance entre *nbor* et *estimnbo* est inférieure ou égale à p.

1.3.15 Syracuse

Soit la suite récursive donnée par $u_0 = A$ et par la relation de récurrence :

$$u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair} \\ 3 * u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

NB : la grande question est de savoir si, pour toute valeur initiale A cette suite contient l'élément 1. Les matheux n'ayant pas résolu ce problème, on dit que ce problème est ouvert. On dit aussi que "cette suite passe par 1 quel que soit le point de départ" est la conjecture de Syracuse.

1. Écrire un programme qui demande la saisie d'un entier strictement positif et détermine pour quel n on a : $u_n = 1$ (c'est la "durée de vol")

2. Modifier ce programme pour qu'il affiche tous les termes u_n calculés au fur et à mesure.

3. Modifier ce programme pour qu'il affiche le maximum atteint par la suite (la plus grande valeur d'un terme u_n).

4. Cette suite est donc croissante quand un terme est impair (et le suivant est pair), décroissante tant que les termes sont pairs. Bonus : modifier ce programme pour qu'il affiche la longueur de la plus longue descente (suite décroissante), en nombre de termes.

1.3.16 [INF104] Ensemble fractal de Mandelbrot

Mots-clés: *ensembles de Julia — systèmes dynamiques — chaos — attracteur de Lorenz*
Soit la famille de suites :

$$f_c : \begin{cases} z_n = 0 \\ z_{n+1} = z_n^2 + c \end{cases} \qquad \text{avec } c = x + iy \text{ un nombre complexe donné.}$$

L'ensemble de Mandelbrot est le lieu des points c du plan complexe tels que la suite associée f_c ne diverge pas en module. Par exemple le point c de coordonnées $(-1, 0)$ appartient à l'ensemble car la suite associée est constituée de la succession des valeurs 0, -1, 0, -1, dont le module ne diverge donc pas.

1. Écrire un programme qui demande les valeurs de x et y et détermine si la suite des modules de z_n est divergente, sachant que l'on est certain que la suite diverge à partir du moment où $|z_n| > 2$.

2. Écrire un autre programme, qui demande des nombres x_{min} et x_{Max}, découpe cet intervalle en 20, et pour chaque portion de l'intervalle affiche (à la suite sur une même ligne) un espace si $-1 < x_i < 1$ ou le caractère "`o`" sinon. Par exemple pour $[x_{min}; x_{Max}] = [-3; 5]$: `ooooo` `ooooooooo`.

3. Assembler ces deux programmes en un seul et apporter les ajouts nécessaires pour représenter en mode texte l'ensemble de Mandelbrot. La position sur la ligne sera la coordonnée x du plan complexe, la numéro de la ligne sera la coordonnée y. On utilisera 50 lignes et 100 colonnes, et on affichera le caractère "`#`" si la suite diverge en moins de 10 itérations, "`*`" en moins de 30, "`o`" en moins de 100, "`.`" en moins de 300. Au-delà, on considère que la suite ne diverge pas et on affiche un espace.

Vous pourrez essayer votre programme avec les valeurs ci-contre.

	x_{min}	x_{Max}	y_{min}	y_{Max}
Ensemble entier	-1,5	0,5	-1,0	1,0
Bord supérieur	-0,35	0,15	0,55	1,05
"Loin" du bord	-0,169	-0,149	1,023	1,043

Boucles imbriquées

1.3.17 Escaliers

Écrire un programme qui demande un entier L à l'utilisateur, et qui affiche la forme suivante, constituée de L lignes :

```
a. ****     b.  *      c.      *   d.  *****   e.  *****   f.      *
   ****         **            **       ****         ****         ***
   ****         ***          ***       ***           ***        *****
   ****         ****        ****       **             **       *******
   ****         *****      *****       *               *      ****o****
```

Dans la figure e, la porte (caractère 'o') est en bonus, pour simplifier vous pouvez afficher une '*' à la place. Dans cet exercice on s'interdira d'utiliser l'opérateur de multiplication pour créer les chaînes à afficher, on n'affichera qu'une étoile à la fois.

1.3.18 Triangles d'entiers

Écrire un programme qui demande à l'utilisateur un nombre de lignes, et affiche un triangle d'entiers avec ce nombre de lignes, sous la forme suivante (dans les exemples avec n=4)

```
1               1             1 2 3 4         1 2 3 4
1 2             2 3           1 2 3           5 6 7
1 2 3           4 5 6         1 2             8 9
1 2 3 4         7 8 9 10      1               10
  a)              b)            c)              d)

10 9 8 7          1               1
 6 5 4          1 2             2 3
 3 2          1 2 3           4 5 6
 1          1 2 3 4         7 8 9 10
  e)            f)              g)
```

1.3.19 Moyenne de classe

On veut calculer la moyenne d'un groupe de 30 étudiants. Pour ce faire, on veut écrire un programme qui, pour chaque étudiant du groupe, demande à l'utilisateur de saisir 10 notes. A la fin de la saisie de ces 10 notes, le programme doit afficher la moyenne de l'étudiant. A la fin du programme, la moyenne de tout le groupe doit être affichée.

Bonus : afficher aussi en fin de programme la moyenne et le numéro de l'étudiant qui a la meilleure moyenne.

1.3.20 Pyramide ailée

Écrire un programme qui demande un entier n à l'utilisateur et affiche une pyramide ailée de hauteur n comme sur la figure suivante :

1.3.21 Nombres premiers (bis)

Le bout de programme suivant vérifie si un nombre N est premier et met le résultat dans le drapeau `est_premier`.

```
est_premier=True
i = 2
while i * i <= N and est_premier:
    if N % i == 0:
        est_premier=False
    i = i + 1
```

1. Comprendre et expliquer le fonctionnement de ce code
2. L'utiliser pour écrire un premier programme qui affiche les 100 premiers nombres premiers. Bonus : les afficher sur une seule ligne, séparés par des virgules. Bonus : éviter la virgule finale.
3. L'utiliser pour écrire un 2e programme qui :
   - Lit deux entiers positifs A et B
   - Filtre ces entiers pour qu'ils soient positifs, et que B soit supérieur ou égal à A
   - Affiche 'Oui' s'il existe au moins un nombre premier entre A et B inclus (et affiche ce nombre premier), et 'Non' sinon.

1.4 Fonctions

1.4.1 Comprendre les fonctions déjà connues

Pour chacun des appels de fonction ci-contre :

1. Quel est le nom de la fonction ?
2. Combien y a-t-il d'arguments ?
3. Pour chaque argument (s'il y en a), donnez son type et sa valeur.
4. Y a-t-il une valeur de retour, et si oui laquelle (valeur et type) ?
5. Y a-t-il des "effets de bord" (interactions entrées/sorties, dessin turtle...) provoqués par l'appel ?

```
import math
import random
import turtle

n = int(input("Borne sup ?"))
x = random.randint(0,n)
print("Nombre aleatoire: ", x)

text=input("Quel est votre nom?")
print("Bonjour",text)

turtle.up()
turtle.forward(100)
turtle.down()
turtle.circle(50.5)

z=math.sqrt(5.5)
print("La racine de 5.5 est", z)
```

1.4.2 Lecture

Prédire le fonctionnement des 3 programmes ci-dessous.

Programme 1

```
def f(x,y):
    c=x**2+y**2
    return c
# programme principal
a=3
b=4
d=f(a,b)
print("d=", d)
d=f(4,5)
print("d=", d)
```

Programme 2

```
def est_voyelle(l):
    if l=="a" or l=="e" or l=="i" or l=="o" or l=="u" or l=="y":
        return True
    else:
        return False
def demande_voyelle():
    reponse=""
    while not(est_voyelle(reponse)):
        reponse=input("Choisissez une lettre : ")
        if not(est_voyelle(reponse)):
            print("Ceci n'est pas une voyelle.")
    return reponse
```

```
# programme principal
l=demande_voyelle()
print("La voyelle que vous avez choisie est:", l)
demande_voyelle()
```

Programme 3

```
def somme_premiers_entiers(n, modeAffichage):
    i=1
    somme=0
    while i<n:
        somme=somme+i
        if modeAffichage: # pareil que if modeAffichage==True
            print("0+ ... +", i, "=", somme)
        i=i+1
    return somme
# programme principal
s=somme_premiers_entiers(5,False)
print("s=", s)
modeAffichage=True
s=somme_premiers_entiers(5,modeAffichage)
print("s=", s)
```

1.4.3 Calcul de polynome

Écrire une fonction qui reçoit 3 entiers a,b,c et un réel x et qui calcule et renvoie la valeur du polynôme ax^2+bx+c. Écrire ensuite un programme principal qui :

- demande à l'utilisateur 2 réels y et z ; utilise la fonction ci-dessus pour calculer les polynômes $3y^2-4y+7$ et $7z^2-10z-5$; affiche les valeurs avec un message clair
- tire au hasard 3 entiers h_1, h_2, h_3 (rappel : fonction `randint` du module `random`) entre -10 et 10, et utilise la fonction ci-dessus pour calculer la valeur du polynôme $9h_1-3h_2+h_3$, puis affiche la valeur

1.4.4 Dates

Écrire une fonction qui reçoit un jour, un mois et une année (3 entiers) et qui teste si la date correspondante est correcte. Cette fonction renvoie donc un booléen. Par exemple le 31/2/2012 n'est pas correct. Le 20/7/2015 est correct.

On pourra écrire une fonction auxiliaire qui teste si un numéro de jour donné est correct pour un mois donné.

1.4.5 Somme des chiffres

Écrire une fonction qui reçoit un entier (par exemple 37251) et renvoie la somme de tous ses chiffres (ici : 3+7+2+5+1=18). On a donc besoin pour commencer de savoir combien de chiffres a ce nombre :

1. Méthode 1 (facile) : on peut mesurer la longueur de la chaîne de caractères correspondante pour trouver le nombre de chiffres.
2. Méthode 2 : on s'interdit de convertir l'entier en chaîne. *Indice* : utiliser l'opérateur modulo pour trouver son nombre de chiffres.

1.4.6 Conversions binaire-décimal

Dans cet exercice on s'intéresse à convertir des entiers entre la base 2 (écriture binaire) et la base 10 (écriture décimale). **Remarque** : Python fournit des fonctions prédéfinies permettant de faire ces conversions (mais on ne veut pas les utiliser dans cet exercice).

- La fonction `bin(x)` renvoie une chaîne de caractères représentant l'écriture binaire de l'entier x. Par exemple `bin(11)` renvoie la chaîne '0b1011', le préfixe '0b' indiquant qu'il s'agit d'une écriture binaire.
- La fonction `int(chaine,base)` renvoie l'entier décimal représenté par la chaîne de caractères donnée dans la base donnée. Par exemple `int('1010',10)` interprète la chaîne '1010' en base 10 et renvoie donc l'entier décimal 1010 ; `int('1010',2)` interprète la chaîne '1010' en base 2 (en binaire) et renvoie donc l'entier décimal 10.

Vous pourrez cependant vous servir de ces fonctions pour vérifier vos résultats.

1. Écrire une fonction qui prend en argument un entier n, et renvoie une chaîne de caractères représentant son écriture en binaire. *Par ex. cette fonction appelée sur l'entier 9 renvoie la chaîne de caractères "1001".*

2. Écrire ensuite la fonction inverse, qui reçoit une chaîne de caractères représentant un entier en binaire, et qui renvoie un entier égal à sa valeur en base 10. Par exemple cette fonction appelée sur la chaîne "1100" renvoie l'entier 12 ; avec la chaîne "11001" elle renvoie l'entier 25.

3. Écrire enfin un programme principal qui effectue la boucle suivante :

   - propose à l'utilisateur 3 choix : conversion binaire→décimal, décimal→binaire, ou arrêt
   - s'il choisit d'arrêter, la boucle se termine
   - s'il choisit de faire une conversion décimal vers binaire, le programme lui demande un entier et affiche sa conversion,
   - s'il choisit de faire une conversion binaire vers décimal, le programme lui demande une chaîne de caractères et affiche sa conversion en entier décimal
   - puis le programme recommence jusqu'à ce que l'utilisateur choisisse d'arrêter

1.4.7 Maximum

Note : Il existe déjà une fonction Python pour calculer le maximum, elle s'appelle `max`*. Cependant, dans cet exercice, nous vous demandons de ne pas l'utiliser.*

1. Écrire une fonction `maximum` qui prend en arguments deux nombres et qui renvoie le plus grand des deux.

2. Écrire un programme principal qui demande deux entiers à l'utilisateur, puis qui affiche `Le plus grand des deux est: ...`. Vous ferez un **appel** à votre fonction `maximum`.

3. Écrire une fonction `maximum3` qui prend en arguments trois nombres et qui renvoie le plus grand des trois. Vous ferez appel à votre fonction `maximum`.

4. Écrire une fonction `maximum3_input` qui ne prend pas d'arguments, qui demande trois nombres à l'utilisateur puis qui renvoie le maximum des 3 nombres. Vous ferez appel à votre fonction `maximum3`.

5. Écrire un programme principal utilisant `maximum3_input`, qui demande trois nombres à l'utilisateur puis qui affiche le plus grand des trois.

6. Écrire une fonction `max_input` qui lit des entiers strictement positifs au clavier (jusqu'à lire un entier nul qui signifie fin de saisie), et en fin de série affiche le maximum lu.

1.4.8 Moyenne pondérée

1. Écrire une fonction `moyenne_ponderee` qui prend en arguments quatre nombres que l'on appellera `note1`, `note2`, `coeff1` et `coeff2` et qui calcule la moyenne pondérée de `note1` et `note2`, ayant pour coefficients respectifs `coeff1` et `coeff2`.

2. Écrire un programme principal qui demande deux notes puis deux coefficients à l'utilisateur, puis qui affiche la moyenne pondérée. Vous ferez un appel à votre fonction `moyenne_ponderee`.

3. Bonus : modifier ce programme pour qu'il demande à l'utilisateur le nombre de notes (n, par exemple 2 ci-dessus), puis la saisie des n notes et n coefficients, puis affiche la moyenne pondérée. *Indice* : utiliser une boucle.

1.4.9 Dé

1. Écrire une fonction qui prend un argument un entier et qui teste si la valeur donnée est celle d'un dé (entre 1 et 6). Si oui, la fonction affiche `Valeur correcte` et renvoie `True`, sinon la fonction affiche `Valeur incorrecte` et renvoie `False`.

2. Écrire un programme principal qui demande à l'utilisateur une première valeur de dé puis qui teste si la valeur est correcte grâce à la fonction précédente (avec l'affichage correspondant) ; puis, si la première valeur était correcte, le programme doit demander une deuxième valeur de dé et la vérifier (avec l'affichage correspondant) ; enfin, si les deux valeurs étaient correctes, le programme doit afficher la somme des deux dés.

3. On décide finalement de ne pas afficher `Valeur correcte` si la valeur du dé testée est effectivement correcte (mais on garde l'affichage si la valeur est incorrecte). Quelle partie de votre code doit être modifiée ?

1.4.10 Factorielle qui dépasse

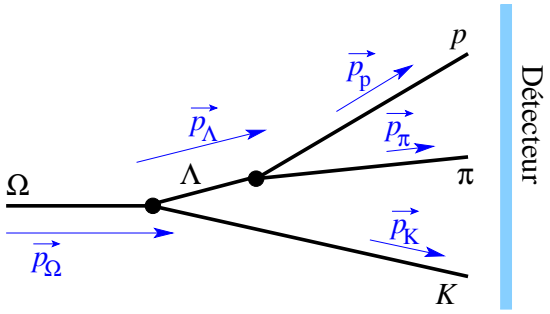
Écrire une fonction `depasse` qui prend en argument un entier `A` et qui renvoie le plus petit entier n tel que $n!$ soit supérieur ou égal à `A`, i.e. le plus petit entier n dont la factorielle atteint ou dépasse A. *Rappel :* $n! = 1 \times 2 \times 3 \times \cdots n$

1.4.11 [INF104] Étude du plasma de quarks et de gluons

Mots-clés: *plasma de quarks et de gluons — interaction forte — interaction faible — hadron — distribution en masse invariante*

L'accélérateur de particules LHC du CERN permet de déposer une énergie de plus de 0,1 mJ dans un volume de la taille d'un noyau d'atome, et ainsi de recréer l'état de l'Univers lors de sa première microseconde d'existence : le plasma de quarks et de gluons.

La particule Ω^-, qui a joué un rôle majeur dans la découverte de ce milieu, permet de l'étudier, mais elle ne peut être détectée directement car elle se désintègre en moins de 0,1 ns. Les particules filles de la désintégration sont un K^-, qui peut arriver aux détecteurs, et un Λ^0, qui se désintègre ensuite lui-même en p et π^-. Beaucoup d'autres particules étant présentes en même temps, il faut trouver une astuce pour reconnaître celles qui proviennent effectivement d'un Ω^-.



Les lois de conservation permettent de calculer la masse d'une particule parente à partir de l'énergie et de la quantité de mouvement mesurées des particules-filles : si $A \to B + C$, alors :

$$m_A c^2 = \sqrt{(E_A + E_B)^2 - (\overrightarrow{p_A} c + \overrightarrow{p_B} c)^2} \qquad \text{sachant que :} \quad \overrightarrow{p}^2 = p_x^2 + p_y^2 + p_z^2$$

Si la masse obtenue ne correspond pas à la masse de la particule A, alors les deux particules "assemblées" n'ont en fait pas de lien entre elles.

Dans notre cas, l'énergie des particules filles n'est pas mesurée, mais déduite de leur impulsion et de leur masse par $E_i = \sqrt{(\overrightarrow{p_i} c)^2 + (m_i c^2)^2}$.

1. Écrire une fonction `calcE` qui calcule une énergie à partir d'une masse et des trois coordonnées d'impulsion.
2. Écrire une fonction `calcMassParent` qui calcule la masse d'une particule supposée parente à partir des impulsions de deux particules-filles.
3. Écrire un programme qui lit les impulsions de 3 particules filles et teste si elles peuvent provenir de la désintégration d'un Ω^-.
4. Dans le même programme, ajouter d'autres tests en se basant sur le tableau ci-dessous.

Particule	Décroissance	mc^2 (MeV)	Particule	Décroissance	mc^2 (MeV)
π^-	Détecté	140	Ξ^-	$\Lambda^0\pi^-$	1321
K^-	Détecté	494	Ω^-	$\Lambda^0 K^-$	1672
p	Détecté	938	D^0	$K^-\pi^+$	1865
Λ^0	$p\pi^-$	1116	Λ_c^+	$\Xi^- K^+\pi^+$	2285

1.4.12 Calculatrice

Dans cet exercice, nous allons nous interdire d'utiliser les opérateurs mathématiques de Python, ainsi que les fonctions du module math. A partir d'une seule fonction de départ, nous allons tenter de recréer tous les opérateurs arithmétiques nous mêmes.

1. Écrire une fonction *soustraction(a, b)* qui reçoit 2 entiers a et b, et renvoie leur soustraction (a-b). Ici on peut utiliser l'opérateur -. C'est notre fonction de départ, à partir de laquelle nous allons recréer les autres. Cette fonction nous suffira pour définir toute l'arithmétique de base pour les entiers et plus encore !
2. En utilisant la fonction *soustraction*, définir la fonction *addition(a, b)*, qui calcule la somme de ses arguments. On rappelle qu'il ne faut utiliser aucun opérateur de Python (y compris l'opérateur -) mais uniquement la fonction soustraction.
3. En utilisant la fonction *addition*, définir la fonction *multiplication(a, b)* qui calcule le produit de ses arguments.
4. En utilisant les fonctions précédemment définies, écrire une fonction *division(a, b)* qui retourne le quotient de la division de a par b.

5. Bonus avec fonctions avancées : écrire à la place une fonction *division(a, b)* qui retourne deux valeurs : le quotient de la division entière de a par b ainsi que le reste de cette division.

6. En utilisant la fonction `multiplication` définie ci-dessus, définir les fonctions *puissance(a, n)* et *factorielle(a)*.

À présent, nous allons écrire un programme principal qui fonctionne comme une calculatrice. Le programme commence par lire un premier entier A, suivi d'un opérateur. Si (et seulement si) l'opérateur requiert un autre opérande, un autre entier B est lu. Si l'opérateur ne requiert pas d'autre opérande (exemple : factorielle), le résultat est affiché directement. Après l'affichage du résultat, le programme se relance et attend que l'utilisateur exécute une nouvelle opération. Les opérateurs qu'on veut supporter sont les suivants :

- "+" : addition
- "-" : soustraction
- "*" : multiplication
- "/" : division
- "**" : puissance
- "!" : factorielle

Exemples d'exécution :

```
A=5
+
B=8
13

A=9
-
B=12
-3

A=12
/
B=5
(2, 2)

A=13
/
B=5
(2, 3)

A=5
*
B=9
45

A=5
!
120

A=2
**
B=6
64

A=5
$
Opérateur inconnu, on recommence !

A=5
**
B=5
3125
```

1.5 Caractères et chaînes de caractères

Remarque : les exercices suivants demandent parfois d'écrire des fonctions. Si vous n'avez pas encore vu les fonctions, vous pouvez à la place écrire un programme principal qui fait la même chose. Les exercices qui nécessitent des boucles peuvent être réalisés avec des boucles `while`, ou des boucles `for` si vous les avez déjà vues.

Remarque2 : Python dispose déjà de fonctions `isalpha`, `lower`, `upper`, `islower`, `isupper`, `capitalize`, mais on s'interdit de les utiliser ici.

Caractères

1.5.1 Booléens et caractères

Écrire les expressions booléennes correspondant aux assertions suivantes (on suppose toutes les variables correctement initialisées, x et y sont des chaînes de caractères, n est un entier) :

- La variable x est une lettre majuscule de l'alphabet
- La variable y est une voyelle en majuscule
- Les variables x et y sont des lettres minuscules de l'alphabet
- x est strictement avant y dans l'ordre alphabétique
- x contient entre 3 et 7 lettres (inclus)
- x est une lettre minuscule, et y est la même lettre en majuscule
- n est le code ASCII d'une lettre majuscule de l'alphabet

1.5.2 Pyramide alphabétique

Écrire un programme qui affiche l'alphabet sous forme d'un triangle (en complétant par des tirets après la fin de l'alphabet).

```
a                               a
b c                           b c
d e f                       d e f
g h i j                   g h i j
k l m n o               k l m n o
p q r s t u           p q r s t u
v w x y z - -       v w x y z - -
```

1.5.3 Caractère alphabétique

Écrire un programme / une fonction qui reçoit une chaîne de caractères et teste si c'est une lettre de l'alphabet (majuscule ou minuscule). Cette fonction renvoie un booléen (vrai pour une lettre, qu'elle soit majuscule ou minuscule ; faux pour un autre caractère ou pour une chaîne de plusieurs caractères).

Remarque : il faut donc vérifier d'abord si la chaîne est bien de longueur 1, c'est-à-dire ne comprend qu'un seul caractère. Rappel : la fonction `len` donne la longueur d'une chaîne ou d'une liste.

1.5.4 Minuscule ou majuscule ?

1. Écrire un programme qui lit au clavier une lettre de l'alphabet, et teste si elle est en minuscule ou en majuscule. Remarque : Python fournit déjà des méthodes `isupper()` et `islower()` sur les chaînes de caractères, mais on demande de ne pas les utiliser.

2. Modifier ce programme pour qu'il propose de rejouer avec une nouvelle lettre.

3. Bonus avec fonctions : à la place d'un programme, écrire une fonction `est_minuscule` qui reçoit la lettre en paramètre et renvoie un booléen indiquant si elle est minuscule (True) ou majuscule (False).

1.5.5 Position dans l'alphabet

1. Écrire un programme qui lit un caractère et affiche sa position dans l'alphabet, ou `-1` si ce n'est pas une lettre. *Remarque : la lettre b en minuscule ou B en majuscule ont la même position, à savoir 2. Rappel : la fonction prédéfinie* `ord` *reçoit un caractère et renvoie son code ASCII.*

2. Écrire ensuite un programme qui lit un entier (entre 1 et 26) et un caractère ('m' pour minuscule ou 'M' pour majuscule) et affiche la lettre située à cette position dans l'alphabet, en minuscule/majuscule selon. *Indice : la fonction prédéfinie* `chr` *fait la conversion inverse, du code ASCII vers le caractère correspondant.*

3. Bonus avec fonction : au lieu d'un programme principal, écrire 2 fonctions qui reçoivent le caractère, ou l'entier pos et un booléen maj, en paramètre, et renvoient le résultat. Écrire un programme principal qui appelle ces fonctions et affiche le résultat.

1.5.6 Décalage

Écrire une fonction qui reçoit un caractère `x` et un entier `n` :

- Si ce caractère n'est pas une lettre, il est renvoyé tel quel
- Si c'est une lettre de l'alphabet (majuscule ou minuscule), la fonction renvoie le caractère situé `n` positions après `x` (sans changer la casse : une majuscule reste une majuscule).
- Si on dépasse la lettre `'z'` il faut revenir à `'a'` ; si on dépasse le 'Z' il faut revenir à 'A'. Par exemple 3 positions après `'y'` on veut obtenir `'b'` ; 3 positions après 'Y' on veut obtenir 'B'.

1.5.7 Inversion de casse

Écrire une fonction qui reçoit un caractère. Si c'est une lettre minuscule, elle renvoie la même lettre en majuscule. Si c'est une majuscule, elle renvoie la même lettre en minuscule. Si ce n'est pas une lettre, elle renvoie le caractère tel quel. On s'interdit d'utiliser les fonctions Python `upper()` et `lower()`.

Chaînes de caractères

1.5.8 Ajoutez des lettres

1. Écrire une fonction `ajoute_suffixe` qui prend en argument deux chaînes de caractères `chaine` et `suffixe`, et un entier `nb_fois` et qui renvoie la chaîne de caractères obtenue à partir de `chaine` en ajoutant `nb_fois` fois la chaîne `suffixe` à la fin. Par exemple, l'appel `ajoute_suffixe("hello", "you", 3)` doit renvoyer `"helloyouyouyou"`.

2. Écrire une fonction `ajoute_b` qui prend en argument une chaîne de caractères `chaine` et un entier `nb_fois` et qui renvoie la chaîne de caractères obtenue à partir de `chaine` en ajoutant `nb_fois` fois la lettre *b* à la fin. Par exemple, `ajoute_b("bonjour", 5)` doit renvoyer `"bonjourbbbbb"`. Vous pouvez utiliser votre fonction précédente.

3. Écrire un programme principal qui demande à l'utilisateur un mot, un entier et un suffixe, puis qui teste chacune des deux fonctions avec les bons arguments, puis qui affiche le résultat. Par exemple :
```
Donnez un mot: merci
Donnez un nombre: 3
La fonction "ajoute_b" appelée avec ces arguments renvoie : mercibbb
Donnez un suffixe: la
La fonction "ajoute_suffixe" appelée avec ces arguments renvoie : mercilalala
```

1.5.9 Lecture de mots

Écrire les programmes suivants :

1. Demande à l'utilisateur de taper un mot au clavier, jusqu'à ce que ce mot commence par 'a' (en majuscule ou en minuscule), et affiche alors le nombre d'erreurs faites avant la bonne saisie.

2. Demande à l'utilisateur un mot, affiche sa longueur, lui propose de rejouer ; à la fin (refus de rejouer), affiche le mot le plus long qui a été lu.

3. Demande à l'utilisateur de saisir des mots jusqu'à ce qu'il indique 'stop' ; affiche alors combien de mots ont été lus ('stop' ne compte pas), le mot le plus long, et sa longueur.

4. Demande des mots jusqu'à la saisie d'une chaîne vide pour arrêter ; affiche alors lequel des mots lus est le premier dans l'ordre alphabétique.

1.5.10 Palindrome

Écrire un programme / une fonction qui reçoit une chaîne de caractères et vérifie si c'est un palindrome. Afficher le résultat sous la forme "le mot XXX est un palindrome" ou "le mot XXX n'est pas un palindrome".

1.5.11 Premier mot

Écrire une fonction qui reçoit une chaîne et renvoie le premier mot de cette chaîne. Indice : on considère n'importe quel caractère non alphabétique (espace, ponctuation...) comme séparant les mots.

Par exemple pour la chaîne "bonjour à tous", la fonction renvoie "bonjour".

1.5.12 Verlan

Écrire une fonction qui reçoit une chaîne de caractères et la renvoie à l'envers.

1.5.13 Sans voyelles

Écrire une fonction qui reçoit une chaîne de caractères et renvoie cette chaîne dont on a retiré toutes les voyelles.

1.5.14 Ordre alphabétique

Écrire une fonction qui demande à l'utilisateur de rentrer des chaînes de caractères en ordre alphabétique. La fonction s'arrête dès que l'utilisateur tape 'stop' ou tape un mot qui n'est pas dans l'ordre alphabétique. Elle renvoie alors le nombre de mots saisis (sans compter 'stop' ou le mot qui a causé l'arrêt).

Exemple d'exécution (le mot 'marche' cause l'arrêt car il est avant 'voiture' dans l'ordre alphabétique, la fonction renvoie donc 5, le nombre de mots saisis avant 'marche') :

```
Tape des mots dans l'ordre alphabétique ou stop pour arrêter
avion
bateau
train
velo
voiture
marche
5
```

Plus d'exercices sur les chaînes de caractères dans le TD 1.7 (avec boucle for).

1.6 Listes

Basiques

1.6.1 Exercice : code à trous

```
>> Multiple3 = [3, 6, 9, 15, 21]
>> Multiple3[2]
_________________________
>> Multiple3[____]
21
>> Multiple3._____________________
>> Multiple3
[3, 6, 9, 15, 21, 24]

>> Multiple3 = [3, 6, 9, 15, 21, 24, 27]

>> Multiple3.insert( ___, 12)
>> Multiple3.insert( ___, 18)
>> Multiple3
[3, 6, 9, 12, 15, 18, 21, 24, 27]
>> Multiple3._____________([30,33])
>> Multiple3
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33]
>> Multiple3._____________________
>> Multiple3
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36]

>> EhEh = ["tra", "la", "la", "la", "lère"])
>> EhEh._____________
>> EhEh
["tra", "la", "la", "la"]
>> EhEh._____________
>> EhEh
["tra", "la", "la"]
```

1.6.2 Création de liste

Écrire un programme qui demande à l'utilisateur de taper des entiers (jusqu'à obtenir `0`), ajoute les entiers positifs pairs à une liste appelée `liste_pairs`, les entiers positifs impairs à une autre liste appelée `liste_impairs`, et ne fait rien des entiers négatifs. Le programme affiche ensuite les 2 listes. Par exemple si l'utilisateur tape `1,-5,7,8,13,-4,0` alors le programme affiche : `pairs: [8] impairs: [1, 7, 13]` On remarque que les nombres négatifs et le `0` ne sont pas affichés.

1.6.3 Affichage de liste

Écrire une fonction qui reçoit une liste et l'affiche sous les formats suivants :

- Un élément par ligne
- Sur une seule ligne, éléments séparés par des point-virgules
- *Comment ne pas afficher de point-virgule après le dernier élément ?*
- Sur plusieurs lignes, avec 10 éléments par ligne (éventuellement moins sur la dernière) séparés par des espaces
- *Plus difficile* : sur plusieurs colonnes, avec 7 éléments par colonne, par exemple pour l'alphabet :

```
A  H  O  V
B  I  P  W
C  J  Q  X
D  K  R  Y
E  L  S  Z
F  M  T
G  N  U
```

Réécriture de fonctions existantes

On se propose ici de réécrire un certain nombre de fonctions classiques de manipulation de listes, disponibles par défaut en Python mais pas dans d'autres langages comme le C par exemple. On s'interdira donc d'utiliser les fonctions Python prédéfinies. *Remarque* : avant d'avoir vu les boucles `for`, il est possible de faire tous ces exercices en utilisant des boucles `while`.

1.6.4 Test d'appartenance

Écrire une fonction qui reçoit une liste de nombres et un nombre, et qui teste si ce nombre appartient bien à la liste. Cette fonction renvoie donc un booléen. *Remarque : Python permet de faire ce test avec le mot-clé* `in`*, mais on veut ici réécrire cette fonction pour comprendre comment la recherche fonctionne.*

1.6.5 Indice d'apparition

Écrire une fonction qui reçoit une liste de nombres et un nombre, et qui renvoie l'indice de la première position de ce nombre dans cette liste, ou renvoie `-1` si le nombre n'y apparaît pas. *Remarque : Python a une fonction* `index()`*, mais on se propose ici de la réécrire pour comprendre son principe.*

1.6.6 Comptage d'élément

Écrire une fonction qui reçoit une liste de nombres et un nombre et qui compte combien de fois ce nombre apparaît dans la liste. La fonction renvoie ce compteur. *Remarque : Python dispose d'une fonction* `count` *prédéfinie, mais on veut la réécrire.*

1.6.7 Recherche de minimum ou maximum

1. Écrire une fonction qui reçoit une liste de nombres et qui renvoie son élément minimum.
2. Écrire une fonction qui reçoit une liste de nombres et qui renvoie l'indice de son plus grand élément.

Remarque : les fonctions `min` *et* `max` *existent déjà en Python mais on se propose de les réécrire.*

1.6.8 Calcul de somme et moyenne

1. Écrire une fonction qui reçoit une liste de nombres et renvoie la somme de ces nombres.
2. Écrire une fonction qui reçoit une liste de nombres et renvoie la moyenne de ces nombres.

Remarque : la fonction `sum` *existe déjà dans Python. Le module* `statistics` *contient une fonction* `mean` *qui calcule la moyenne. Mais encore une fois on veut les réécrire.*

Manipulation de listes de nombres

1.6.9 Puissances

Écrire une fonction qui reçoit en paramètre un entier n, et qui renvoie la liste des puissances de 2, de la puissance 0 à la puissance n incluse. Par exemple pour $n = 5$, la fonction renvoie la liste $[1, 2, 4, 8, 16, 32]$.

1.6.10 Compter les multiples

Écrire une fonction qui reçoit en paramètre une liste d'entiers, li et un entier x, qui compte combien d'éléments de li sont multiples de x, et qui renvoie la valeur de ce compteur. Par exemple pour $x = 3$ et $li = [3, 2, 5, 9]$ la fonction renvoie 2. Pour la même liste mais $x = 4$, la fonction renvoie 0.

1.6.11 Diviseurs, nombres premiers

1. Écrire une fonction qui reçoit un entier positif et renvoie la liste de ses diviseurs.
2. Écrire une fonction qui reçoit un entier positif et renvoie un booléen qui indique si cet entier est premier. On utilisera la fonction précédente. *On rappelle qu'un entier est premier si ses seuls diviseurs sont 1 et lui-même.*

1.6.12 Jeu de dé et statistiques

1. Écrire une fonction `une_partie` qui reçoit un entier n, lance n fois un dé à 6 faces, stocke les n résultats dans une liste d'entiers, et la renvoie.

2. Écrire une fonction `compteurs_faces` qui reçoit une liste de n tirages, et calcule et renvoie une liste de 6 compteurs indiquant le nombre d'apparitions de chaque face dans cette liste. On s'appliquera à ne parcourir **qu'une seule fois** la liste de tirages, et donc on s'abstiendra d'utiliser `count`.

3. Écrire une fonction `stats_partie` qui reçoit une liste de n tirages, et affiche pour chaque face le pourcentage de tirages qui l'ont obtenue. (Par exemple : `1 - 17.2% ; 2 - 15.5% ;` etc). On utilisera la fonction `compteurs_faces`.

4. Écrire une fonction `face_gagnante` qui reçoit une liste de n tirages, et renvoie la face qui est apparue le plus souvent sur cette partie (la plus grande si égalité). On utilisera la fonction `compteurs_faces`.

5. Écrire un programme principal qui demande à l'utilisateur le nombre de tirages par partie (n), le nombre de parties à jouer (p) ; qui joue p parties de n tirages ; qui affiche pour chaque partie la face gagnante ; qui affiche pour chaque face combien de parties elle a gagné.

1.6.13 [INF104] Construction d'un histogramme

Mots-clés: *distribution de probabilité — représentation graphique — analyse de données*

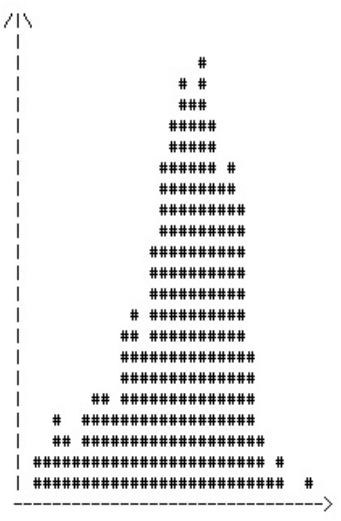
Lorsqu'un processus prend des valeurs aléatoires, on peut en représenter la distribution de probabilité en construisant l'histogramme des valeurs mesurées. Un histogramme est un comptage d'un nombre de valeurs appartenant à divers intervalles, appelés *classes* (ou *bins*). Par exemple un histogramme comportant 14 classes et s'étendant de 80 à 150 km/h (chaque classe est donc de largeur 5 km/h) permet de représenter la distribution des vitesses des véhicules en un point d'une autoroute.

1. Écrire une fonction `genere_valeurs` qui reçoit un entier n, une moyenne μ et une largeur σ, et renvoie une liste de n nombres aléatoires tirés selon une distribution gaussienne de paramètres $(\mu; \sigma)$.

2. Écrire une fonction `construit_histo` qui reçoit un nombre de classes et une liste de valeurs, détermine le minimum et le maximum de cette dernière, et renvoie une liste contenant ces deux valeurs suivies du nombre de classes.

3. Écrire une fonction `remplit_histo` qui reçoit une liste de valeurs et un nombre de classes, et renvoie une liste avec le nombre d'observations dans chaque classe de l'histogramme (c'est-à-dire que la valeur en position 0 dans la liste renvoyée correspond au nombre de valeurs dans la classe numéro 0, et ainsi de suite).

4. Écrire un programme principal qui demande μ et σ à l'utilisateur, qui génère des valeurs correspondantes, et qui trace l'histogramme sous forme de colonnes de hauteur égale au nombre d'observations dans chaque classe, comme dans la figure ci-contre.



1.6.14 [INF104] Correction d'efficacité d'un détecteur

Mots-clés: *détecteur silicium — trajectographe — Geant4*

En physique des particules, la mesure de taux de production de telle ou telle particule est fondamentale pour tester le modèle standard. Elle nécessite cependant de corriger le taux mesuré de l'efficacité du détecteur, car, ce dernier n'étant jamais parfait, il ne réagit pas au passage de toutes les particules.

L'efficacité d'un détecteur est définie comme la proportion de particules qui le traversent qui donnent naissance à la reconstruction d'une "entité particule".

On considère par exemple un détecteur constitué d'un nombre $N = 8$ de plans de lecture, ayant chacun la même probabilité $p = 0{,}98$ de générer un signal lorsqu'une particule le traverse. L'algorithme est capable de reconstruire une "entité particule" lorsqu'au moins $n = 5$ des plans ont généré un signal.

1. Écrire une fonction `generation_particule` qui reçoit N et p, tire N nombres aléatoires et renvoie une liste de N `0` ou `1`, selon que la particule laisse ou non un signal dans chaque couche considérée.

2. Écrire une fonction `particule_reconstruite` qui reçoit N, n et une liste de `0` ou `1`, teste si la longueur de la liste est correcte, et renvoie `0` si une "entité particule" peut être reconstruite, `1` sinon.

3. Écrire une fonction `estim_eff` qui reçoit les valeurs N, p et n, simule aléatoirement le passage de 1000 particules et renvoie l'efficacité ε.

4. En réalité, pour reconstruire une "entité particule" il faut aussi que cette dernière ait laissé un signal dans au moins l'une des 2 premières couches. Modifier ce qu'il faut pour prendre cela en compte.

5. Écrire une fonction qui calcule une résolution (sur une grandeur quelconque), que l'on supposera égale à $1/\sqrt{n}$.

6. Écrire une fonction qui calcule à partir de 1000 particules la résolution moyenne σ du détecteur.

7. Écrire un programme qui appelle la fonction pour des valeurs croissantes de p et N et représente l'évolution de ε et de σ en fonction de p.

8. Modifier la fonction `estim_eff` de façon à pouvoir tenir compte d'une proportion de canaux de lecture défectueux propre à chaque plan de lecture.

Manipulation de listes de caractères

Ces exercices utilisent les fonctions sur les caractères et chaînes de caractères.

1.6.15 Codage par décalage

1. Écrire une fonction `decal_list` qui reçoit une liste de caractères (représentant un mot) et un entier `n`, et qui renvoie la liste de caractères correspondant à ce mot codé par décalage de chaque lettre de `n` positions. On pourra utiliser la fonction de décalage d'un caractère codée précédemment (exo 1.5.6) ou la réécrire.

2. Écrire une fonction `list_to_string` qui reçoit une liste de caractères et renvoie la chaîne de caractères obtenue en les concaténant dans l'ordre. Par exemple cette fonction transforme la liste `['a','z','e','r','t','y']` en la chaîne de caractères `"azerty"`. **Remarque** : la fonction prédéfinie `join` permet aussi de faire cette concaténation (cf cours **??**), mais on veut la réécrire.

3. Écrire ensuite un programme principal qui demande à l'utilisateur un mot et un entier, code ce mot par décalage, et affiche le résultat. Par exemple le mot `"Bonjour"` décalé de 3 positions doit devenir `"Erqmrxu"`. Il faut donc récupérer la liste des caractères de ce mot (fonction `list`).

4. Modifier ce programme principal pour demander en boucle un mot à l'utilisateur (s'arrête en lisant le mot `"stop"`), puis un entier, et afficher le codage de ce mot par décalage de cet entier.

1.6.16 Comptage alphabétique (force brute vs subtile)

1. Écrire une fonction `compte_carac` qui reçoit une liste de caractères et une lettre, compte combien de fois cette lettre apparaît dans la liste (que ce soit en minuscule ou en majuscule), et renvoie le résultat.

2. Écrire une fonction `compte_alphab` qui reçoit une liste de caractères, compte toutes les lettres de l'alphabet avec la fonction précédente, et renvoie la liste des 26 compteurs.

3. Question : combien de fois a-t-on parcouru la liste de caractères ?

4. Écrire une nouvelle fonction `compte_alphab2` qui compte toutes les lettres de l'alphabet en un seul parcours de la liste de caractères, et renvoie la liste des 26 compteurs.

5. Bonus avec des dictionnaires : écrire une fonction `compte_alphab_dico` pour compter tous les caractères qui apparaissent dans un texte ; il peut y avoir des caractères non alphabétiques, et surtout on ne sait pas à l'avance quels caractères apparaissent ou pas. Cette fonction renvoie un dictionnaire dont les clés sont les caractères du texte, et dont les valeurs associées sont les compteurs correspondants.

1.6.17 Liste de mots

1. Écrire une fonction *taille(liste)* qui prend en argument une liste des chaînes de caractères et renvoie la liste des tailles de chaque élément de la liste.

2. Écrire une fonction *lire(n)* qui reçoit en paramètre un entier n, lit au clavier n chaînes de caractères, les stocke dans une liste, et renvoie cette liste. Un exemple d'exécution avec n=5 :

```
Tapez un mot : train
Tapez un mot : cheval
Tapez un mot : voiture
Tapez un mot : avion
Tapez un mot : accordeon
['train','cheval','voiture','avion','accordeon']
```

3. Écrire une fonction *affiche(liste)* qui reçoit en paramètre une liste de chaînes, calcule la liste de leurs tailles, affiche chaque chaîne et sa taille, puis la moyenne des tailles. Par exemple avec la liste précédente :

```
Taille du mot train : 5
Taille du mot cheval: 6
Taille du mot voiture: 7
Taille du mot avion: 5
Taille du mot accordeon: 9
Taille moyenne: 6.4
```

4. Compléter pour afficher aussi les mots plus longs que la taille moyenne, sur une seule ligne.

```
Mots plus longs que la moyenne: voiture ; accordeon ;
```

5. Écrire une fonction `nbocc(mot,carac)` qui compte et renvoie le nombre d'occurrences (le nombre d'apparitions) d'un caractère donné dans une chaîne donnée.

6. Utiliser la fonction `nbocc` pour écrire une fonction *compteCarac(liste,car)* qui reçoit une liste de chaînes et un caractère, affiche les mots contenant ce caractère et le nombre total d'occurrences de ce caractère dans tous les mots. Si ce caractère n'est présent dans aucun mot, la fonction affiche un message d'erreur à la place. Cette fonction ne renvoie rien. Deux exemples d'exécution (avec la liste précédente de 5 mots et la lettre 'o' puis 'w') :

```
Mots contenants le caractère o :
voiture
avion
accordéon
Le caractère o apparaît 4 fois.

Erreur la lettre w n'est présente dans aucun des mots
```

7. Utiliser la fonction `nbocc` pour trouver le mot d'une liste (reçue en paramètre) où un caractère donné (en paramètre) apparaît le plus de fois. En cas d'égalité, favoriser le mot le plus court contenant autant de fois ce caractère. Renvoyer ce mot. **Version avancée :** renvoyer ce mot **et** le nombre d'occurrences du caractère dedans.

8. Écrire un programme principal qui utilise les fonctions ci-dessus et : lit au clavier le nombre de mots de la liste, puis les mots ; les affiche avec leur taille ; puis lit un caractère, le compte dans les mots, et affiche le mot qui contient le plus de fois ce caractère.

9. Comment modifier ce programme pour proposer à l'utilisateur de rejouer la dernière étape (même liste, nouveau caractère à compter) jusqu'à ce qu'il refuse ?

1.7 Itération inconditionnelle `for`

Quelques exercices basiques pour commencer

1.7.1 Exercices de lecture

Pour les instructions suivantes, indiquer ce qui sera affiché.

```
for i in range(4) :
    print(i)

for j in range(2,5):
    print(j)

for k in range(3,12,3):
    print(k)

for l in range(12, 3):
    print(l)

for m in range(12,3,-2):
    print(m)
```

Expliquer ce qu'affiche le programme suivant.

```
# la liste en arg. doit ne contenir que des nb
def somme_des_positifs(liste) :
    s=0
    for e in liste:
        if e>0:
            s=s+e
    return s

# prog. principal
ma_liste = [2,-4,6,0,-5,1]
for e in ma_liste:
    print(e+1)
t=somme_des_positifs(ma_liste)
print(t)
```

1.7.2 Les bases des boucles for

1. Écrire une boucle for qui affiche les entiers de 1 à 10 (un par ligne)
2. Écrire une boucle for qui affiche les entiers de 1 à 10 en ordre inverse
3. Idem mais en les affichant sur une seule ligne, séparés par une virgule, ligne terminée par un point.
4. Écrire une fonction qui reçoit 3 entiers (deux bornes et un pas) et affiche les entiers entre ces deux bornes incluses, séparés par ce pas. (Par ex. pour les bornes 1 et 20, et un pas de 3, il doit afficher : 1, 4, 7, 10, 13, 16, 19.)
5. Écrire un programme principal qui demande à l'utilisateur des bornes `binf` et `bsup` et un pas `step`, et appelle la fonction précédente pour afficher la liste des entiers entre ces bornes séparés par ce pas.
6. Écrire une boucle for qui affiche les multiples de 3 et les multiples de 5 compris entre 1 et 50.
7. Idem mais en n'affichant pas les multiples de 15. Par exemple on veut afficher : 3,5,6,9,10,12,18,20,21,etc

1.7.3 Sommes d'entiers

Écrire des fonctions qui reçoivent un entier n et qui :

1. f1 renvoie la somme des n premiers entiers
2. f2 renvoie la somme des n premiers entiers impairs
3. f3 renvoie la somme des n premiers carrés

Écrire un programme principal qui lit un entier au clavier, appelle ces fonctions l'une après l'autre, et affiche leurs résultats (avec un texte clair à chaque fois).

1.7.4 Suite croissante

- Écrire une fonction qui reçoit une liste d'entiers et qui renvoie un booléen indiquant si les éléments de cette liste sont dans l'ordre croissant (non strict), c'est-à-dire si pour tout indice i, l'élément en position i est inférieur ou égal à l'élément en position i+1.
- On suppose qu'on a une fonction similaire qui détermine si une liste reçue en paramètre est en ordre décroissant. Écrire une fonction qui détermine si une liste reçue en paramètre est dans le désordre (ni en ordre croissant, ni en ordre décroissant).

1.7.5 Comptons les moutons

1. Écrire un programme qui demande à l'utilisateur un nombre de moutons, et qui "compte les moutons" :

```
Combien de moutons ? 3
1 moutons
2 moutons
3 moutons
```

2. Que se passe-t-il si le nombre de moutons rentré par l'utilisateur est 1 ? Est nul ? Est négatif ?
3. Modifier le programme précédent pour qu'il affiche un seul mouton au singulier ("1 mouton" plutôt que "1 moutons"). Attention : on évitera les comparaisons inutiles.

1.7.6 Règle graduée

Dans cet exercice on veut utiliser des boucles for, on s'interdit d'utiliser la multiplication de chaînes (par exemple n*'-' donne directement une chaîne de n caractères '-').

1. Écrire un programme qui demande à l'utilisateur un entier n, puis affiche avec des '-' une règle de longueur n. Par exemple :

```
Longueur ? 13
-------------
```

2. Modifier ce programme pour qu'il affiche une règle graduée, avec un '|' au début, et un '|' à la place de chaque x-ième '-', où l'intervalle x est aussi demandé à l'utilisateur. Par exemple :

```
Longueur ? 27
Intervalle ? 10
|---------|---------|-------
```

Les choses se compliquent un peu...

1.7.7 Accumulateurs

1. Écrire une fonction qui utilise un `for` pour calculer et renvoyer la factorielle d'un entier n reçu en paramètre.
2. Écrire une fonction qui reçoit en paramètre deux entiers n et a, utilise une boucle for pour calculer n à la puissance a (sans utiliser l'opérateur de puissance **), et renvoie le résultat.
3. Que se passe-t-il si l'utilisateur saisit un a négatif ? Le résultat est-il correct ?
4. Modifier cette fonction pour qu'elle soit capable de calculer les puissances négatives (rappel : $a^{-1} = 1/a$)

1.7.8 Pile ou face, et statistiques

1. Écrire une fonction `nb_pile(T)` qui reçoit un entier T, simule une partie de T tirages Pile ou Face, et renvoie le nombre de Pile obtenus.
2. Écrire une fonction `liste_nbpile(N,T)` qui reçoit un entier N et un entier T, qui simule N parties de T tirages Pile ou Face, qui construit la liste de N entiers contenant le nombre de Pile obtenus pour chaque partie, et qui renvoie cette liste.

3. A partir d'une liste d'éléments correspondant au nombre de Pile obtenus dans plusieurs parties de Pile ou Face, et d'un entier T (le nombre de tirages par partie), on veut construire et afficher un histogramme du nombre de parties ayant obtenu chaque nombre de Pile possible (entre 0 et T inclus). On va procéder en plusieurs étapes :

   (a) Écrire une fonction `compte_parties_xpile(liste_pile,x)` qui reçoit la liste de N nombres de tirages Pile par partie (pour les N parties), et qui compte le nombre d'occurrences d'une valeur x. Cela représente le nombre de parties qui se sont conclues par x tirages Pile.

   (b) Utiliser cette fonction pour écrire une fonction `liste_cpt_parties` qui construit une liste de compteurs correspondant au nombre de parties qui se sont conclues par chaque nombre de tirages Pile possible (T+1 valeurs, entre 0 si on a fait Face à chaque fois, et T si on a fait Pile à chaque fois). La fonction renvoie cette nouvelle liste.

   (c) Combien de fois parcourt-on la liste des résultats des N parties ?

   (d) Réécrire cette fonction pour compter le nombre de parties conclues par chaque nombre de tirages Pile (liste à T+1 éléments) en un seul parcours de la liste de N parties.

   (e) Affichage histogramme : écrire une fonction qui reçoit une liste de T+1 entiers et l'affiche sous forme d'un histogramme horizontal à T+1 lignes, sur chacune desquelles apparaît le numéro i de la ligne (de 0 à T inclus) et un nombre d'étoiles proportionnel à la valeur associée (qui correspond au nombre de parties ayant obtenu exactement i Pile).

4. Écrire un programme principal qui demande à l'utilisateur un nombre de parties N et un nombre de tirages par partie T, qui simule les tirages et affiche l'histogramme résultant, en appelant les fonctions précédentes.

1.7.9 [INF104] Déchets nucléaires

Mots-clés: *filiation radioactive — équilibre séculaire — datation*

Pour une chaîne de désintégrations nucléaires $A_1 \to A_2 \to ... \to A_n$, les équations de Bateman permettent de connaître le nombre de noyaux de chaque élément A_k à l'instant t, connaissant leurs demi-vies T_k. Lorsque seuls des noyaux de A_1 sont présents en $t = 0$,

$$N_k(t) \;=\; N_1(0) \times \left(\prod_{i=1}^{k-1} \lambda_i\right) \times \sum_{i=1}^{k} \frac{e^{-\lambda_i t}}{\prod\limits_{j=1, j\neq i}^{k} (\lambda_j - \lambda_i)} \qquad \text{avec :} \quad \lambda_i = \frac{\ln 2}{T_i}$$

Une version simplifiée de la chaîne de désintégrations du Plutonium 239, un déchet abondamment produit par les centrales nucléaires, est la suivante :

- Il se désintègre en Uranium-235 avec une période de 24000 ans,
- Qui se désintègre en Protactinium-231 avec une période de 704 millions d'années,
- Qui se désintègre en Actinium-227 avec une période de 33000 ans,
- Qui se désintègre en Radium-223 avec une période de 21,8 ans,
- Qui se désintègre en Plomb-207 (stable) avec une période de 11,4 jours.

Écrire un programme qui demande un temps et le numéro d'un élément dans la chaîne, et calcule le nombre de noyaux de cet élément au bout du temps demandé, en prenant $N_1(0) = 10^{21}$.

Manipulation de chaînes de caractères

On rappelle que les chaînes de caractères sont un type itérable, c'est-à-dire dont on peut parcourir les éléments (les caractères individuels) à l'aide d'une boucle. On l'a déjà pratiqué avec des boucles `while` *en parcourant les indices des éléments, on va maintenant le pratiquer avec les boucles* `for` *en parcourant soit les indices, soit directement les éléments. On pourra aussi refaire les exercices du TD 1.5 en utilisant* `for` *plutôt que* `while`*.*

1.7.10 Modification de chaînes

1. Écrire une fonction qui supprime tous les espaces d'une chaîne de caractères reçue, et renvoie le résultat.
2. Écrire une fonction qui reçoit une chaîne de caractères, et renvoie son inverse. Par exemple :

```
>>> inverse_chaîne("inf101") == "101fni"
True
```

3. Écrire une fonction qui reçoit une chaîne de caractères, mélange ses lettres, et renvoie le résultat (sous forme d'une chaîne, et pas d'une liste).

1.7.11 Palindromes

1. Écrire une fonction qui reçoit un mot, qui détermine si ce mot est un palindrome, et qui renvoie le résultat (un booléen). Un palindrome est un mot qui se lit de la même manière de la gauche vers la droite ou de la droite vers la gauche, comme "kayak" ou "elle".

2. Écrire une fonction qui reçoit une phrase, détermine si la phrase est un vrai palindrome (dans un vrai palindrome, les espaces sont répartis de manière symétrique dans la phrase, par exemple : "rats live on no evil star") ; un pseudo-palindrome (dans un pseudo palindrome, il y a une symétrie entre les lettres mais pas entre les espaces, par exemple "eve reve") ; ou ni l'un ni l'autre. Cette fonction renvoie le résultat sous la forme d'un entier (par exemple vrai palindrome = 2, pseudo-palindrome = 1, pas un palindrome = 0).

3. Écrire un programme principal qui boucle jusqu'à ce que l'utilisateur saisisse "stop". Pour chaque chaîne de caractère saisie :

   - Si c'est le mot "stop", le programme s'arrête en affichant "fin"
   - S'il s'agit d'un mot (ne contenant pas d'espaces), il appelle la première fonction pour déterminer si c'est un palindrome, et affiche le résultat
   - S'il s'agit d'une phrase (contient des espaces), il appelle la deuxième fonction pour déterminer son type, et affiche un message correspondant.
   - Demande ensuite une nouvelle chaîne de caractères et recommence.

1.7.12 Zip

1. Écrire une fonction qui reçoit 2 chaînes et renvoie leur "fermeture éclair", c'est-à-dire la liste de toutes les chaînes obtenues en combinant un caractère de la première chaîne suivi d'un caractère de la deuxième chaîne.

```
>>> zip_chaine("abc", "d") == ['ad', 'bd', 'cd']
True
>>> zip_chaine("a", "bcd") == ['ab', 'ac', 'ad']
True
>>> zip_chaine("ab", "cd") == ['ac', 'ad', 'bc', 'bd']
True
```

2. Définir une fonction qui réalise la fermeture éclair de 2 listes lst1 et lst2 reçues en paramètres, c'est-à-dire la liste de toutes les paires composées d'un élément de lst1 et d'un élément de lst2. Par exemple :

```
>>> zip_liste([1], [2,3,4]) == [(1, 2), (1, 3), (1, 4)]
True
>>> zip_liste([1,2,3], [4]) == [(1, 4), (2, 4), (3, 4)]
True
>>> zip_liste([1,2], [3,4]) == [(1, 3), (1, 4), (2, 3), (2, 4)]
True
```

1.7.13 Codage et décodage de texte

1. Soit la fonction `NextElem` qui prend en argument un élément `elm` et une liste d'au moins 2 éléments `liste`. La fonction renvoie l'élément suivant `elm` dans la liste `liste`, ou le premier élément de la liste si `elm` est son dernier élément. La fonction renvoie None si `elm` n'est pas dans la liste (cf exemple ci-dessous).

```
> print(NextElem('a',['a','b','c']))
b
> print(NextElem('c',['a','b','c'])
a
> print(NextElem('d',['a','b','c'])
None
```

2. Écrire une fonction `genereCode` qui génère et renvoie une liste mélangée de l'alphabet.

3. On souhaite écrire un programme principal qui lit un texte (en minuscules sans accents), puis le crypte en utilisant le principe de "codage par décalage de lettres". Pour cela, on utilise les fonctions précédentes pour générer une liste mélangée de caractères, et décaler les caractères du texte selon l'ordre de cette liste. Les caractères non alphabétiques ne sont pas modifiés. On affichera le résultat obtenu.

Manipulation de listes

Les listes sont aussi un type itérable, c'est-à-dire dont on peut parcourir les éléments (qui peuvent être de n'importe quel type) à l'aide d'une boucle. On l'a déjà pratiqué avec des boucles `while` *en parcourant les indices des éléments, on va maintenant le pratiquer avec les boucles* `for` *en parcourant soit les indices, soit directement les éléments. On pourra aussi refaire les exercices du TD 1.6 en utilisant* `for` *plutôt que* `while`*.*

1.7.14 Copie et inversion de listes

1. Écrire une fonction qui copie la liste reçue en paramètre et renvoie la nouvelle liste. On demande donc en fait de ré-implémenter la fonction `list` qui existe déjà en Python.
2. Écrire une fonction qui crée une copie en ordre inverse de la liste reçue en paramètre, et renvoie cette copie inversée. On ne veut pas utiliser la fonction `reverse()` qui existe déjà en Python.

1.7.15 Liste de mots

1. Écrire une fonction qui reçoit une liste de mots lm, un caractère c, et qui renvoie une liste des mots de lm qui se terminent par c.
2. Écrire une fonction similaire mais qui renvoie la liste des mots de lm qui commencent par c
3. **Bonus (fonctions avancées) :** Écrire une fonction similaire avec un paramètre optionnel booléen permettant de choisir si l'on veut que le mot commence ou termine par la lettre c.
4. Écrire une fonction avec les mêmes paramètres qui renvoie la liste des mots de lm qui contiennent la lettre c.
5. Écrire une fonction similaire avec un paramètre supplémentaire n (bonus : optionnel, valeur par défaut $= 1$) qui renvoie la liste des mots de lm contenant au moins n fois la lettre c.

1.8 Listes avancées : `for`, tri, listes à 2 dimensions, etc

Ces exercices sont réalisables avec une boucle `while`*, mais seront plus faciles avec la boucle* `for` *(voir cours* **??***).*

1.8.1 Mélange de liste

1. Écrire une fonction qui reçoit une liste, et en crée une copie mélangée qu'elle renvoie. On ne veut pas modifier la liste reçue. On ne veut pas utiliser la fonction existante `shuffle`, mais la réécrire. On pourra utiliser le module `random`.
2. Écrire une fonction qui reçoit deux listes de même taille, contenant les mêmes éléments dans le désordre, et qui compte le nombre d'éléments qui sont à la même position dans les 2 listes. Cette fonction renvoie ce compteur.
3. Écrire une fonction qui demande à l'utilisateur de saisir un par un des entiers positifs, avec -1 pour finir la saisie (-1 n'est pas ajouté à la liste). La fonction renvoie alors la liste ainsi créée.
4. Écrire un programme principal qui lit une liste saisie par l'utilisateur (avec la fonction ci-dessus), puis qui génère un mélange de cette liste en boucle jusqu'à ce qu'aucun élément ne soit à la même place dans la copie mélangée. Le programme affiche alors le nombre de mélanges qui ont été nécessaires pour atteindre ce résultat.

1.8.2 Insertion par recherche dichotomique

Étant donnée une liste d'entiers triée dans l'ordre croissant, on souhaite insérer un nouvel élément à la bonne position pour garder la liste triée.

1. Écrire une fonction *insert_ intuitif(liste_ triee, e)* qui prend en argument une liste d'entiers supposée déjà triée et un entier, et qui insère le nouvel élément par la méthode qui vous semble la plus intuitive. Attention la fonction ne renvoie rien mais modifie la liste qui lui est passée en argument.
2. Si on note n la longueur de la liste, quel est le nombre maximum d'itérations utilisées dans *insert_ intuitif* pour réaliser l'insertion ?

On voudrait utiliser une méthode dichotomique pour insérer un entier e dans une liste triée. L'idée est de comparer e avec un élément de la liste qu'on appelle le pivot, idéalement situé au milieu. Si e est plus grand que le pivot alors il doit être inséré après, sinon il doit être inséré avant. On recommence ensuite la même opération avec la moitié de liste concernée.

Illustrons cela par un exemple. Dans la liste $[2, 12, 17, 25, 33, 35, 44, 54, 77, 91]$ on souhaite insérer 49. Pour chaque itération on délimite par d et f la portion de la liste où l'on sait que 49 doit être inséré. p représente le pivot.

2	12	17	25	33	35	44	54	77	91
d				p					f

2	12	17	25	33	35	44	54	77	91
				d			p		f

2	12	17	25	33	35	44	54	77	91
				d	p		f		

2	12	17	25	33	35	44	54	77	91
					d	p	f		

2	12	17	25	33	35	44	54	77	91
						d	f		

On voit que 49 doit être inséré entre 44 et 54.

3. Écrire une fonction *insert_ dicho(liste_ triee, e)* qui implémente cette méthode.
4. Supposons que la longueur de la liste est une puissance de 2, notée $n = 2^k$. Quel est le nombre d'itérations utilisées dans *insert_ dicho* pour réaliser l'insertion ? Quelle est la méthode la plus rapide ?

1.8.3 [INF104] Température des étoiles

Mots-clés: *corps noir — loi Stefan-Boltzmann — diagramme Hertzsprung-Russel*

Les étoiles peuvent être considérées comme des corps noirs, c'est-à-dire que la répartition de l'énergie lumineuse émise selon les longueurs d'onde ne dépend que de leur température, et suit la loi de Planck :

$$f_T(\lambda) \propto \frac{1}{\lambda^5 \left(e^{\frac{hc}{\lambda k_B T}} - 1\right)} \qquad \text{où } h,\ c \text{ et } k_B \text{ sont des constantes fondamentales.}$$

Il est donc possible de connaître la température d'une étoile à distance en mesurant le maximum du spectre lumineux émis. En dérivant la fonction, on montre que le pic se trouve à la longueur d'onde vérifiant cette équation :

$$e^{\frac{hc}{\lambda k_B T}} \left(\frac{hc}{k_B T} - 5\lambda \right) - 5\lambda^4 = 0$$

Malheureusement, cette équation n'admet pas de solution algébrique. Écrire un programme qui demande une température et détermine une approximation de la valeur de la longueur d'onde du pic.

1.8.4 Jouons avec des listes triées

1. Soit la fonction `InsererDans` ayant deux arguments `el` et *liste_triee*, une liste de nombres classée en ordre croissant. La fonction ne doit pas la modifier ! La fonction `InsererDans(el,` *liste_triee*`)` retourne une liste dans laquelle l'élément `el` est placé à la bonne position dans *liste_triee*.

   ```
   > print(InsererDans(1,[]))
   [1]
   > print(InsererDans(4,[1,3,5]))
   [1,3,4,5]
   ```

   Indice : il faut d'abord cloner la liste *liste_triee* dans une variable locale, la parcourir pour identifier la position où l'élément doit être inséré, et utiliser la fonction insert. Attention au cas où la liste triée est vide.

2. Soit la fonction `FusionListesT` qui reçoit pour arguments deux listes ordonnées, et qui renvoie une liste ordonnées correspondant à la fusion des deux listes, sans perdre aucun élément (il peut donc y avoir des doublons). La taille de la liste retournée doit donc être égale à la somme des tailles des deux listes fournies en argument. On utilisera la fonction `InsererDans` définie ci-dessus.

   ```
   > print(FusionListesT([],[]))
   []
   > print(FusionListesT([1,3,4],[2,2,7,9])
   [1,2,2,3,4,7,9]
   ```

3. Soit la fonction `SupprimDoublons` qui reçoit en argument une liste ordonnée, et retourne une nouvelle liste ordonnée dans laquelle les doublons ont été éliminés.

   ```
   > print(SupprimDoublons[1,2,3])
   [1,2,3]
    > print(SupprimDoublons[1,1,1])
   [1]
   ```

1.8.5 Tri par sélection

Écrire une fonction qui reçoit une liste en paramètre, et la modifie pour la trier. Cette fonction ne renvoie rien, par contre elle a des effets de bord (elle modifie la liste reçue en paramètre).

On utilisera l'algorithme du tri par sélection qui consiste à répéter en boucle les étapes suivantes :

- Chercher le minimum de la partie droite de la liste (partie non triée).
- Échanger le minimum trouvé avec l'élément à l'indice i (indice de début de la partie non triée).
- Décaler l'indice i, donc la limite entre partie triée et partie non triée, vers la droite.

L'algorithme commence avec une limite au début de la liste (partie triée de la liste : vide, partie non triée = liste entière). A chaque étape, la longueur de la partie triée augmente et la longueur de la partie non triée diminue. L'algorithme s'arrête quand la limite entre parties triée et non triée atteint la fin de la liste, et donc que la liste est entièrement triée. Par exemple :

```
>>> tri_selection([1,2,3]) == [1,2,3]
True
>>> tri_selection([3,2,1]) == [1,2,3]
True
>>> tri_selection([3,1,2]) == [2,1,3]
False
```

1.8.6 Cartes bancaires et formule de Luhn

Pour protéger les numéros de cartes bancaires contre les erreurs aléatoires (faute de frappe par exemple), ceux-ci sont générés de façon à respecter la formule de Luhn. Il s'agit d'une somme de contrôle, c'est-à-dire une série d'opérations arithmétiques, dont le résultat doit être un multiple de 10 pour que le numéro soit valide. On modélise un numéro par la liste de ses chiffres de gauche à droite. Voici la description de l'algorithme de Luhn faite par wikipedia (`https://fr.wikipedia.org/wiki/Formule_de_Luhn`), en trois étapes :

- L'algorithme multiplie par deux un chiffre sur deux, en commençant par l'avant dernier et en se déplaçant de droite à gauche. Si un chiffre qui est multiplié par deux est plus grand que neuf (comme c'est le cas par exemple pour 8 qui devient 16), alors il faut le ramener à un chiffre entre 1 et 9. Pour cela, il y a 2 manières de faire (pour un résultat identique) :
  - Soit les chiffres composant le doublement sont additionnés (pour le chiffre 8 : on obtient d'abord 16 en le multipliant par 2 puis 7 en sommant les chiffres composant le résultat : 1+6).
  - Soit on lui soustrait 9 (pour le chiffre 8 : on obtient 16 en le multipliant par 2 puis 7 en soustrayant 9 au résultat).
- La somme de tous les chiffres obtenus est effectuée.
- Le résultat est divisé par 10. Si le reste de la division est égal à zéro, alors le nombre original est valide.

1. Commencer par écrire une fonction `changeUnChiffre` qui reçoit un chiffre et le modifie selon l'étape 1 de l'algorithme.
2. Puis écrire une fonction `changeChiffres` qui reçoit une liste de chiffres, les parcourt et les modifie selon l'étape 1 de l'algorithme (on rappelle qu'on ne change qu'un chiffre sur 2).
3. Enfin écrire une fonction `resteDivSomme` qui reçoit une liste de chiffres (résultats de l'étape 1), calcule leur somme (étape 2), la divise par 10 et renvoie le reste de cette division (étape 3).
4. On peut maintenant écrire une fonction *verifie_ Luhn(numero)* qui prend en argument un numéro représenté par la liste de ses chiffres, et qui renvoie un booléen indiquant s'il vérifie la formule de Luhn. On utilisera les fonctions précédentes.
5. Pour générer un numéro valide, il suffit d'ajouter à la fin (à droite) d'un numéro quelconque un chiffre de contrôle. Pour déterminer ce chiffre, on applique la formule de Luhn sur le numéro auquel on a ajouté un chiffre de contrôle égal à zéro. Si le résultat S est un multiple de 10 il n'y a rien à changer. Sinon, on remplace le chiffre de contrôle par $10 - (S \ \% \ 10)$. Écrire une fonction *ajoute_ chiffre_ controle(numero)* qui prends en argument une liste de chiffres et qui lui rajoute un chiffre de contrôle pour la rendre valide.
6. Écrire une fonction *genere_ numero(n)* qui prend un entier positif en argument et qui renvoie un numéro de n décimales valide généré aléatoirement. On utilisera la fonction *randint* du module *random*, et la fonction définie ci-dessus.

Listes de listes, ou listes "à 2 dimensions"

1.8.7 Jeu de démineur (listes de listes)

Dans cet exercice on veut coder un jeu de démineur (avec un affichage uniquement textuel). Même si cet exercice est d'abord abordé en TD, il vous est conseillé de le tester ensuite sur machine. La grille de démineur (à N lignes et M colonnes) sera représentée par une liste de listes de valeurs. Chaque liste de M éléments contient toutes les valeurs d'une ligne donnée de la grille. La liste de N éléments contient donc les N lignes de la grille. Il s'agit ici de la grille solution, c'est-à-dire la grille contenant la position des mines. Il est par ailleurs nécessaire de stocker aussi une autre grille, indiquant quelles cases ont déjà été dévoilées par le joueur. On choisit pour cela d'utiliser une liste de listes de booléens. La valeur True dans une case signifie que le joueur a déjà dévoilé cette case, alors que la valeur False signifie que cette case n'a pas encore été dévoilée.

1. Écrire une fonction `creerGrille` qui : reçoit en paramètres les entiers N et M, et un paramètre optionnel v (par défaut 0) représentant la valeur d'initialisation de toutes les cellules ; crée une grille de démineur à N lignes et M colonnes (donc une liste de N listes de M entiers), ne contenant que des valeurs v (0 par défaut, ou autre si spécifiée) ; et renvoie cette liste. *Par exemple pour N=5 et M=7 on obtient la liste suivante :*

```
[[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0]]
```

2. Écrire une fonction `placerMines` qui : reçoit en paramètre une grille de démineur et un entier X ; modifie cette liste pour y placer X mines (valeur 1) à des positions au hasard ; ne renvoie rien. *Par exemple si on demande de placer 7 mines dans la liste précédente, on obtient :*

```
[[0, 0, 1, 0, 0, 0, 0], [0, 0, 1, 1, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1],
 [1, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 0]]
```

3. Écrire une fonction `afficheSolution` qui : reçoit en paramètre une liste `positionsMines` (grille d'entiers indiquant les positions des mines), et affiche cette grille sous la forme d'un rectangle de caractères représentant la solution du démineur, c'est-à-dire dévoilant les positions des mines. On choisit d'afficher avec un '-' (tiret) les cases non minées (valeur 0), et avec une '*' (étoile) les cases minées (valeur 1). *Par exemple la liste précédente sera affichée sous la forme suivante (à gauche avant placement des mines, à droite après) :*

```
-------             --*----
-------             --***--
-------             ------*
-------             *----*-
-------             -------
```

4. Écrire une fonction `testMine` qui : reçoit la grille des positions des mines, et 2 coordonnées i (numéro de ligne entre 0 et N-1) et j (numéro de colonne entre 0 et M-1) supposées correctes (on n'a pas besoin de les tester) ; vérifie s'il y a une mine sur la case indiquée par ces coordonnées ; et renvoie un booléen indiquant le résultat. *Par exemple :*

```
testMine(0,2) renvoie True
testMine(1,1) renvoie False
```

5. Écrire une fonction `compteMinesVoisines` qui : reçoit la grille des positions des mines, et 2 coordonnées i et j supposées correctes ; compte le nombre de mines sur les cases voisines (attention aux effets de bord, certaines cases ont moins de voisines que d'autres ! 3 voisines dans les coins, 5 voisines sur les bords, 8 voisines au centre) ; renvoie ce compteur. On pourra utiliser une fonction auxilliaire qui calcule et renvoie la liste des cellules voisines d'une cellule donnée.

6. Écrire une fonction `afficheJeu` qui : reçoit en paramètre une liste `positionsMines` (grille d'entiers indiquant les positions des mines), et une liste `casesDevoilees` (grille de booléens indiquant les cases dévoilées) ; affiche la grille de jeu sous la forme d'un rectangle de caractères représentant la grille telle que le joueur la voit pendant la partie (il ne voit pas les positions des mines). On choisit d'afficher avec un '?' (point d'interrogation) les cases non encore découvertes ; avec un '*' une case découverte minée (se produit quand le joueur perd) ; sur les cases découvertes non minées, on affichera un entier indiquant le nombre de mines sur les cellules voisines (compté avec la fonction précédente). *Par exemple affichage avec seulement 2 cases découvertes dans les coins :*

```
0??????
???????
???????
???????
??????1
```

7. Écrire une fonction `getCoords` qui : reçoit en paramètre la grille indiquant les cases déjà dévoilées, et les dimensions N et M ; qui demande à l'utilisateur des coordonnées i et j et les filtre jusqu'à ce qu'elles soient correctes (comprises dans les bornes autorisées et correspondant à une case non encore dévoilée) ; puis qui renvoie ces coordonnées une fois correctes. **Attention** : on ne redemande que la coordonnée incorrecte s'il n'y en a qu'une. *Exemples d'interactions :*

```
A toi de jouer !
Ligne? 10
Ligne < 5 svp ? 18
Ligne < 5 svp ? 4
Colonne? 10
```

```
Colonne < 7 svp ? 7
Colonne < 7 svp ? 6

A toi de jouer !
Ligne? 3
Colonne? 4
Case deja devoilee, recommence
Ligne? 3
Colonne? 5
```

8. Écrire un programme principal qui :
   - Initialise une grille
   - Demande à l'utilisateur le nombre X de mines à placer et le filtre, puis place les X mines
   - Initialise la grille de booléens indiquant les cases dévoilées (pour l'instant aucune)
   - Affiche la grille de jeu
   - Demande à l'utilisateur un coup (filtre jusqu'à avoir des coordonnées correctes) et dévoile la case correspondante dans la grille de booléens
   - Vérifie s'il a perdu (touché une mine) : dans ce cas s'arrête et affiche la grille de jeu (y compris la mine touchée), puis la solution. *Par exemple :*

     ```
     Perdu, touche une mine !
     0??????
     01*????
     ???????
     ???????
     ???????
     La solution etait :
     -------
     --*-***
     -------
     ---**--
     ------*
     ```

   - Sinon affiche la grille de jeu, en remplaçant donc le ? de la nouvelle cellule dévoilée par le nombre de mines sur ses cases voisines
   - Recommence avec un nouveau coup
   - S'arrête dès que le joueur perd (en touchant une mine), ou une fois qu'il a dévoilé toutes les cases sauf les mines (dans ce cas il a gagné). *Par exemple :*

     ```
     Coup numero 28
     A toi de jouer !
     Ligne? 4
     Colonne? 6
     1?22110
     23?2?10
     ?224320
     221??10
     ?112210
     Tu as gagne en 28 coups, bravo !
     ```

   - **Attention :** ce programme principal doit utiliser les fonctions déjà codées ci-dessus.

9. **Bonus** : quand le joueur choisit une case à découvrir qui n'est entourée d'aucune mine, découvrir automatiquement récursivement toutes les cases voisines qui n'ont pas de mines (c-à-d qu'on peut dévoiler automatiquement toutes les voisines d'une case dont on sait qu'elle a 0 mines sur ses voisines, et ainsi de suite pour les autres cases découvertes ainsi qui ont aussi 0 mines voisines).

1.9 Boucles `for` et maths

Exercices de révision sur les boucles **`for`** *et les listes, à thème "mathématiques". Ce sont des exos classiques d'examen.*

1.9.1 Diviseurs propres et nombres parfaits

1. Écrire une fonction qui reçoit un entier naturel n et renvoie la liste de ses diviseurs (y compris 1 et lui-même).
2. Écrire une fonction qui reçoit un entier naturel n, et calcule et renvoie la somme de ses diviseurs propres (ses diviseurs autres que lui-même). Par exemple :

```
>>> sommeDivPropre(0)
0
>>> sommeDivPropre(1)
0
>>> sommeDivPropre(4)
3
>>> sommeDivPropre(6)
6
```

3. Un entier naturel est dit parfait s'il est égal à la somme de tous ses diviseurs propres (par exemple 6). Écrire une fonction qui teste si un entier naturel reçu en paramètre est parfait (en utilisant la fonction précédente).

```
>>> estParfait(6)
    True
>>> estParfait(10)
    False
```

4. Écrire une fonction qui reçoit deux entiers binf et bsup (optionnels, valeurs par défaut respectivement 2 et 100), et affiche tous les nombres parfaits de l'intervalle [binf,bsup], sur une seule ligne, séparés par un espace.

```
>> parfaits_entre()
Nombres parfaits de [2,100]
6 28
>> parfaits_entre(7)
Nombres parfaits de [7,100]
28
```

1.9.2 Nombres de Armstrong

Un nombre est dit de Armstrong s'il est égal à la somme des cubes de ses chiffres. Par exemple 153 est un nombre de Armstrong car $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$.

1. Écrire une fonction qui décompose un entier en la liste de ses chiffres, et renvoie cette liste.
2. Écrire une fonction *est_ Armstrong(nombre)* qui prend en argument un nombre et renvoie True si le nombre est de Armstrong, False sinon.
3. Écrire une fonction qui prend en argument une borne max et renvoie la liste des nombres de Armstrong inférieurs à cette borne.
4. Écrire un programme qui génère et affiche la liste des nombres de Armstrong inférieurs à 1000.

1.9.3 Triangle de Pascal

1. Écrire une fonction `factorielle` qui prend en argument un entier naturel `n` et qui renvoie la valeur de $n! = 1 \times 2 \times \cdots \times n$.
2. Écrire une fonction `coeff_binomial` qui prend en argument deux entiers naturels `n` et `k` (avec $k \leq n$) et qui renvoie la valeur du coefficient binomial correspondant, donnée par la formule ci-dessous :

$$\binom{n}{k} = \frac{n!}{(n-k)!\ k!}$$

3. Écrire une fonction `triangle_pascal` qui prend en argument un entier `nb_lignes` et qui affiche le triangle de Pascal (voir définition ci-dessous) en s'arrêtant au bout du nombre de lignes indiqué par l'argument. Voici un exemple du triangle de Pascal avec 6 lignes :

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

En numérotant les lignes et les colonnes à partir de zéro, le nombre sur la ligne numéro n et la colonne numéro k est le coefficient binomial $\binom{n}{k}$ (c-à-d le nombre de combinaisons de k éléments parmi n). Par exemple, pour la colonne numéro 0, on a toujours $\binom{n}{0} = 1$ (un seul choix, l'ensemble vide), et pour la colonne 1 on a toujours $\binom{n}{1} = n$ (n choix : tous les singletons différents). De même, lorsque $k = n$, on a $\binom{n}{n} = 1$ (un seul choix : l'ensemble complet), donc chaque ligne se termine par un 1.

4. *Bonus :* Comment faire pour éviter le décalage causé par les nombres à plusieurs chiffres ?

1.9.4 Approximation de la racine carrée

Étant donné un réel positif a, on définit la suite réelle x_n de la manière suivante. Cette suite converge vers $\sqrt{a}$.

$$\begin{cases} x_0 = a \\ x_{i+1} = \frac{x_i^2 + a}{2 * x_i} & pour\ i > 0 \end{cases}$$

1. Écrire une fonction `maRacine` qui reçoit un réel a et un entier n, qui calcule l'approximation de la racine de a par le n-ième terme de la suite ci-dessus, et renvoie cette valeur.

2. Écrire un programme principal qui demande à l'utilisateur un réel a et un entier n, calcule l'approximation de la racine avec la fonction ci-dessus, l'affiche, et affiche son carré pour vérifier si c'est une bonne approximation.

3. Écrire une fonction `precisionRacine` qui a partir d'un réel a et d'un entier n reçus en paramètres, calcule la valeur absolue de la différence entre l'approximation (obtenue par la fonction de la première question) et la racine de a (obtenue par la fonction sqrt du module math, ou avec l'opérateur de puissance **).

4. Écrire une fonction `maRacinePrec` qui reçoit un réel a et un réel p représentant la différence maximale souhaitée entre l'approximation de la racine et sa valeur réelle. Cette fonction calcule l'approximation de la racine de a et sa précision, avec un n de plus en plus grand, jusqu'à obtenir la précision voulue. Elle renvoie alors la valeur de n nécessaire pour obtenir cette "bonne" approximation.

1.9.5 Approximation de e

Un étudiant e1 place 1 kopec dans une banque, qui le rémunère au taux de 100% à la fin d'une année (taux 1 tous les 1 ans). L'étudiant se retrouvera donc en possession de 2 kopecs au bout d'un an. Un deuxième étudiant e2 choisit de placer son kopec dans une banque lui offrant un taux de rémunération de 50% tous les six mois (taux 0.5 tous les 0.5 ans). Ce dernier se retrouvera donc en possession de 2,25 kopecs à la fin de l'année (1.5 * 1.5 * 1).

1. Écrire une fonction qui calcule ce que l'étudiant e_n, qui a placé son kopec dans une banque lui offrant un taux de rémunération de 1/n toutes les 1/n années, possède à la fin de l'année.
   Cette fonction calcule en fait des valeurs approchées du nombre e (appelé nombre exponentiel, nombre d'Euler, ou constante de Neper), qui vaut environ 2,718281828459045.

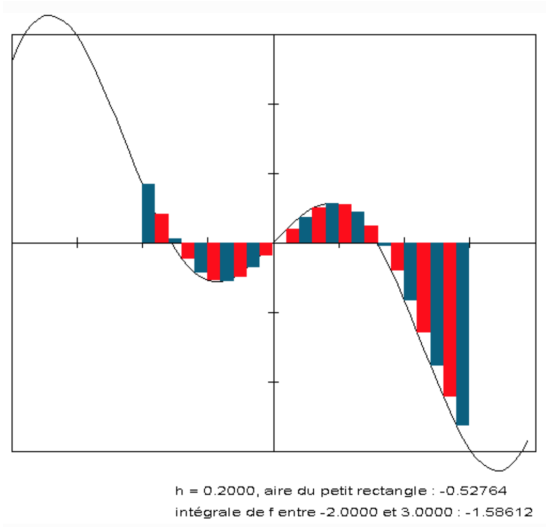
2. Écrire une fonction `estim_incond` qui demande à l'utilisateur un entier nmax, et affiche successivement toutes les approximations (calculées avec la fonction ci-dessus) pour toutes les valeurs de n entre 1 et nmax.

```
nmax?10
Approx avec 1 = 2.0
Approx avec 2 = 2.25
Approx avec 3 = 2.3703703703703702
Approx avec 4 = 2.44140625
Approx avec 5 = 2.48832
Approx avec 6 = 2.5216263717421135
Approx avec 7 = 2.546499697040712
Approx avec 8 = 2.565784513950348
Approx avec 9 = 2.5811747917131984
Approx avec 10 = 2.5937424601000023
```

3. Écrire une fonction `estim_cond` qui affiche toutes les estimations successives jusqu'à une précision de 0.01 (c-à-d que la distance entre e et son approximation doit être inférieure ou égale à 0.01).

1.9.6 Approximation de l'intégrale d'une fonction

On veut calculer l'approximation de l'intégrale [1] de la fonction cosinus sur un intervalle donné, par la méthode des rectangles, pour un nombre n de rectangles reçu en paramètre. Par exemple pour la fonction cosinus sur l'intervalle [-2,3], on obtient le graphique ci-dessous. La formule à utiliser pour le calcul est la suivante, où n est la largeur des rectangles, donc plus n est petit, plus le calcul est précis.



$$\int_a^b f(x)dx \approx \frac{b-a}{n} \sum_{i=1}^{n} f\left(a + i\frac{b-a}{n}\right)$$

1. Le package `scipy.integrate` fournit des fonctions de calcul d'intégrales, cf `https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html`

1.10 Fonctions avancées

Paramètres optionnels

1.10.1 Affichage d'une liste

Écrire une fonction qui reçoit une liste, et un booléen optionnel. Cette fonction affiche la liste à l'endroit si le booléen est vrai (par défaut), à l'envers sinon.

1.10.2 Sauvons la planète...

Écrire une fonction qui reçoit en paramètre un nombre de kilomètres à parcourir, un nombre de passagers (par défaut 1), un booléen indiquant s'il y a des embouteillages (par défaut non), et qui renvoie le meilleur moyen de transport (voiture, vélo, ou tram) selon la situation. On considère que la voiture n'est pas recommandée à moins de 2 passagers, ni pour moins de 10km, ni quand il y a des embouteillages. Le vélo est recommandé jusqu'à 10 km (inclus). On pourra rajouter d'autres paramètres optionnels : niveau de forme, météo, etc.

1.10.3 Le menu

Un restaurant possède trois menus : le `Basique` à 9€, le `Gourmand` à 15€ et le `Complet` à 19€. Pour chacun des menus, le client peut choisir de rajouter une boisson à 4€. De plus, le client peut choisir un supplément fromage et/ou un supplément café, chaque supplément coûtant 1,50€.

1. Écrire une fonction `prix_menu` qui prend comme argument le nom du menu, puis deux arguments optionnels : un booléen `avecBoisson` dont la valeur par défaut est `False`, et un entier `nb_supplement` qui doit valoir 0 par défaut. La fonction doit renvoyer le prix du menu correspondant.
2. Écrire un programme principal qui calcule puis affiche le prix total de l'addition pour la table suivante :
   - Jacqueline a choisi seulement un menu *Basique*,
   - Michel a pris un menu *Gourmand* avec boisson,
   - Johanna a choisi un menu *Basique* avec suppléments fromage et café,
   - et Antoine a choisi un menu *Basique* avec boisson et supplément café.

1.10.4 Intervalle avec paramètre optionnel

1. Écrire une fonction `afficheIntervalle` qui reçoit en paramètre 2 entiers a et b, et qui affiche les entiers de l'intervalle [a,b].
2. Que se passe-t-il si on appelle cette fonction avec un seul paramètre ?
3. Écrire une nouvelle fonction `afficheIntervalle2` qui reçoit en paramètre 2 entiers, le 2e étant optionnel (valeur par défaut = None). Cette fonction affiche les entiers de l'intervalle [a,b], ou uniquement l'entier a s'il y a un seul paramètre. On appellera la première fonction.
4. Écrire une troisième fonction `afficheIntervalle3` qui reçoit 2 paramètres optionnels (entiers a et b), et affiche les entiers de l'intervalle [a,b]. Si b est absent elle n'affiche que a. SI a est aussi absent elle n'affiche rien. On appellera la 2e fonction.

1.10.5 Tables de multiplication

1. Écrire une fonction qui affiche la table de multiplication d'un entier reçu en paramètre, de la manière suivante.

```
>>> table(3)
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
```

2. Écrire une nouvelle version de cette fonction pour pouvoir préciser deux paramètres optionnels : la valeur de début (par défaut 1) et la valeur de fin (par défaut 10) des multiplicateurs de la table de multiplication affichée.

```
>>> table(3, 2, 4)
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
```

3. Écrire une troisième version de cette fonction qui reçoit un 3e paramètre optionnel : le pas entre les multiplicateurs considérés (par défaut 1). Par exemple :

```
>>> table(3, 2, 4, 2)
3 x 2 = 6
3 x 4 = 12
```

1.10.6 Les moustiques

Depuis le début de l'année 2017, deux scientifiques Marc et Alice étudient l'évolution d'une population de moustiques sur l'île Chépaou. Ils ont réussi à obtenir l'estimation suivante sur l'évolution de la population : si la population contient x moustiques au cours d'une année, alors il y a aura $1.09x - 200$ moustiques l'année suivante. Par contre, ils ne sont pour l'instant pas d'accord sur l'estimation de la population en 2017 : ils s'accordent seulement sur le fait que ce nombre est compris entre 8 000 et 12 000. Il faudra donc considérer cette donnée comme une variable.

1. Écrire une fonction `f` qui prend en argument le nombre x de moustiques à une certaine année, et qui renvoie le nombre de moustiques l'année suivante.
2. Écrire une fonction `nb_moustiques` qui prend en arguments `nb_debut`, le nombre estimé de moustiques en 2017, et un entier `annee_voulue`. La fonction doit renvoyer le nombre de moustiques qu'il y aura en `annee_voulue`.
3. Écrire une fonction `annee_atteindra` qui prend en argument un entier `seuil` et un entier `nb_debut` (qui correspondra au nombre de moustiques en 2017) et qui renvoie l'année à partir de laquelle le nombre de moustiques sera supérieur ou égal à `seuil`.
4. Écrire un programme principal qui demande à Marc son estimation du nombre de moustiques actuellement, puis à Alice la sienne. Votre programme demandera ensuite une année et affichera le nombre de moustiques qu'il y aura cette année-là, selon l'estimation initiale de Marc puis selon celle d'Alice. Enfin, votre programme demandera un seuil et affichera en quelle année on atteindrait ce seuil, en fonction de chacune des deux estimations initiales.
5. Marc et Alice se rendent compte que leur formule pour l'évolution de la population de moustiques n'était pas correcte. Ils souhaitent la remplacer par $1.05x - 150$. Quelle partie de votre programme doit être modifiée ?

Retours multiples

1.10.7 Occurrences

- Écrire une fonction qui reçoit une chaîne de caractères, et renvoie le caractère apparaissant le plus souvent dans cette chaîne, ainsi que son nombre d'occurrences.
- Écrire un programme principal qui lit une chaîne au clavier, puis affiche le résultat dans un message clair.

1.10.8 Modulo

- Écrire une fonction qui reçoit 2 entiers a et b, et calcule et renvoie le quotient et le reste de la division de a par b
- Écrire un programme principal qui lit 2 entiers au clavier, puis affiche le résultat avec un message explicite.

1.10.9 Liste d'entiers

1. Écrire une fonction `LireListeEntiers` sans argument qui permet de lire des entiers positifs ou nuls saisis au clavier. La saisie s'arrête lorsque l'on entre un nombre négatif, et la fonction renvoie la liste des entiers saisis.
2. Écrire le programme principal qui permet d'appeler cette fonction et d'afficher la liste renvoyée.
3. Écrire une fonction `LireListeReelsBornes` avec deux arguments facultatifs bmin et bmax (valeurs par défaut 0 et 100 respectivement), qui lit des réels saisis au clavier, jusqu'à ce que l'utilisateur saisisse une valeur hors de l'intervalle délimité par les 2 bornes (inclues). La fonction renvoie la liste des éléments (corrects) saisis.
4. Définir une fonction *MMSListe* qui prend en argument une liste et qui renvoie le minimum, le maximum, et la somme des éléments de la liste.
5. Écrire un programme principal qui lit une liste de réels au clavier ; en calcule le minimum, le maximum et la somme ; et les affiche dans la console.

1.11 Dictionnaires

1.11.1 Les bases : dictionnaire à clés numériques

Soit le dictionnaire suivant :

```
d = {
      1: [1, 2, 3],
      2: [1, 4, 9],
      'autres': {
          3: None,
          4: [1, 16],
          5: [1, 32]
      }
  }
```

1. Que valent les expressions suivantes ?
   a) d[1] b) d[2][-1] c) d[d[1][1]] d) d[len(d['autres'][4])]
2. On veut supprimer l'élément qui vaut None, donner l'instruction qui permet de le faire.
3. On veut rajouter l'entier 81 à la liste associée à la clé 4. Quelle instruction permet de le faire ?
4. On veut mettre les éléments qui sont dans le dictionnaire $d['autres']$ dans le dictionnaire d, puis supprimer $d['autres']$. Donner le code qui permet de le faire sans utiliser for, puis en utilisant for.

1.11.2 Compteurs alphabétiques

Utiliser un dictionnaire pour compter le nombre d'occurrences de chaque caractère dans une chaîne de caractères reçue en argument. Le dictionnaire ne doit contenir que les caractères qui apparaissent dans la chaîne. On ne veut parcourir la chaîne qu'une seule fois.

1.11.3 Dictionnaire d'anniversaires

Soit le dictionnaire suivant qui associe à chaque personne (combinaison nom prénom supposée unique) sa date de naissance sous la forme d'une liste de valeurs (jour, mois, année).

```
d = {
    'Michel Durand' : [20, 9, 1963],
    'Sylviane Flinch' : [13, 5, 1977],
    'Lucille Doré' : [30, 4, 2005],
    'Kilian Dupont' : [7, 12, 1999]
}
```

1. Comment récupérer la date de naissance de Kilian Dupont ?
2. Comment ajouter votre date de naissance dans ce dictionnaire ?
3. Comment effacer une date de naissance de ce dictionnaire ?
4. Écrire une fonction `afficheDate` qui reçoit en paramètre ce dictionnaire, demande à l'utilisateur un nom de famille puis un prénom, et affiche la date de naissance sous la forme suivante : 'PRENOM NOM est né(e) le JJ/MM/AAAA' (en remplaçant les variables en majuscules par leur valeur), ou 'personne inconnue' si elle n'est pas dans le dictionnaire.
5. Écrire une fonction `listeNatifs` qui reçoit en paramètre ce dictionnaire et une année, et qui renvoie la liste des personnes nées cette année (ou une liste vide s'il n'y en a aucune).

1.11.4 Annuaire des plaques d'immatriculation

Soit le dictionnaire suivant qui stocke pour chaque personne (identifiée par la chaîne prénom + nom, supposée unique) son numéro de téléphone (sous forme de chaîne de caractères) et sa plaque d'immatriculation (une autre chaîne de caractères). Les valeurs de ce dictionnaire sont donc de type dictionnaire.

```
d = { 'Michel Durand' : {'tel': '0606060606','plaque': '718 YC 971'},
    ... }
```

1. Comment affiche-t-on la plaque d'immatriculation de la personne dont le nom (complet) est dans la variable `chauffeur` ?

2. Bonus : comment accède-t-on à son numéro de département ? (on suppose la plaque à l'ancien format, avec le numéro de département en dernier, comme dans l'exemple)

3. Écrire une fonction `identifierProprio` qui reçoit en paramètre ce dictionnaire et une plaque d'immatriculation, et qui renvoie le nom et le numéro de téléphone du propriétaire.

4. Comment change-t-on la plaque d'immatriculation d'un chauffeur quand il change de voiture ?

5. Comment change-t-on le nom du propriétaire d'une voiture (d'une plaque d'immatriculation) lorsqu'elle est vendue ? (et donc aussi le numéro de téléphone associé)

1.11.5 Personnaliser ses pokémons

Dans un jeu vidéo Pokémon, on souhaite donner des noms personnalisés à certains types de pokémons. Pour ce faire, on définit une fonction *noms_perso(pokemons, noms)*, qui prend en argument une liste de pokémons attrapés, et un dictionnaire faisant correspondre des noms de pokémons originaux à leurs nouveaux noms. La fonction renvoie une nouvelle liste de pokémons, avec les noms modifiés. Les noms qui n'apparaissent pas dans le dictionnaire restent inchangés. Exemple :

```
>>> mes_pokemons = ["pikachu", "bulbizarre", "roucarnage", "lippoutou"]
>>> mes_noms = {"bulbizarre": "jean-jacques", "roucarnage": "gros pigeon"}
>>> noms_perso(mes_pokemons, mes_noms)
["pikachu", "jean-jacques", "gros pigeon", "lippoutou"]
```

1.11.6 Dictionnaire français-anglais

1. Écrivez une fonction *ajoute(mot1, mot2, d)* qui prend en argument un mot en français *mot1*, sa traduction en anglais *mot2* et ajoute ces deux mots dans le dictionnaire *d* uniquement si *mot1* n'est pas déjà une clé du dictionnaire (dans ce cas afficher un message d'erreur).

2. Écrivez une fonction *affiche(d)* qui prend en argument un dictionnaire et affiche à l'écran toutes les valeurs correspondant à ses clés.

3. Écrivez une fonction *supprime(car, d)* qui prend en argument un caractère *car* et un dictionnaire *d* et renvoie un nouveau dictionnaire ne contenant pas les entrées du dictionnaire *d* correspondant à des clés qui commencent par la lettre *car*.

4. Écrivez un programme principal qui :

   - Demande à l'utilisateur un dictionnaire dont les clés sont 5 mots de la langue française et les valeurs correspondent à la traduction en anglais de chacun de ces mots (on appellera la fonction `ajoute`)
   - Affiche ce dictionnaire (avec la fonction `affiche` ci-dessus)
   - Lit ensuite un mot français (le filtre pour qu'il ne soit pas déjà dans le dictionnaire), demande ensuite sa traduction en anglais, et l'ajoute au dictionnaire
   - Lit une lettre de l'alphabet, et supprime du dictionnaire les mots commençant par cette lettre. Est-il nécessaire de filtrer la lettre lue ?
   - Affiche à nouveau le dictionnaire

1.11.7 Polynômes

Dans cet exercice on veut travailler avec des polynômes de degrés quelconques. On peut représenter chaque polynôme avec un dictionnaire, dont les clés correspondent aux puissances de x, et les valeurs aux coefficients. Par exemple, pour représenter le polynôme $x^6 + 3x^2$, on peut utiliser le dictionnaire : $\{6 : 1, 2 : 3\}$.

1. Écrire une fonction *lire(n)* qui reçoit un entier n (le degré du polynôme), et qui demande à l'utilisateur les différents coefficients dans l'ordre décroissant des degrés (ordre intuitif d'écriture), et qui renvoie le dictionnaire représentant ce polynôme. Dans l'exemple ci-dessous pour $n = 3$, l'utilisateur saisit le polynôme $2*x^3-5*x^2+x-7$.

```
Coeff de x**3? 2
Coeff de x**2? -5
Coeff de x**1? 1
Coeff de x**0? -7
{3:2, 2:-5, 1:1, 0:-7}
```

2. Écrire une fonction `evaluer(p, x)` qui prend un polynôme p et un nombre x en arguments, et renvoie la valeur du polynôme au point x.

3. Écrire une fonction `simplifier(p)` qui reçoit en argument un polynôme p (sous forme d'un dictionnaire), et modifie ce dictionnaire pour le simplifier, en supprimant les clés dont la valeur est 0. Cette fonction ne renvoie rien. *Par exemple la simplification du polynôme 2 :1, 1 :0, 0 :-3 est le dictionnaire 2 :1, 0 :-3.*

4. Écrire une fonction `affiche(p)` qui reçoit un polynôme et l'affiche "joliment". Par exemple le dictionnaire 2 :1, 1 :3, 0 :-3 sera affiché : $x^2 + 3 * x - 3$. On remarque qu'on n'affiche pas les monômes nuls, ni le coefficient multiplicateur ou la puissance si égal(e) à 1. On peut commencer par une version simple sans cas particulier, qui affichera alors : $1 * x^2 + 3 * x^1 - 3 * x^0$.

5. Écrire une fonction *somme_polynomes(p1, p2)* qui prend deux polynômes (dictionnaires) en arguments et qui renvoie un nouveau dictionnaire représentant la somme des deux polynômes p1 et p2. Attention aux effets de bord indésirables, il ne faut pas modifier p1 et p2.

6. Écrire une fonction *produit_polynomes(p1, p2)* qui prend deux polynômes en arguments et renvoie le produit des deux polynômes dans un nouveau dictionnaire. Attention aux effets de bord indésirables, il ne faut pas modifier p1 et p2.

7. Écrire une fonction *derivee_polynome(p)* qui reçoit en argument un polynôme et qui renvoie sa dérivée. Attention on ne doit pas modifier p, il faut créer un nouveau dictionnaire.

```
Exemple d'exécution:
>>> evaluer({3:1, 1: 2, 0: -1}, 2)
11
>>> somme_polynomes({3:1, 2:1, 0:1}, {4:2, 2:3})
{0: 1, 2: 4, 3: 1, 4: 2}
>>> produit_polynomes({3:1, 2:1, 0:1}, {4:2, 2:3})
{2: 3, 4: 5, 5: 3, 6: 2, 7: 2}
>>> somme_polynomes({2:1, 1:1, 0:1}, {2:1, 1:-1, 0:1})
{0: 2, 2: 2}
>>> derivee({5:1,3:2,2:-1,1:7,0:-4})
{4: 5, 2: 6, 1: -2, 0: 7}
```

1.11.8 Acides aminés

En utilisant un dictionnaire, déterminez le nombre d'occurrences de chaque acide aminé dans la séquence AGWPSGGASAGLAILWGASAIMPGALW. Le dictionnaire ne doit contenir que les acides aminés présents dans la séquence.

1.11.9 Séquençage génétique

1. Créez une fonction `compte2(seq)` qui prend comme argument une séquence sous la forme d'une chaîne de caractères et qui renvoie tous les mots de 2 lettres qui existent dans la séquence sous la forme d'un dictionnaire. *Par exemple pour la séquence ACCTAGCCCTA, le dictionnaire renvoyée serait : 'AC' : 1, 'CC' : 3, 'CT' : 2, 'TA' : 2, 'AG' : 1, 'GC' : 1*

2. Créez une nouvelle fonction `compten(seq,n)` qui a un comportement similaire mais qui compte tous les mots de n lettres, et renvoie le dictionnaire résultant.

3. Écrire un programme principal utilisant ces fonctions pour afficher les mots de 2 et 3 lettres et leurs occurrences trouvés dans la séquence d'ADN suivante : ACCTAGCCATGTAGAATCGCCTAGGCTTTAGCTAGCTCTAGCTAGCTG

1.11.10 Gestion des notes (simple)

On considère le dictionnaire suivant contenant les notes d'une classe en informatique. Les clés sont les noms des étudiants (supposés uniques), les valeurs sont des listes de notes (à tous les examens d'informatique du semestre). Un étudiant qui a raté un examen aura moins de notes dans sa liste.

```
notes_info = {
    'toto': [12.5, 15, 10, 14],
    'titi': [13, 17, 12, 14.5, 16],
    ...}
```

1. Écrire une fonction `compteNotes` qui reçoit en paramètres ce dictionnaire et le nom d'un étudiant et qui renvoie le nombre d'examens qu'il a passés.
2. Écrire une fonction `meilleureNote` qui reçoit en paramètres ce dictionnaire et le nom d'un étudiant et qui renvoie sa meilleure note.
3. Écrire une fonction `meilleurSemestre` qui reçoit en paramètre ce dictionnaire, et renvoie le nom et la note de l'étudiant qui a eu la meilleure note du semestre (tous examens confondus).
4. Écrire une fonction `plusAbsent` qui reçoit en paramètre ce dictionnaire, et renvoie le nom et le nombre d'examens passés par l'étudiant qui a été le plus absent (celui qui a le moins de notes dans sa liste).

1.11.11 Gestion des notes d'une classe (bis)

On veut gérer les notes d'examens des étudiants d'un groupe. Tous les étudiants du groupe ne suivent pas forcément les mêmes cours, ni le même nombre de cours. Les notes sont organisées dans une liste, dont chaque élément est un dictionnaire contenant le nom de l'étudiant et ses notes dans les matières qu'il a choisies. Exemple :

```
notes_groupe=[
    {'nom': 'lambert', 'notes': {'physique': 12, 'info': 11}},
    {'nom': 'meng', 'notes': {'maths': 13.5, 'info': 18, 'sport': 17}},
    ...]
```

1. Écrire une fonction *moyenne_ etudiant(etudiant)* qui prend en argument un élément de la liste décrite précédemment et qui retourne la moyenne de l'étudiant.
2. Écrire une fonction *meilleur(notes_ groupe)* qui prend la liste du groupe en argument et retourne la meilleure moyenne du groupe ainsi que le nom de l'étudiant qui l'a obtenue.
3. (Difficile) Écrire une fonction *matiere_ difficile(notes_ groupe)* qui retourne le nom de la matière la plus difficile (celle qui a la pire moyenne de groupe).

1.11.12 Gestion des notes (ter)

On travaille avec une liste de dictionnaires de notes pour les étudiants, dont chacun suit un ensemble différent de matières. Le dictionnaire de chaque étudiant a les clés suivantes :

- nom : la valeur est le nom de l'étudiant
- notes : la valeur est un dictionnaire stockant les notes de cet étudiant dans chaque matière (clé = nom de la matière, valeur = la note)
- coeffs : la valeur est un dictionnaire associant chaque nom de matière (clé) avec le coefficient de cette matière pour cet étudiant (valeur) selon son parcours

Par exemple :

```
[{'nom' : 'felix', 'notes' : {'maths' : 20, 'info':12},'coeffs':{'maths':9,
'info':4}},{'nom':'arthur','notes':{'maths':18,'physique':17}, 'coeffs':{'maths':6,
'physique':6}},{'nom':'roxanne', 'notes':{'maths':19,'info':16}, 'coeffs':
{'maths':9,'info':4}},{'nom':'lucien', 'notes':{'maths':17,'physique':19},
'coeffs':{'maths':3,'physique':8}},{'nom':'alice','notes':{'info':18,
'physique':15},'coeffs':{'info':2,'physique':8}}, {'nom':'maxence','notes':
{'physique':8},'coeffs':{'physique':4}}]
```

1. Écrire une fonction qui reçoit un nom d'étudiant, et la liste de dictionnaires, et renvoie le dictionnaire qui est associé à cet étudiant

2. Écrire une fonction qui reçoit un nom d'étudiant et la liste de dictionnaires, et calcule et renvoie sa moyenne en tenant compte des coefficients de chaque matière. On appellera la fonction précédente pour obtenir d'abord le dictionnaire de cet étudiant.

3. Écrire une fonction auxiliaire qui appelle la fonction précédente et crée et renvoie un dictionnaire associant à chaque étudiant sa moyenne.

4. Écrire une fonction qui reçoit la liste de dictionnaires et renvoie le nom du meilleur étudiant (meilleure moyenne générale) et sa moyenne. Cette fonction utilise les précédentes.

5. Écrire une fonction qui reçoit la liste et un nom de matière, et renvoie le nom de l'étudiant qui a la meilleure note dans cette matière, et sa note

6. Écrire une fonction qui reçoit la liste et renvoie le nom de la matière la plus difficile (avec la plus mauvaise moyenne de la classe) ainsi que la moyenne

7. Écrire une fonction qui reçoit la liste de dictionnaires de chaque étudiant, et qui calcule et renvoie une nouvelle liste de dictionnaires, chaque dictionnaire représente une matière, la clé 'nom' doit avoir pour valeur le nom de la matière, et la clé 'notes' doit avoir pour valeur un dictionnaire associant chaque nom d'étudiant (clé) à sa note dans cette matière. *Par exemple :*

   ```
   [{'nom' : 'maths', 'notes' : {'felix':20, 'arthur':18, 'lucien':17,
   'roxanne':19}},{'nom' : 'info', 'notes' : {'felix':12, 'roxanne':16,
   'alice':18}}, {nom : 'physique', 'notes' : {'arthur':17, 'lucien':19,
   'alice':15, 'maxence':8}}]
   ```

8. Écrire une fonction qui reçoit la liste des dictionnaires des étudiants, utilise la fonction ci-dessus pour calculer la liste des dictionnaires des matières, puis parcourt cette liste pour calculer et afficher pour chaque matière le nombre d'étudiants et la moyenne de classe. *Par exemple :*
   maths, 4 étudiants, moyenne 18.5

1.11.13 Transformer un dictionnaire en liste triée

Écrivez une fonction qui prend en argument un dictionnaire dont les clés sont des entiers, et qui renvoie la liste des valeurs de ce dictionnaire, triée selon l'ordre croissant de leur clé.

1.11.14 Rendez-vous

Alice et Bob ont tous les deux un emploi du temps électronique. Chaque évènement est modélisé par un dictionnaire qui contient trois clés : *"debut"*, *"fin"* et *"titre"*. Les valeurs associés aux deux premiers sont des horaires exprimés par un nombre entier de minutes écoulés depuis minuit, la valeur associée au dernier est une chaîne de caractères. Un emploi du temps est une liste qui regroupe les évènements d'une journée donnée, **classée dans l'ordre chronologique**, et dont aucun ne se chevauche. Exemple d'emploi du temps :

```
edt = [{"debut": 9*60, "fin": 11*60, "titre": "Réunion"},
       {"debut": 12*60, "fin": 13*60, "titre": "Déjeuner avec John"},
       {"debut": 16*60, "fin": 16.5*60, "titre": "cours de piano"}]
```

1. Écrivez une fonction *espaces_ disponibles(edt, duree)* qui prend en argument un emploi du temps et une durée en minutes (entier), et qui renvoie la liste des indices pour lesquels un évènement de durée *duree* pourrait être ajouté avec l'opérateur *insert* sans provoquer de chevauchement.

   Exemple d'exécution :

   ```
   >>> espaces_disponibles(edt, 2*60)
   [0, 2, 3]
   >>> espaces_disponibles(edt, 6*60)
   [0, 3]
   ```

2. Alice souhaite donner rendez-vous à Bob le plus tôt possible dans la journée après son premier évènement mais avant son dernier, sans rien avoir à décaler dans les deux emplois du temps. Écrire une fonction *ajouter_rdv(edtA, edtB, duree, titre)* qui prend en argument les emplois du temps d'Alice et de Bob, une durée en minutes (entier) et un titre (chaîne de caractères), et qui ajoute le rendez-vous dans les deux emplois du temps, si c'est possible. La fonction renvoie un booléen qui indique si le rendez-vous a été ajouté.

   Exemple 1 :

```
>>> edtA = [{"debut": 540, "fin": 660, "titre": "Reunion"},
            {"debut": 720, "fin": 780, "titre": "Déjeuner avec John"}]
>>> edtB = [{"debut": 480, "fin": 600, "titre": "Rdv avec Jack"},
            {"debut": 720, "fin": 900, "titre": "Pot de retraite de Joe"}]
>>> ajouter_rdv(edtA, edtB, 1*60, "Rdv Alice Bob")
True
>>> edtA
[{'titre': 'Reunion', 'fin': 660, 'debut': 540},
 {'titre': 'Rdv Alice Bob', 'fin': 720, 'debut': 660},
 {'titre': 'Déjeuner avec John', 'fin': 780, 'debut': 720}]
>>> edtB
[{'titre': 'Rdv avec Jack', 'fin': 600, 'debut': 480},
 {'titre': 'Rdv Alice Bob', 'fin': 720, 'debut': 660},
 {'titre': 'Pot de retraite de Joe', 'fin': 900, 'debut': 720}]
```

   Exemple 2 :

```
>>> edtA = [{"debut": 540, "fin": 660, "titre": "Reunion"},
            {"debut": 720, "fin": 780, "titre": "Déjeuner avec John"}]
>>> edtB = [{"debut": 480, "fin": 600, "titre": "Rdv avec Jack"},
            {"debut": 720, "fin": 900, "titre": "Pot de retraite de Joe"}]
>>> ajouter_rdv(edtA, edtB, 2*60, "Rdv Alice Bob")
False
```

3. Alice et Bob n'ont malheureusement pas pu placer leur rendez-vous, mais Alice souhaite vraiment rencontrer Bob. Elle décide de décaler ses autres évènements pour rencontrer Bob au premier moment de la journée où il est disponible, mais après 8h. Ecrivez une fonction *imposer_rdv(edtA, edtB, duree, titre)* qui permettrait de modifier les emplois du temps en conséquence.

   Exemple :

```
>>> edtA = [{"debut": 540, "fin": 660, "titre": "Reunion"},
            {"debut": 720, "fin": 780, "titre": "Déjeuner avec John"}]
>>> edtB = [{"debut": 480, "fin": 600, "titre": "Rdv avec Jack"},
            {"debut": 720, "fin": 900, "titre": "Pot de retraite de Joe"}]
>>> imposer_rdv(edtA, edtB, 2*60, "Rdv Alice Bob")
>>> edtA
[{'fin': 720, 'titre': 'Rdv Alice Bob', 'debut': 600},
 {'fin': 840, 'titre': 'Reunion', 'debut': 720},
 {'fin': 900, 'titre': 'Déjeuner avec John', 'debut': 840}]
>>> edtB
[{'fin': 600, 'titre': 'Rdv avec Jack', 'debut': 480},
 {'fin': 720, 'titre': 'Rdv Alice Bob', 'debut': 600},
 {'fin': 900, 'titre': 'Pot de retraite de Joe', 'debut': 720}]
```

1.12 Fichiers

1.12.1 Groupe d'étudiants

Écrire un programme qui demande le nombre d'étudiants, puis les prénoms de tous les étudiants d'une classe, et enregistre ces prénoms dans un fichier, un par ligne.

1.12.2 Groupe d'étudiants (bis)

Écrire un programme qui lit un fichier etudiants.txt (qu'on suppose existant) et affiche les données suivantes :

- Longueur du prénom le plus long dans le fichier
- Longueur moyenne des prénoms dans le fichier
- Prénom le plus représenté (on commencera par créer un dictionnaire pour compter le nombre d'occurrences de chaque prénom) et son nombre d'occurrences
- Nombre de prénoms représentés plusieurs fois

1.12.3 Analyse de texte

On veut connaître les lettres de l'alphabet qui sont les plus utilisées dans la langue française. Pour ce faire, on regroupe dans un fichier texte un grand nombre de romans écrits en français (en minuscule) et on souhaite écrire un programme qui permet de compter la fréquence d'apparition de chaque lettre, en ignorant la ponctuation et les caractères spéciaux.

1. Écrire une fonction *lire_fichier(nom_fichier)*, qui prend le nom du fichier contenant le texte en argument, et retourne une chaîne de caractères contenant l'intégralité du texte.
2. Écrire une fonction `liste_spec` qui reçoit un texte et renvoie la liste de ses caractères non alphabétiques (pour simplifier on considère qu'il n'y a pas de lettres accentuées dans le texte).
3. Écrire une fonction *compter_ lettres(texte, exceptions)* qui prend un texte en argument et retourne un dictionnaire qui associe à chaque lettre sa fréquence d'apparition dans le texte (le nombre de fois où elle apparaît). L'argument exceptions est une liste contenant les caractères spéciaux à ignorer.
4. Écrire une fonction *lettre_ rare(frequences)* qui prend le dictionnaire construit dans la question précédente en argument, et retourne la lettre de l'alphabet la plus rare (celle qui apparaît, mais le moins souvent).
5. Écrire le programme principal qui demande à l'utilisateur le nom du fichier, l'ouvre pour le lire, affiche la lettre la plus rare et son nombre d'occurrences, et ferme le fichier.

1.12.4 Table harmonieuse

Harmonie a décidé d'inviter ses amis autour d'une seule grande table. Pour que tout soit parfait, elle veut qu'un homme ait au moins une femme à ses côtés et qu'une femme ait au moins un homme à ses côtes. Harmonie a beaucoup d'amis, ils viennent pour la plupart accompagnés. Pour ne pas se torturer l'esprit, elle choisit la méthode informatique pour vérifier que sa table respecte bien ses conditions. Elle dispose de fichiers textes contenant la liste des noms des invités (un fichier pour les hommes et un pour les femmes) et a écrit la disposition de la table dans un autre fichier.

1. Écrire une fonction *read_file(nom)* qui prend en argument le nom d'un fichier texte et renvoie une liste des lignes du fichier.
2. Utiliser cette fonction pour récupérer les fichiers :
   - les prénoms masculins (homme.txt)
   - les prénoms féminins (femme.txt)
   - la configuration de la table (table.txt)
3. Écrire une fonction *genre(prenom, femmes, hommes)* qui prend en argument un prénom, une liste des femmes et une liste des hommes. Cette fonction vérifie si le prénom est masculin ou féminin à l'aide des listes en entrée *femmes* et *hommes*. La fonction retourne `True` si le prénom est féminin, `False` si c'est un prénom masculin.

4. Écrire une fonction *contrainte(table, femmes, hommes)* qui prend en argument la liste des membres de la table et vérifie si la table de Harmonie suit les contraintes qu'elle s'est imposée ou non (un homme a au moins une femme à ses côtés et une femme a au moins un homme à ses côtes). *Attention* : le dernier membre est à côté du premier !

5. Bonus : écrire une fonction qui génère une configuration de table respectant les contraintes (on suppose que c'est possible). Cette fonction doit écrire la configuration dans un fichier tableAuto.txt .

1.12.5 Analyse de mesures

On considère tout au long de cet exercice des dictionnaires représentant des mesures réelles, de type *float*, faites à divers temps comptés en heures et minutes à partir d'un top de départ "00h00" et sur une durée totale ne pouvant pas dépasser "99h59". Par exemple, { "01h00" :0.5, "02h45" :1.55, "05h00" :2.5, "07h35" :18.5, "10h00" :0.2, "15h11" :4.0 }

Ces mesures sont supposées être les niveaux de concentration d'une protéine tout au long d'une expérience. La chaîne de caractères représentant le temps est normalisée comme dans l'exemple, de sorte que deux temps peuvent être comparés simplement comme des chaînes de caractères.

1. **Création d'un fichier de données**

   Écrire une fonction qui prend en argument le nom d'un fichier à créer, qui demande à l'utilisateur des données représentant les niveaux de concentration d'une protéine à différents temps, et qui écrit un fichier contenant ces informations sous la forme présentée dans la question suivante.

2. **Lecture dans un fichier**

   Ecrire la fonction chargement qui prend en entrée le nom d'un fichier dont le contenu a la forme parfaitement normalisée comme ci-dessous, et qui retourne le dictionnaire correspondant.

   01h00 = 0.5
   02h45 = 1.55
   05h00 = 2.5
   07h35 = 18.5
   10h00 = 0.2
   15h11 = 4.0

   Remarquez que, même si le fichier est trié par temps croissants, le dictionnaire ne l'est plus.
   INDICATION : la méthode .split(ch) appliquée à une chaîne de caractères renvoie la liste des sous-chaînes qui sont séparées par le motif ch. Par exemple,
   "a-c-d-et-r".split('-') renvoie ['a', 'c', 'd', 'et', 'r']

3. Ecrire une fonction *horloge* qui prend en entrée un dictionnaire d comme précédemment, et fournit en sortie la liste triée des temps du dictionnaire. **INDICATION** : la méthode .sort() appliquée à une liste renvoie une liste triée.

4. Ecrivez la fonction *afficheProfil* qui, à partir d'un dictionnaire représentant un profil, écrit à l'écran, son profil dans l'ordre croissant des temps.

   - Ecrivez la fonction *covariantes* qui prend en entrée deux dictionnaires représentant des mesures effectuées (simul- tanément) sur deux protéines differentes et retourne un booléen disant si elles augmentent et diminuent en même temps (sans pour autant avoir les mêmes mesures). On pourra par exemple d'abord écrire une fonction listeVariations qui prend en entrée un dictionnaire et renvoie la liste des variations (attention à l'ordre). Pour ceux qui veulent aller plus loin. Comment modifier votre code pour prendre en compte qu'une variation inférieure à 10% n'est pas significative ?
   - Ecrivez la fonction *cluster* qui prend en entrée :
     - d'une part un dictionnaire *ref* donnant les mesures pour une protéine de référence
     - et d'autre part, une liste l de dictionnaires représentant chacun les mesures pour une autre protéine (on supposera que les mesures sont simultanées à celles faites pour la protéine de référence)

     et fournit en sortie la liste des dictionnaires covariants avec la protéine de référence.

1.12.6 Fichier CSV

Le format CSV (*Comma-separated values*, valeurs séparées par des virgules) est un format de fichier texte permettant de stocker des données tabulaires, sous forme de valeurs séparées par des virgules. Chaque ligne du texte correspond à une ligne du tableau, et les virgules correspondent aux séparations entre colonnes. Chaque portion de texte entre les virgules correspond donc à une cellule du tableau.

1. Écrire une fonction qui reçoit le nom d'un fichier CSV, qui l'ouvre, lit son contenu, et crée une liste à 2 dimensions représentant son contenu sous forme tabulaire. Ne pas oublier de fermer le fichier après usage.

```
1,2,3
2,4,9            ==> [[1,2,3],[2,4,9],[3,8,27]]
3,8,27
```

2. On suppose maintenant que la première ligne du fichier fournit les noms des colonnes, et les lignes suivantes les valeurs. On veut écrire une fonction qui crée à partir de ce fichier une liste de dictionnaires, chaque dictionnaire représente une ligne de valeurs du fichier, les clés sont les noms présents sur la première ligne.

```
Nom,Naissance,Ville
'Toto',1977,'Ajaccio'             [{'Nom':'Toto','Naissance':1977,'Ville':'Ajaccio'},
'Alice',2000,'Montpellier'  ==>    {'Nom':'Alice','Naissance':2000,'Ville':'Montpellier'}
'Bob',1998,'Grenoble'              {'Nom':'Bob','Naissance':1998,'Ville':'Grenoble'}]
```

3. Écrire une fonction qui reçoit un dictionnaire et une chaîne de caractères, et crée un fichier CSV au format ci-dessus, nommé comme indiqué par la chaîne reçue en paramètre, pour représenter le contenu de ce dictionnaire.

1.12.7 Zoo et devinettes

On suppose qu'on dispose d'une liste de dictionnaires représentant des animaux sous la forme suivante :

```
[{'nom': 'girafe', 'poids': 1100, 'taille': 5.0},
 {'nom': 'singe', 'poids': 50, 'taille': 1.75},
 {'nom': 'guepard', 'poids': 50, 'taille': 1.20},
 {'nom': 'lion', 'poids': 200, 'taille': 1.20},
 ...]
```

1. Écrire une fonction qui reçoit une chaîne de caractères (nom d'un critère, par exemple 'poids' ou 'taille') et qui renvoie le nom de l'animal ayant la plus grande valeur de ce critère.

2. Écrire une fonction `devinette()` qui demande à l'utilisateur de penser à un animal, puis qui lui pose des questions pour deviner de quel animal il s'agit. Si plusieurs animaux ont la même valeur d'un critère, il faudra poser plusieurs questions pour raffiner. L'ordre des questions pourra être choisi aléatoirement.

```
Pense a un animal
Quel est le poids de ton animal ? 50
Quelle est la taille de ton animal ? 1.20
C'est : guepard !
```

3. Écrire une fonction `info()` qui répond aux questions de l'utilisateur en suivant le modèle d'exécution ci-dessous.

```
Infos sur les animaux
Choisis un animal parmi : girafe, singe, guepard, lion ? elephant
Animal inconnu
Choisis un animal parmi : girafe, singe, guepard, lion ? lion
Que veux-tu savoir ? age
Information inconnue
Que veux-tu savoir ? taille
taille de lion = 1.20 m
Que veux-tu savoir ? poids
poids de lion = 200 kg
Que veux-tu savoir ? stop
Choisis un animal parmi : girafe, singe, guepard, lion ? stop
Au-revoir
```

Chapitre 2

Sujets de TP

Chaque TP correspond à la mise en application des notions apprises pendant la semaine correspondante.

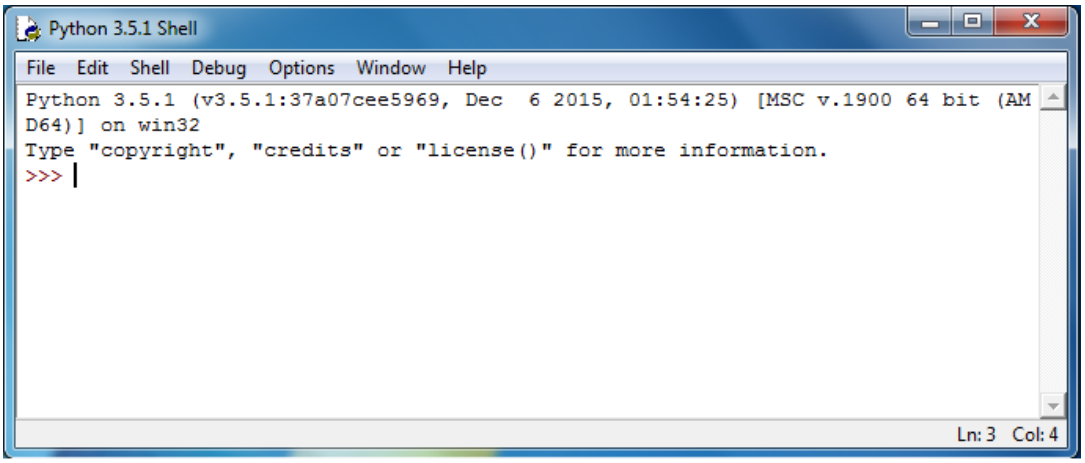
2.1 TP1 : prise en main des outils, lecture et premiers programmes

Notions pratiquées : entrées-sorties, instruction conditionnelle if L'objectif de ce TP est d'acquérir la maîtrise des outils qui sont à votre disposition ; pour bien comprendre ce qui est en jeu, faites tous les essais qui vous paraîtront nécessaires.

CaseIne : des exercices supplémentaires, des fichiers complémentaires pour certains TP, des diapos de cours, et autres ressources, sont disponibles sur le cours en ligne Python sur CaseIne, accessible avec vos identifiants de connection UGA. `http://caseine.org/course/view.php?id=86`

2.1.1 Première prise en main de Idle

`Idle` est un environnement de développement - c-à-d un outil permettant de faciliter la programmation - en `Python`. Lancer le programme `Idle`. `Idle` lance une fenêtre contenant plusieurs menus. Cette fenêtre est un interpréteur de commandes pour `Python`, vous pouvez y taper directement des lignes en `Python`. Cela se présente plus ou moins ainsi :



Dans la suite des TP, nous allons écrire nos programmes dans des fichiers et non directement dans cette fenêtre. Mais cette fenêtre reste utilisable directement à tout moment, par exemple pour tester certaines instructions. C'est pratique car nous obtenons directement le résultat (cela peut aussi vous servir de calculatrice puissante). Vous pouvez par exemple taper les lignes suivantes :

```
>>> 3 + 5
>>> "bon" + "jour"
>>> "3" + "5"
>>> 3 + "5"
>>> 3 + int("5")
>>> 1 + 1
>>> "1" + "1"
>>> a = 3
>>> print(a)
>>> type(a)
```

```
>>> a = a + 17
>>> b = a * 2
>>> print("a =", a, "et b =", b)
>>> a == b
>>> type(a == b)
>>> type(a = b)
>>> 17 // 5
>>> 17 / 5
>>> 17 % 5
>>> 17.0 / 5
>>> type(3.3)
```

Une ou plusieurs de ces lignes provoquent des erreurs, c'est normal. Surlignez-les. Avez-vous pris le temps de lire et de comprendre les messages d'erreur affichés ?

2.1.2 Écriture et exécution d'un programme sous Idle

- Dans le menu ***File*** (comme fichier) choisir ***New File***. `Idle` ouvre une fenêtre nommée "Untitled" dans laquelle on va pouvoir écrire le texte *source* du programme ; cela permet aussi de garder une trace de son travail. Commencez par enregistrer le fichier (*File > Save*) sous le nom `TP1_exo1.py`.

  Attention ! veillez à toujours sauver vos fichiers sur votre répertoire personnel (`home`). Il est conseillé d'y créer un dossier nommé ***INF101*** pour y ranger tous les TP de l'UE. Vous pouvez aussi copier votre travail sur une clé USB si vous en avez une.

- Écrire maintenant le texte du programme suivant :

  ```
  print("Premier programme")
  nom = input("Donnez votre nom : ")
  print("Bonjour", nom)
  ```

  Sauvegardez votre travail. Vous allez maintenant exécuter ce programme : pour cela allez dans le menu ***Run*** et choisissez ***Run Module*** (vous pouvez aussi appuyer directement sur la touche ***F5***).

  Le programme s'exécute alors dans la première fenêtre, celle de l'interpréteur `Python` (qui est nommée `Python 3.6.1 Shell`).

- Essayez maintenant d'introduire une erreur, c'est-à-dire modifiez le texte de façon à ne pas respecter les conventions du Python ; ensuite sauvegarder, puis demander l'exécution. Que se passe-t-il ?

- Ensuite, modifiez le texte du programme de façon que s'affiche à l'exécution :

  ```
  Premier programme, deuxieme version
  Donnez votre nom en majuscules :
  PAUL
  Bonjour, PAUL, comment vas-tu ?
  ```

  Faites des essais en modifiant le texte du programme jusqu'à obtention de l'affichage requis.

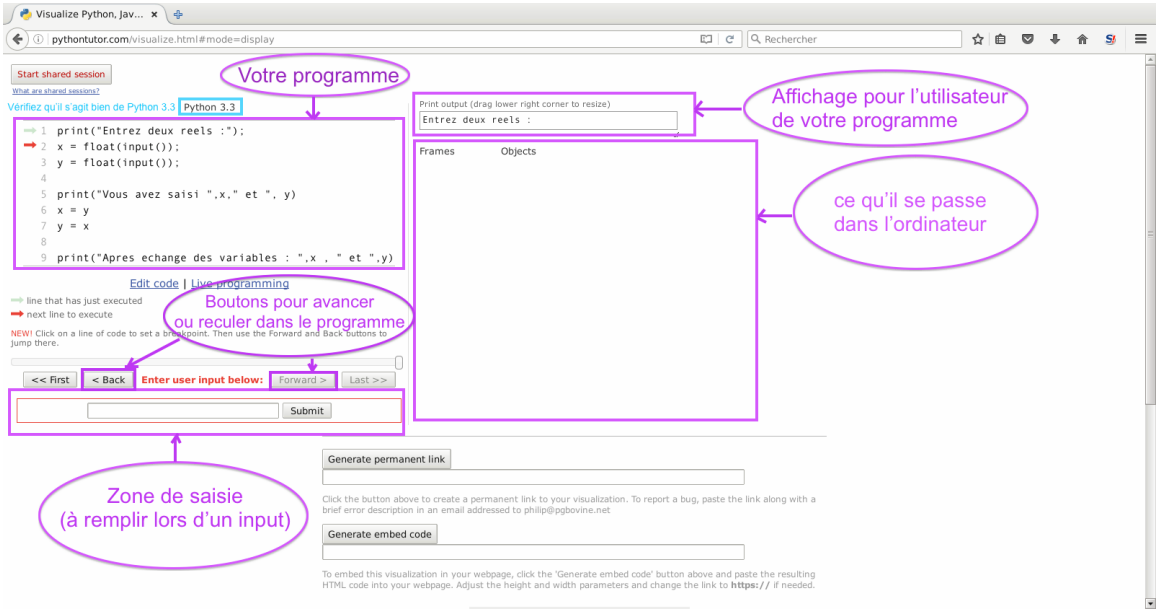
2.1.3 Découverte de Python Tutor

`Python Tutor` est un outil en ligne qui permet de mieux comprendre ce qui se passe quand l'ordinateur exécute un programme (en `Python` mais aussi en d'autres langages de programmation). Cela deviendra particulièrement utile au fil des semaines quand nous aborderons des notions de plus en plus complexes.

- Pour utiliser `Python Tutor`, allez sur le site *http ://pythontutor.com/* (ajouter ce site à vos marque-pages pour y revenir plus rapidement aux prochaines séances).
- Choisissez Python 3.6 (pas Python 2.7 !)
- Cliquez sur ***"Start visualizing code"***
- Recopiez le code ci-dessous :

```
# programme pour échanger les valeurs de deux variables
print("Entrez deux reels :")
x = float(input())
y = float(input())

print("Vous avez saisi ",x," et ", y)
x = y
y = x

print("Apres echange des variables : ",x , " et ",y)
```

- Cliquez sur "Visualize Execution" puis avancez ligne par ligne dans l'exécution du programme (une zone de saisie apparaît sous la zone où vous avez tapé votre programme quand une saisie est attendue). A droite, vous pouvez voir l'affichage produit par votre programme et surtout une visualisation de ce qui se passe dans la mémoire de l'ordinateur, c'est-à-dire ici les variables existantes et leur valeur.



Appuyez sur ***Forward*** (resp. ***Back***) pour avancer (resp. reculer) d'une ligne dans le programme.

- Corrigez maintenant le programme pour qu'il fasse réellement l'échange entre les deux valeurs des variables (avec une variable temporaire dans un premier temps).

2.1.4 Exercice de lecture

Inspiré d'un exercice du partiel 2016

On considère le programme ci-dessous, censé déterminer la valeur absolue de la différence de deux entiers saisis au clavier. (Rappel : la valeur absolue de -4 est 4, celle de 7 est 7, celle de -2.5 est 2.5, etc...)

```
# Programme valeur absolue
text = input('Donner un entier : ')
x = int(text)
text = input('Donner un entier : ')
y = int(text)
z = x - y
if (z < 0)
    resultat = -z
else :
    z = resultat
    print 'valeur absolue :', resultat
```

1. Trouver les erreurs présentes dans ce programme, et corriger chaque ligne fausse
2. Que se passe-t-il si vous enlevez les `int(...)` autour des `text` ?
3. Écrire ce programme (corrigé) sous Python Tutor et observer son exécution.

2.1.5 Quelques exercices

On travaille à nouveau sous `Idle`.

Un peu d'arithmétique

1. Copier et faire exécuter ce programme (le sauver dans un fichier) :

```
print("Calcul de la somme de 2 nombres entiers")
a = int(input("Donner un nombre : "))
b = int(input("Donner un autre nombre : "))
c = a + b
print("\nLa somme est : ", c, "\n")
```

   Modifiez le programme précédent pour qu'il demande deux nombres, et affiche la somme, la différence et le produit de ces deux nombres.

2. Écrire un programme qui demande deux nombres entiers et affiche un message pour dire si les nombres sont égaux ou différents :

```
(si l'on a tapé 3 et 5)
les deux nombres sont differents

(ou bien, si l'on a tapé 4 et 4)
les deux nombres sont egaux
```

3. Écrire un programme qui demande deux nombres, et affiche le quotient entier du premier par le deuxième lorsque le deuxième n'est pas nul, ainsi que le reste, précédés du message :

   `Le quotient de` (ici le nombre) `par` (l'autre nombre) `est :` (le quotient)`. Le reste est` (le reste).

   Lorsque le deuxième nombre est nul, le programme doit afficher le texte : `Erreur, division par zéro impossible.`
   (Rappel : comme $14 = 4 \times 3 + 2$, alors le quotient de 14 par 4 est 3, et le reste est 2.)

Jeux de dés

1. Écrire un programme `DeuxDes.py` qui lit deux entiers, qui vérifie que ces entiers sont compris entre 1 et 6 (valeurs d'un dé à jouer), et qui, dans le cas de valeurs correctes, affiche les valeurs des deux dés dans l'ordre décroissant (le plus grand d'abord).

   Voici un exemple d'affichage du programme :

```
Donner la valeur du premier dé : 4
La valeur du premier de est correcte
Donner la valeur du deuxieme de : 6
La valeur du deuxieme de est correcte
Les des classes en ordre decroissant sont : 6 4
```

2. Écrire un programme `TroisDes.py` :

   - qui lit trois entiers
   - qui vérifie que ceux-ci sont compris entre 1 et 6
   - qui dans le cas de valeurs correctes, affiche les trois dés dans l'ordre décroissant
   - qui, en plus, dans le cas des trois valeurs 4, 2 et 1, affiche "Gagne!", et dans les autres cas corrects, affiche "Perdu!".
   - (dans le cas de valeurs incorrectes le programme ne fait rien)

Calcul de la moyenne et de la mention d'un étudiant

1. Écrire un programme qui calcule la moyenne d'un module qui comporte quatre disciplines :

   - Chimie
   - Physique
   - Mathématiques
   - Informatique

   Le programme demandé

   - lit les quatre notes de l'étudiant
   - calcule la moyenne des notes et l'affiche

   Si une note n'est pas comprise entre 0 et 20, le programme affiche "INCORRECT" au lieu d'afficher la moyenne.

2. Écrire une nouvelle version de ce programme mais qui, dès qu'il lit une note incorrecte, affiche "INCORRECT" et ne demande pas les notes suivantes.

3. Écrire un programme qui calcule la moyenne pondérée d'un module comportant quatre disciplines avec les coefficients suivants :

Chimie	3
Physique	4
Mathématiques	2
Informatique	2

   Le programme demandé lit les quatre notes de l'étudiant, calcule la moyenne pondérée des notes et l'affiche. Si une note n'est pas comprise entre 0 et 20, le programme affiche "INCORRECT".

4. Écrire un programme qui calcule la moyenne pondérée d'un module comme précédemment et affiche en plus la mention de l'étudiant. Les mentions sont :

   $$\begin{array}{rcccl} & \text{moyenne} & < 10 & : & \text{AJOURNE} \\ 10 \leq & \text{moyenne} & < 12 & : & \text{PASSABLE} \\ 12 \leq & \text{moyenne} & < 14 & : & \text{ASSEZ BIEN} \\ 14 \leq & \text{moyenne} & < 16 & : & \text{BIEN} \\ 16 \leq & \text{moyenne} & & : & \text{TRES BIEN} \end{array}$$

5. Modifier ce programme pour qu'il donne une 2e chance à l'utilisateur quand il saisit une note incorrecte.

Bonus si vous connaissez déjà les boucles while

1. Écrire une nouvelle version de ce programme qui filtre chaque note entrée jusqu'à ce qu'elle soit bien comprise entre 0 et 20

2. Écrire une nouvelle version de ce programme qui :

   - Demande à l'utilisateur combien de disciplines compte le module
   - Pour chacune de ces disciplines, lui demande sa note puis le coefficient de cette discipline
   - Calcule et affiche la moyenne pondérée du module
   - Affiche la mention correspondante

2.2 TP2 : prix du gros lot

Notions pratiquées : importation de modules, nombres aléatoires, boucles while

2.2.1 Les modules Python

Beaucoup de problèmes en informatique ont déjà été résolus. Par exemple, comment calculer le logarithme d'un nombre réel, ou, comment établir une connexion internet avec un ordinateur distant ? Un programmeur qui souhaite résoudre un nouveau problème qui utilise ces fonctionnalités ne devrait pas avoir à tout reprogrammer. C'est pourquoi, comme beaucoup d'autres langages de programmation, Python regroupe les fonctions couramment utilisées dans ce que l'on appelle des *modules*.

Un module est un fichier Python qui regroupe des variables, des classes et des fonctions, permettant de résoudre des problèmes spécifiques à thème donné. Par exemple, le module 'math' contient des variables comme π, et des fonctions telles que le cosinus, le sinus, la racine carrée, etc. Le module 'email' contient des fonctions permettant de créer, d'envoyer et de recevoir des emails. Comparée à d'autres langages de programmation tels que C, Python a une bibliothèque de modules très riche. Dans ce TP, nous allons apprendre à utiliser deux de ces modules : `math` et `random`.

Le module math

Ouvrez IDLE et tapez la commande suivante :

```
>>> import math
```

Cette directive permet d'accéder à toutes les fonctions du module 'math'. Essayez d'exécuter les instructions suivantes et vérifiez qu'elles donnent le résultat attendu :

```
>>> math.pi
...
>>> math.cos(0)
...
>>> math.sin(math.pi / 2)
...
>>> math.tan(math.pi / 4)
...
>>> math.tan(math.pi / 3)
...
>>> math.sqrt(16)
...
>>> math.log(10)
...
>>> math.exp(math.log(89))
...
```

Pour accéder au contenu du module math (fonctions et constantes telles que π), on a donc besoin de rajouter '*math.*' avant la fonction ou la constante à chaque fois. Il est toutefois possible d'importer directement les fonctions qui nous intéressent sans passer par le nom du module :

```
>>> from math import cos, pi, exp, log
>>> cos(pi)
...
>>> x = exp(log(2))
>>> print(x)
...
```

Dans ce cas, seuls 'cos', 'pi', 'exp' et 'log' sont importés, et peuvent être référencés directement. On peut également importer tout le contenu du module en utilisant le caractère spécial '*', qui veut dire 'tout'.

```
>>> from math import *
>>> x = log(2) + sqrt(exp(1)) * tan(pi/8)
>>> print(x)
...
```

Cependant, on évitera cette méthode d'importation car certains modules peuvent contenir des noms identiques aux noms des variables de notre programme, ce qui risque de créer des conflits.

1) Écrire un programme qui demande à l'utilisateur 2 nombres h et a, correspondant à l'hypoténuse et à l'un des deux angles non-droits (en radians) d'un triangle rectangle. Le programme doit afficher la longueur des deux côtés restants du triangle.
Indice : La somme des angles d'un triangle est égale à π.
Rappels de trigo : $cos(\alpha) = \frac{adjacent}{hypothenuse}$, $sin(\alpha) = \frac{oppose}{hypothenuse}$, et $tan(\alpha) = \frac{oppose}{adjacent}$
Exemple d'exécution :

```
Entrez la longueur de l'hypothénuse: 2
Entrez la valeur d'un angle: 1.0472
Les deux côtés sont:  0.9999957585450914 ,  1.7320532563670865
```

1b) Après calcul des autres côtés, vérifier que les valeurs trouvées vérifient bien le théorème de Pythagore.

2) Écrire un script qui demande à l'utilisateur 3 nombres : a, b et c en boucle jusqu'à ce que la valeur $delta = b^2 - 4ac$ soit supérieure ou égale à 0. Le programme affichera alors les racines du polynôme $ax^2 + bx + c$.

Le module random

Certains types de programme nécessitent l'introduction d'un comportement aléatoire. Par exemple, dans le cas d'un jeu, on veut que le joueur ait une expérience différente à chaque exécution du programme. Le module `random` de Python permet de générer des nombres aléatoires (cf cours **??**). En particulier, on s'intéressera à la fonction `randint` de ce module.

1. Importer le module `random`.
2. Exécuter les instructions suivantes plusieurs fois de suite, qu'observez-vous ?

```
>>> x = random.randint(1, 4)
>>> print(x)
```

3. Écrire un script qui lit 2 entiers a et b, vérifie que $a < b$, et affiche un nombre aléatoire compris entre a et b.

2.2.2 Prix du gros lot

Vous allez à présent développer un petit jeu dont les règles sont les suivantes :

- Le programme tire un nombre au hasard compris entre 0 et 100 inclus mais ne l'affiche **pas** à l'écran.
- Le joueur doit deviner la valeur de ce nombre en saisissant sa proposition au clavier.
- Le programme affiche selon le cas :
  - "Gagné" si le joueur a deviné le bon nombre. Le programme se termine aussitôt.
  - "Trop grand" si le joueur a saisi un nombre plus grand que le bon prix.
  - "Trop petit" si le joueur a saisi un nombre plus petit que le bon prix.

Le jeu de base

1. Écrire le programme décrit ci-dessus. Le joueur peut continuer à essayer jusqu'à ce qu'il devine le bon nombre.
2. Modifier pour afficher les nouvelles bornes (inférieures et supérieures) à chaque essai raté. Par exemple si le nombre secret est 77 :

```
Devine mon nombre entre 0 et 100
50
Trop petit
Devine mon nombre entre 51 et 100
80
Trop grand
Devine mon nombre entre 51 et 79
```

3. Modifier le programme pour que le joueur n'ait que 5 essais avant que le programme ne se termine en affichant le message 'Perdu !'. Si le joueur gagne après X essais, le programme doit afficher "Gagné au bout de X essais !".

4. Modifier pour afficher le nombre d'essais restants à chaque tentative.

```
Devine mon nombre entre 0 et 100 (5 essais)
50
Trop petit
Devine mon nombre entre 51 et 100 (4 essais)
```

5. Modifier le programme pour qu'à la fin de chaque partie, le message "Voulez-vous recommencer ? (o/n)" soit affiché. Si l'utilisateur tape 'o', une nouvelle partie commence, s'il tape 'n', le programme se termine en affichant :
   - Combien de parties le joueur a jouées et combien sont gagnées (format : 'tu as gagné 2 parties sur 5 jouées')
   - Combien d'essais il a utilisé en moyenne (sur l'ensemble des parties **gagnées**, uniquement s'il y en a au moins une) pour trouver la bonne réponse (format : 'tu as mis en moyenne 3.7 essais pour deviner')

Une IA pour le prix du gros lot

On veut maintenant écrire un programme qui sait deviner votre nombre secret. Cette fois les rôles sont inversés, vous choisissez un nombre au hasard et c'est à votre programme de le deviner. Vous devez lui répondre 'g' si sa proposition est trop grande, 'p' si sa proposition est trop petite, et 'b' s'il a deviné le bon nombre. Le nombre d'essais n'est pas limité.

1. Écrire une première version "IA aléatoire" qui affiche chaque proposition de l'ordinateur (tirée au hasard entre les bornes inf et sup) et attend la réponse ('p', 'g', ou 'b') de l'utilisateur. Le programme met à jour la borne inférieure ou supérieure en fonction de l'indice répondu par l'utilisateur, et les affiche aussi. Par exemple si vous avez choisi 88 :

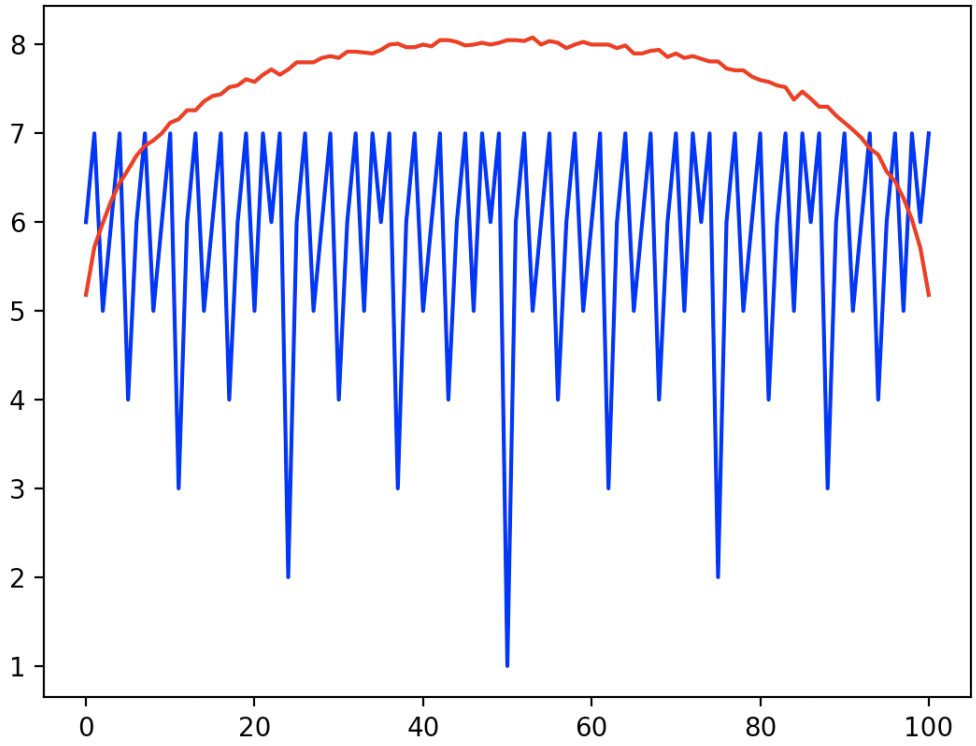
```
Je cherche un nombre entre 0 et 100
Je propose 50 ? p
Je cherche un nombre entre 51 et 100
Je propose 93 ? g
Je cherche un nombre entre 51 et 92
Je propose 88 ? b
J'ai gagné
```

2. Automatiser le rôle du "maître du jeu". Ce n'est plus l'utilisateur qui choisit un nombre secret, mais le programme qui tire un nombre secret aléatoire, puis essaye de le deviner. Cette fois l'IA n'attend plus la réponse de l'utilisateur, c'est le programme qui vérifie si sa proposition est correcte. Le programme affiche chaque essai au fur et à mesure, l'indice correspondant ('p' ou 'g'), et le nombre d'essais à la fin.
3. Écrire un nouveau programme qui fait jouer n parties (n sera demandé à l'utilisateur) à l'ordinateur (avec un nouveau nombre secret tiré aléatoirement à chaque partie), et affiche son nombre moyen d'essais par partie gagnée à la fin (comme on n'a pas limité le nombre d'essais, le programme finit forcément par gagner).
4. Il y a plusieurs manières de programmer l'IA. Trouver un algorithme plus efficace que l'essai aléatoire et programmer cette "IA intelligente". *Indice :* inspirez vous de votre propre raisonnement quand vous essayez de deviner un nombre.
5. Faire jouer n parties aux 2 IA et comparer leur nombre moyen d'essais. Que se passe-t-il si n est faible ? Si n est grand ? On peut remarquer que l'IA dichotomique a toujours le même comportement, il n'est donc pas nécessaire de la faire répéter ; par contre il faut répéter l'IA aléatoire pour "lisser" son comportement.
6. On remarquera qu'en moyenne l'IA "intelligente" est meilleure que l'IA "aléatoire". Mais certains nombres sont plus faciles à deviner que d'autres avec la méthode "dichotomique". Écrire un nouveau programme qui fait jouer n parties à chaque IA sur chacun des nombres secrets possibles (entre 0 et 100 inclus). On affichera pour chaque nombre secret la moyenne d'essais des 2 IA.
7. Afficher les résultats sous forme d'un histogramme textuel : pour chaque nombre secret (entre 0 et 100) afficher deux lignes de caractères dont la longueur dépend du nombre moyen d'essais, une ligne par IA. Par exemple utiliser le caractère '*' pour l'IA intelligente, et le caractère '-' pour l'IA aléatoire. Que remarque-t-on ? Par exemple

```
[0] ****
[0] ------
[1] ****
[1] -------
etc
```

Bonus : affichage graphique

La librairie `matplotlib` permet de créer des graphiques [1]. Utiliser cette librairie pour afficher sur un même graphique les courbes du nombre moyen d'essais de chaque IA, pour toutes les valeurs entre 0 et 100. Penser à afficher les 2 courbes avec des couleurs différentes et à ajouter une légende. Par exemple voici le résultat pour 10000 parties. Expliquer ce résultat.



Bonus : stratégies de jeu

Imaginer d'autres algorithmes qui jouent à ce jeu avec d'autres méthodes, puis calculer et afficher leurs statistiques sur le même graphique pour les comparer.

1. Voir par exemple la page `http://apprendre-python.com/page-creer-graphiques-scientifiques-python-apprendre`

2.3 TP3 : initiation aux fonctions avec `turtle`

Notions pratiquées : dessin avec le module `turtle`, boucles `while` imbriquées, initiation aux fonctions.

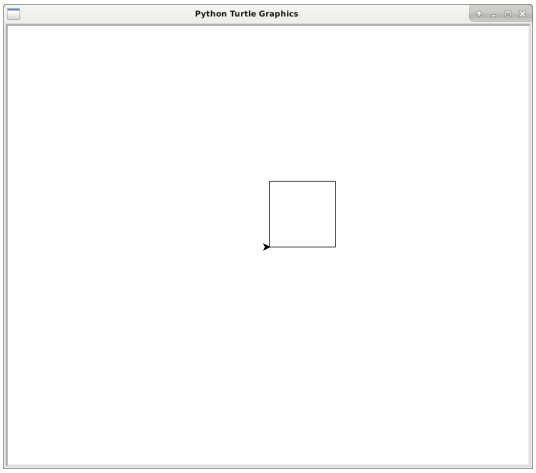
2.3.1 Marchons au pas de `turtle`

Rappel : pour utiliser les fonctions d'un module, il faut utiliser le mot clé `import` suivi du nom du module.

Le module `turtle` est un module graphique, inspiré de la programmation `Logo`,[2] qui permet de déplacer une tortue sur l'écran. Quand la tortue se déplace et que le crayon est baissé, sa trajectoire se trace à l'écran. Nous allons l'utiliser pour dessiner des figures.

Quand on utilise une fonction du module `turtle`, cela ouvre une fenêtre graphique, et le curseur est initialement placé au point de coordonnées (0,0) qui est au milieu de la fenêtre. L'orientation des axes (abscisses et ordonnées) est identique à celle conventionnellement utilisée en mathématiques. Sur la figure ci-dessous, le coin inférieur gauche du carré a pour coordonnées (0,0) et son coin supérieur droit (100,100). La taille de la fenêtre peut varier d'un ordinateur à l'autre mais elle peut être changée en utilisant la fonction `setup(x,y)`.

Voici une liste non exhaustive[3] des fonctions disponibles dans cette bibliothèque :



- `reset()` : efface l'écran, recentre la tortue et remet les variables à zéro
- `forward(distance)` : avance d'une distance donnée (en pixels)
- `backward(distance)` : recule d'une distance donnée (en pixels)
- `left(angle)` : pivote vers la gauche d'un angle donné en degrés
- `right(angle)` : pivote vers la droite d'un angle donné en degrés
- `up()` : relève le crayon (pour pouvoir avancer sans dessiner)
- `down()` : abaisse le crayon (pour recommencer à dessiner)
- `goto(x, y)` : va à l'endroit de coordonnées `(x, y)` (attention à ne pas oublier d'utiliser la fonction `up()` avant d'utiliser `goto(x,y)` car sinon il tracera le parcours effectué
- `circle(r)` : trace un cercle de rayon `r`
- `setup(x,y)` : pour fixer les dimensions de la fenêtre graphique : `x` est la largeur et `y` la hauteur.
- `color(str)` : change la couleur du dessin
- `speed(int)` : change la vitesse de déplacement de la tortue

Nous allons travailler sous `Idle`. Pour chaque exercice, vous écrirez votre programme dans un nouveau fichier.
ATTENTION ! ne nommez **jamais** vos propres fichiers `turtle.py`, sinon c'est votre propre fichier qui est importé au lieu du module turtle...

Carré

Écrivez un programme permettant de tracer un carré de côté 100. A la fin du tracé, le curseur doit être au même endroit et dans la même direction qu'au début du tracé.

Si vous n'avez pas utilisé de boucle, modifiez votre programme pour utiliser une boucle `while`. Vous ne devez utiliser qu'une fois la fonction `turtle.forward(...)`.

Carré bis

Modifiez le programme précédent pour dessiner un carré identique au précédent dont le coin supérieur gauche a pour coordonnées le point (-50,20). A la fin du tracé, le curseur doit être au même endroit (-50,20) et dans la même direction qu'au début du tracé.

Attention : à l'exécution de votre programme, seul le carré doit être tracé ! Si le point (-50,20) n'est pas le coin supérieur gauche mais plutôt le coin inférieur gauche de votre carré, essayez de tourner dans l'autre sens.

2. Si vous voulez retrouver un dessin plus proche de l'original de la tortue Logo vous pouvez utiliser `shape('turtle')` pour changer le curseur en tortue.

3. Il existe d'autres fonctions. Pour connaître la liste de toutes les fonctions disponibles dans le module `turtle`, ainsi que comment les utiliser, visitez la page de la documentation en ligne du module : `https://docs.python.org/3/library/turtle.html`

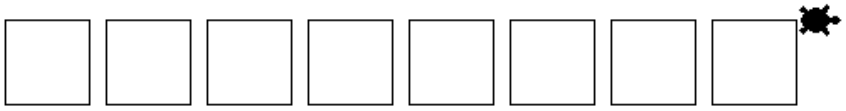
Triangle équilatéral

Écrivez un programme permettant de tracer un triangle équilatéral de côté 100 avec la pointe dirigée vers le bas. A la fin du tracé, le curseur doit être au même endroit et dans la même direction qu'au début du tracé.

Si vous n'avez pas utilisé de boucle, modifiez votre programme pour utiliser une boucle `while`*. Vous ne devez utiliser qu'une fois la fonction* `turtle.forward(...)`*.*

Ligne de carrés

On souhaite maintenant obtenir une ligne de 8 carrés comme sur la figure suivante. Les carrés ont une dimension de 50. La distance entre deux carrés successifs est de 10. Pour l'instant, on ne se soucie pas vraiment de la position du curseur au début ou à la fin du tracé. *Vous devez avoir deux boucles while imbriquées, et seuls les carrés doivent être tracés, sans être reliés par des traits.*



Modifiez votre programme pour que le coin supérieur gauche du premier carré (le plus à gauche) ait pour coordonnées (-200,200).

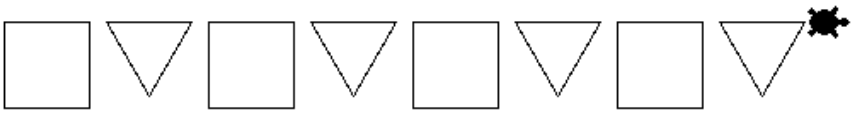
Pavage de carrés

Utiliser une 3e boucle while (imbriquées) pour dessiner un pavage : 4 lignes de 8 carrés de côté 50.

Ligne de carrés et triangles alternés

On souhaite maintenant dessiner une figure très similaire à la précédente mais en alternant carrés et triangles comme ci-dessous. Voici les caractéristiques imposées du dessin :

- on commence par un carré
- les carrés et les triangles ont tous des côtés de 50
- sur la ligne du haut, l'écart entre deux figures consécutives est toujours de 10
- le coin supérieur gauche du premier carré a pour coordonnées (-200,200)



Modifiez maintenant votre programme pour laisser l'utilisateur choisir (avec un `input()`) combien de figures il souhaite tracer en tout. Par exemple, si le tracé ci-dessus s'obtient maintenant en demandant 8 figures.

2.3.2 A la découverte des fonctions à travers le dessin

Programme mystère

Écrivez le programme de droite dans un nouveau fichier. Exécutez ce programme. Quel résultat obtient-on ? Ajoutez l'instruction `turtle.circle(115)` à la fin de votre programme et exécutez.

```
import turtle
i = 0
while i < 360 :
    turtle.forward(2)
    turtle.left(1)
    i=i+1
```

Nous avons trouvé comment tracer une figure proche d'un cercle avec le module turtle mais pour tracer des cercles, nous n'utilisons pas ce programme car la fonction `circle(r)` existe déjà dans le module `turtle` et que c'est beaucoup plus pratique à utiliser. De la même façon, dans les exercices précédents, il aurait été pratique de disposer d'une fonction pour tracer un carré ou un triangle équilatéral, en spécifiant sa taille. Justement, en programmation, il est possible de définir ses propres fonctions. Une fonction est un ensemble d'instructions regroupées sous un nom et s'exécutant à la demande. Cela a plusieurs avantages : éviter de répéter du code (code plus court, moins redondant) ; limiter les

erreurs notamment de copier/coller (une fois qu'on a écrit et testé notre fonction `carre`, on sait que cela marchera à chaque fois qu'on en aura besoin) ; permettre de réutiliser une fonction sans la réécrire ; améliorer la lisibilité d'un programme (on comprend plus vite ce que fait le programme quand on voit écrit `carre`).

Une première fonction

Vous allez maintenant écrire votre première fonction en `Python`. Recopiez la première partie du programme à droite (au dessus de la ligne horizontale).
Que se passe-t-il quand vous exécutez ce programme ?

En fait, ici, vous avez *défini* ***comment*** tracer un carré, mais vous ne demandez pas à ce qu'un carré soit tracé. Une fonction est une sorte de "sous-programme" qui donne la méthode pour effectuer une tâche, mais ensuite il faut faire **appel** à cette méthode quand on veut que la tâche soit effectuée. Pour cela, on fait suivre notre définition de fonction par un programme principal, écrit à la suite dans le même fichier. Complétez donc le programme précédent en lui ajoutant les lignes à droite, puis exécutez le programme.

```
import turtle

def carre() :
    # trace un carre de taille 100
    i = 1 # compteur du nb de cotes
    while i <= 4 :
        turtle.forward(100)
        turtle.right(90)
        i = i + 1
```

```
# programme principal, avec appel de fonction
carre()
turtle.up()
turtle.forward(130)
turtle.down()
carre()
```

En pratique, nous nous trouvons vite limités avec la fonction `carre` que nous venons de définir. Si nous voulons tracer des carres de taille 50, et que pour cela il faut modifier la fonction ou en écrire une autre, cela perd beaucoup de son intérêt. Il serait intéressant de pouvoir spécifier une taille au moment de tracer un carré, de la même façon qu'on utilise la fonction `circle` de `turtle` en spécifiant un rayon. Pour cela il faut définir un **paramètre** à la fonction.

Fonction `carre` avec paramètre

Écrivez maintenant le code de droite et complétez-le avec un programme principal de façon à appeler la fonction pour tracer plusieurs carrés de tailles différentes. Attention, pour appeler cette fonction il faudra spécifier entre les parenthèses la **valeur** qu'on souhaite donner au paramètre `cote`.

```
import turtle

def carre(cote) :
    # trace un carre de taille determinee : cote
    i = 1 # compteur du nb de cotes
    while i <= 4 :
        turtle.forward(cote)
        turtle.right(90)
        i = i + 1
```

En fait, une fonction peut avoir plusieurs paramètres, ainsi, on aurait pu définir une fonction `carre(x,y,cote)` où `(x,y)` désigneraient les coordonnées du coin supérieur gauche du carré et `cote` sa taille. Dans ce cas, l'appel se fera avec 3 nombres entre parenthèses, le premier correspondant au `x` souhaité, le deuxième au `y` et le troisième au `cote`. Ainsi, l'instruction `carre(-50, 20, 100)` permettrait de tracer le carré demandé dans l'exercice *Carré bis*. Essayez de faire cette modification. Pour la suite, vous choisirez la version de la fonction paramétrée qui vous est la plus pratique.

Fonction pour tracer un triangle équilatéral

En vous inspirant de la fonction pour tracer un carré, écrivez une fonction `triangle(cote)` qui trace un triangle équilatéral de côté `cote` avec la pointe orientée vers le bas. Par la suite, vous pourrez modifier cette fonction pour y ajouter des paramètres `(x,y)` correspondant aux coordonnées du sommet en haut à gauche.

Ligne de carrés et triangles avec fonctions

Dans un même fichier, vous allez définir la fonction `carre`, la fonction `triangle` et les utiliser dans un programme principal pour dessiner la même ligne de carrés et triangles alternés qu'à l'exercice 5 (version avec 8 carrés). Voici à droite la structure de l'ensemble.

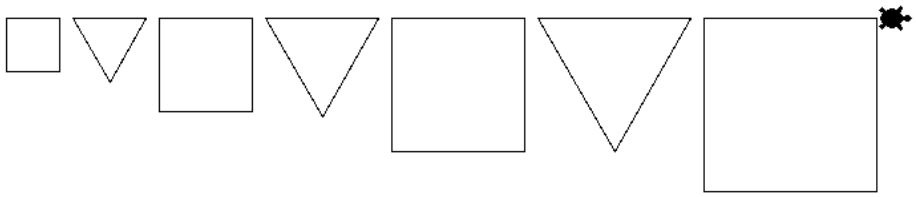
```
import turtle

def carre(cote) :
    ...

def triangle(cote) :
    ...

# programme principal
# trace ligne carres et triangles
...
```

2.3.3 Pour aller plus loin

Si vous avez fini l'ensemble de ce qui précède, vous pouvez essayer de réaliser les figures des exercices suivants.

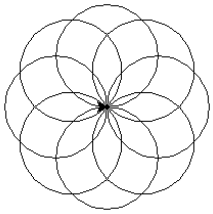
Ligne de carrés et triangles alternés de tailles croissantes

Tracer une ligne de carrés et triangles alternés pour laquelle le premier carré a un côté de 40. La taille augmente de 15 à chaque nouvelle figure tracée. L'espacement entre les figures reste le même.

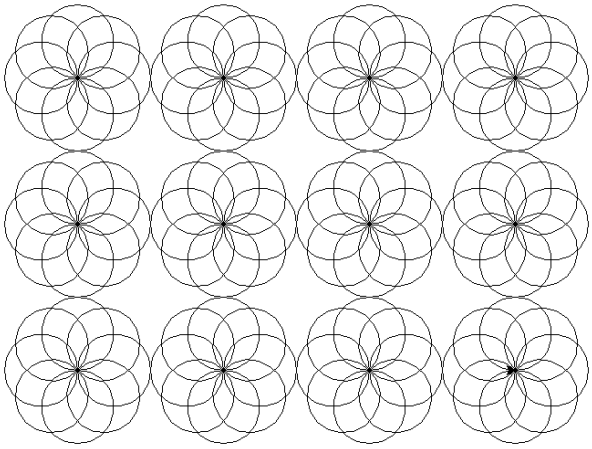


Rosace et pavage de rosaces

1. Définir une fonction `rosace(x, y, nb, r)` qui trace une rosace centrée sur le point de coordonnées `(x,y)`, composée de `nb` cercles chacun de rayon `r`. Attention : le centre de la rosace est l'endroit où on voit le curseur sur la figure ci-dessous pour une rosace à 8 cercles :



2. Utilisez cette fonction pour écrire une autre fonction qui trace une ligne de `tl` rosaces (paramètre).
3. Utilisez cette fonction (ligne) pour écrire une fonction qui trace un pavage de rosaces avec `tp` lignes (paramètre), comme sur la figure ci-dessous (pour `nb=8`, `tl=4` et `tp=3`) :



Si vous n'arrivez pas à définir la fonction, vous pouvez quand même essayer de faire un pavage de rosaces sans utiliser de fonction.

Couleurs

Modifier les fonctions précédentes pour choisir aléatoirement une nouvelle couleur pour chaque rosace.

Bonus : labyrinthe

Créer une fonction qui trace un gros rectangle noir de la taille de la fenêtre, puis fait se déplacer la tortue au hasard en traçant un chemin avec des rectangles blancs pour former un labyrinthe.

2.4 TP4 : création et manipulation de fonctions basiques

Notions pratiquées : fonctions, bases des chaînes de caractères (longueur, opérateurs de base).
Des fichiers sont à télécharger sur CaseIne : `http://caseine.org/mod/resource/view.php?id=3296`

2.4.1 Exercices d'observation

Décrire les appels de fonctions

Tapez et lancez les programmes suivants dans Python Tutor, et comptez combien de fois chacune des fonctions est appelée. Pour chaque appel, décrire quelles sont les valeurs des arguments et la valeur de retour. (Pensez à sauver vos programmes dans un fichier texte avant de les copier-coller dans Python Tutor.)

```
# Programme 1
def est_solution(x,a,b,c):
    # Renvoie True si x est solution de
    # ax^2+bx+c=0
    y=a*x**2+b*x+c
    if y==0:
        rep=True
    else:
        rep=False
    return rep

# prog. principal
rep=est_solution(1,1,-2,1)
print(rep)
print(est_solution(5, 2, -20, 50))
print(est_solution(2.5, 1, 2, 3))
```

```
# Programme 2
def distance(xA,yA,xB,yB):
    # renvoie la distance entre (xA,yA) et (xB,yB)
    d=(xB-xA)**2+(yB-yA)**2
    d=d**(1/2)
    return d

def appartient_cercle(xA,yA,rayon):
    # teste si le point (xA,yA) appartient
    # au cercle de rayon r centré à l'origine
    if distance(0,0,xA,yA)==rayon:
        return True
    else:
        return False

# prog. principal
d=distance(1,2,2,1)
print(d)
rep=appartient_cercle(1,1,2)
print(rep)
print(appartient_cercle(1,0,1))
```

Reconnaître les variables locales

Tapez le programme suivant dans Python Tutor et observez son fonctionnement. Quelles sont les variables locales de la fonction `est_premier` ?

```
def est_premier(N):
    i=2
    a_diviseur=False
    while i<N and (not a_diviseur):
        if N%i==0:
            a_diviseur=True
        i=i+1
    return not a_diviseur

# prog. principal
if __name__=="__main__":
    rep=est_premier(9)
    print("9 est premier ?", rep)
    print("5 est premier ?", est_premier(5))
```

`locals()`

Testez le programme suivant dans IDLE et observez attentivement ce qui s'affiche. Que renvoie l'appel à la fonction `locals()` ?

```
def pente(xA, yA, xB, yB):
    p=(yB-yA)/(xB-xA)
    print("Variables locales de pente :", locals())
    return p

# prog. principal
p1=pente(1,1,2,2)
print("Pente 1: ", p1)
p2=pente(0,2,1,4.5)
print("Pente 2: ", p2)
```

`locals()` appelée dans une fonction renvoie un dictionnaire (voir plus tard le cours sur les dictionnaires) contenant les variables locales de cette fonction et leurs valeurs (si définies avant l'appel à `locals()`. Python Tutor permet aussi d'observer les variables locales d'une fonction et leurs valeurs, qui apparaissent dans un rectangle séparé (*frame*) des variables du programme principal (*global frame*).

Incrémentation

Prédire le fonctionnement des deux programmes suivants, et vérifier votre réponse en les lançant dans Python Tutor pas à pas.

```
# Programme 1
def incremente (a):
    a = a+1

# prog. principal
b=3
incremente(b)
print (b)
a=5
incremente(a)
print (a)
```

```
# Programme 2
def incremente2(a):
    return a+1

# prog. principal
b=1
b = incremente2(b)
print(b)
a=incremente2(b)
print(a)
```

Maintenant, rajoutez l'instruction `a=incremente(a)` à la fin du programme 1 et re-testez dans Python Tutor. Que vaut `a` à la fin du programme ? Traduisez en français et cherchez une explication.

2.4.2 Des petites fonctions

On se place maintenant dans Idle, dans un nouveau fichier. Vous prendrez bien soin de tester vos fonctions au fur et à mesure, soit en faisant des appels dans votre programme principal, soit directement dans l'interpréteur après avoir fait "Run Module" (testez chacune des deux solutions au moins une fois).

La bosse des maths

1. Écrire une fonction `valeur_abs` qui prend en argument un nombre x et qui renvoie sa valeur absolue.
2. Écrire une fonction `signe_different` qui prend en arguments deux nombres x et y et qui renvoie `True` si x et y ont des signes différents, `False` sinon (si x ou y est égal à 0, la fonction doit renvoyer `False`).
3. Soit f la fonction mathématique définie sur $\mathbb{R}$ par $f(x) = 3x^2 + 2x + 3$. Écrire une fonction Python `f` qui prend en argument un réel x et qui renvoie la valeur de $f(x)$.
4. Écrire une fonction `nb_racines` qui prend en argument trois réels a, b, c et qui renvoie le nombre de racines réelles du polynôme $ax^2 + bx + c$. On rappelle la formule du discriminant : $\Delta = b^2 - 4ac$.

Le journal de M. Bizarre

M. Bizarre est rédacteur-en-chef d'un journal, et a souvent des idées farfelues. Suivant les jours de la semaine, il souhaite que les articles de son journal suivent des règles qu'il a inventées.

1. Chaque lundi, il souhaite que les mots soient répétés deux fois, séparés par un espace. Écrivez une fonction `lundi` qui prend en argument un mot et qui transforme ce mot selon la règle du lundi. Par exemple, `lundi("bonjour")` doit valoir `"bonjour bonjour"`. *Rappel : l'opérateur + permet de concaténer des chaînes.*
2. Chaque mardi, il faut que les mots de longueur paire soit répétés 6 fois, en séparant par un tiret, alors que les mots de longueur impaire doivent être répétés 3 fois, en séparant par des virgules. Écrire la fonction `mardi` qui prend en argument un mot et qui renvoie la chaîne de caractères suivant la règle du mardi. *Rappel : la longueur d'une chaîne de caractères est donnée par `len(chaine)`.*
3. Chaque mercredi, il faut que les mots de longueur impaire soit remplacés par le mot `"impair"`. Les mots pairs doivent rester tels qu'ils sont. Écrire la fonction `mercredi` (qui prend en argument un mot et qui renvoie la bonne chaîne de caractères).
4. Chaque jeudi, il faut que les mots soient répétés autant de fois que leur longueur modulo 3 (à la suite, sans espace). Écrire la fonction `jeudi`. Par exemple, `jeudi("merci")` vaut `"mercimerci"`, `jeudi("bonbon")` vaut `""`, `jeudi("comment")` vaut `"comment"`. *Rappel : l'opérateur modulo en Python s'écrit %*
5. Le vendredi, il faut que les mots soient écrits normalement. Et heureusement, il n'y a pas de journal le week-end.
6. Écrire une fonction `transforme(mot, num_jour)` qui prend en argument un mot et le numéro du jour (1 pour lundi, 2 pour mardi, etc....) et qui renvoie le mot transformé selon la règle du jour correspondant.

2.5 TP5 : manipulation de fonctions (suite), dessin avec Turtle

Notions pratiquées : boucles `while` y compris imbriquées, suites numériques, fonctions, module `turtle`.

2.5.1 Retour aux fonctions numériques

La banque

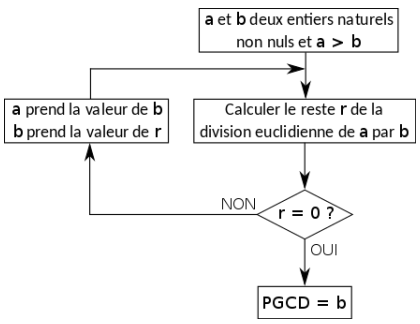
Aline envisage d'ouvrir un compte à la banque Argento, mais elle veut d'abord savoir si cela sera rentable. Sur un tel compte, les intérêts sont de 5% par an, et la banque prélève un coût fixe annuel de 11 euros. Le capital de l'année $n+1$ est donc obtenu par la formule $u_{n+1} = u_n \times 1.05 - 11$, où u_n désigne le capital à l'année n.

1. Écrire une fonction `capital(nb_annees, capital_debut)` qui renvoie le capital en euros qu'Aline aurait sur un tel compte au bout de `nb_annees` en plaçant initialement un capital égal à `capital_debut` (en euros).
2. Écrire une fonction `gagne_argent(nb_annees, capital_debut)` qui renvoie un booléen indiquant si le capital au bout de `nb_annees` sur un tel compte est (ou non) supérieur ou égal au capital de début.
3. Écrire une fonction `placement_min(nb_annees,but)` qui calcule le placement minimum nécessaire pour atteindre au moins le capital `but` après `nb_annees` d'économies.
4. Écrire une fonction `duree_min(capital,but)` qui calcule la durée minimum de placement avec un capital de départ donné pour atteindre le capital but souhaité. **Attention** : si le capital initial est trop petit, les frais annuels vont coûter plus cher qu'il ne rapporte, et il va diminuer au fil du temps, ce qui risque de causer une boucle infinie. Trouver la valeur minimale du capital nécessaire pour qu'il augmente, et gérer ce cas particulier en affichant un message d'erreur au lieu de rentrer dans la boucle.
5. Écrire un programme principal qui teste ces fonctions.

Un peu d'arithmétique

Dans cet exercice, on ne considérera que des entiers strictement positifs.

1. Écrire une fonction `plus_grand_diviseur_premier(n)` qui renvoie le plus grand diviseur premier de l'entier `n`. Il vous est conseillé de commencer par redéfinir la fonction `est_premier(N)` que l'on a déjà vue.
2. Écrire une fonction `pgcd(a,b)` qui renvoie le plus grand commun diviseur des entiers a et b. *Note :* il existe principalement deux algorithmes pour calculer un pgcd, vous pouvez essayer les deux :
   - L'algorithme "naïf" qui parcourt tous les entiers qui sont candidats à être pgcd.
   - L'algorithme d'Euclide, qui est beaucoup plus rapide : voir schéma ci-contre. (Si $a < b$, l'algorithme commence par échanger a et b puis fonctionne normalement suivant le schéma ; si $a = b$, l'algorithme renvoie simplement a).
   - Bonus : comparer le temps de calcul et le nombre de comparaisons faites par chaque algorithme. (Le module `time` fournit une fonction `time()` qui renvoie le temps système en seconde ; on peut alors l'utiliser pour ccalculer le temps d'exécution d'un programme. Le module `timeit` module[4] permet aussi de chronométrer de courts extraits de code.)
3. Écrire une fonction `ppcm(a,b)` qui renvoie le plus petit commun multiple de `a` et `b`.
4. Écrire une fonction `irreductible(numerateur, denominateur)` qui calcule un booléen indiquant si la fraction correspondante $\frac{numerateur}{denominateur}$ est irréductible ou non.
5. Difficile : écrire une fonction qui reçoit 2 entiers pgcd et ppcm, et qui affiche tous les couples d'entiers a et b dans l'ordre croissant donc ces nombres sont le pgcd et le ppcm.



4. https ://docs.python.org/2/library/timeit.html

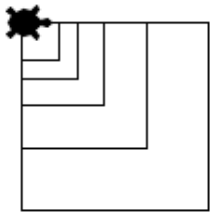
2.5.2 Turtle

Dans cette partie, nous allons utiliser la bibliothèque graphique Turtle. N'oubliez pas de l'importer. Vous testerez bien toutes vos fonctions au fur et à mesure, et vous garderez le même fichier pour tous les exercices de cette partie (les exercices sont dépendants les uns des autres, dans l'ordre).

Note : À partir de maintenant, nous utilisons l'expression "Écrire une fonction `nom_fonction(argument1,argument2, ....)`" pour dire "Écrire une fonction `nom_fonction` qui prend comme arguments `argument1`, `argument2`,....".

Carrés imbriqués

1. Écrire une fonction `carre(cote)` qui trace un carré dont la longueur du côté est `cote`, et dont le point de départ et d'arrivée est le coin supérieur gauche (comme au TP3).
2. Écrire une fonction `carres_imbriques(cote_debut, nb_carres)` qui trace des carrés imbriqués comme sur la figure ci-contre : le plus grand carré a pour côté `cote_debut`, la taille du carré est multipliée par 2/3 à chaque étape, et le nombre de carrés tracés est `nb_carres`.



`carres_imbriques(100,5)`

Se déplacer

1. Écrire une fonction `aller_sans_tracer(x,y)` qui déplace le curseur au point de coordonnées `x,y` sans tracer.

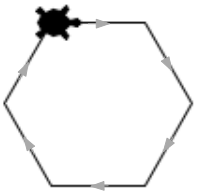
2. Écrire une fonction `descendre_sans_tracer(longueur)` qui descend de `longueur` par rapport à la position actuelle, sans tracer. On supposera qu'au moment de l'appel, le curseur est dirigé vers la droite, et on prendra soin de le remettre dans la même position à la fin.

3. Bonus : écrire une fonction qui fait se promener la tortue au hasard : distance aléatoire, angle aléatoire. Elle pourra laisser des empreintes à chaque pas avec `stamp()` ; dans ce cas on pourra changer la forme du curseur avec `turtle.shape('turtle')`, et aussi choisir une couleur aléatoire (avec `random.choice` par exemple qui peut choisir un élément d'une liste, ou bien écrivez votre propre fonction de sélection de couleur). Enfin on pourra faire en sorte pour ne pas perdre la tortue de vue, que quand elle sort de la fenêtre d'un côté, elle y re-rentre aussitôt de l'autre côté (environnement torique). Rappel : `turtle.screensize()` fournit les dimensions de la fenêtre, et `turtle.setup(x,y)` permet de les modifier.

4. Bonus (cours sur les fonctions avancées) : écrire une fonction de tracé qui reçoit la couleur en paramètre optionnel (par défaut noir).

Polygone

1. Écrire une fonction `polygone(nb_cotes,cote)` qui trace un polygone régulier à `nb_cotes` côtés (triangle, carré, pentagone, hexagone....) et dont la longueur d'un côté est `cote`. Le premier côté tracé doit être horizontal et vous devez tourner dans le sens des aiguilles d'une montre (voir figure ci-contre).
2. Écrire une fonction `diametre_polygone(nb_cotes, cote)` qui renvoie le diamètre d'un polygone régulier à `nb_cotes` de longueur `cote`. Ce diamètre est donné par la formule suivante, avec n de nombre de cotés et c la longueur du côté (n'oubliez pas d'importer le module `math`) :



`polygone(6,50)`

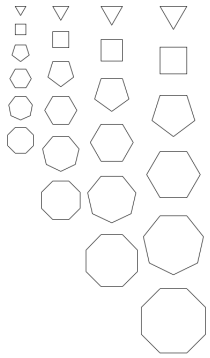
$$\frac{c}{sin\,(\pi/n)}$$

Pavage de polygones

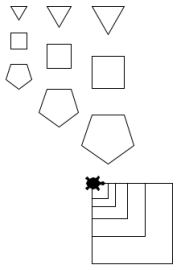
1. Écrire une fonction `colonne_polygone(nb_poly, cote)` qui trace `nb_poly` polygones sur la même colonne, en commençant par un triangle et en augmentant le nombre de côtés de 1 à chaque fois. La longueur du côté des polygones doit toujours être égale à `cote`. Vous vous servirez des fonctions `descendre_sans_tracer` et `diametre_polygone` pour décaler votre curseur de $d + 5$ vers le bas entre deux polygones successifs, où d désigne le diamètre du polygone qui vient d'être tracé.

2. Écrire une fonction `pavage(nb_poly, nb_col, cote)` qui dessine un pavage de polygones comme sur la figure ci-contre, en commençant au point de coordonnées (-270, 330). Le nombre de polygones par colonne doit être réglé par `nb_poly` et le nombre de colonnes par `nb_col`. De plus, le côté des polygones doit être de `cote` pour la première colonne puis augmenter de 10 lorsque l'on change de colonne. Les points de départ de chaque colonne doivent avoir la même ordonnée (les triangles du haut sont alignés), et leurs abscisses doivent être écartées de $d + 10$ (vers la droite), où d est le diamètre du dernier polygone tracé. Pensez à utiliser les fonctions créées précédemment. *Si vous avez du mal à écrire cette fonction, vous pouvez commencer à écrire une fonction* `pavage5()` *qui ne prend aucun argument, et qui dessine un pavage avec 5 lignes et 5 colonnes, en commençant avec un côté égal à 20.*

3. Écrire un programme principal qui dessine la figure ci-contre, contenant un pavage à 3 colonnes avec 3 polygones par colonne, commençant avec un coté de 20 ; puis qui dessine 5 carrés imbriqués, dont le premier a pour longueur 100.



`pavage(6,4,20)`



`prog. principal`

2.5.3 Suites numériques

Fibonacci

Soit la suite de Fibonacci (cf TD 1.3.14) donnée par :

$$\begin{cases} f_0 = 1 \\ f_1 = 1 \\ f_n = f_{n-1} + f_{n-2} \end{cases}$$

Écrire des fonctions pour :

1. Afficher tous les termes jusqu'au n-ième
2. Renvoyer le n-ième terme sans rien afficher
3. Renvoyer la n-ième estimation du nombre d'or ($\frac{f_n}{f_{n-1}}$. Attention : ne calculez pas tous les termes 2 fois !
4. Calculer le nombre d'or avec une précision donnée, renvoyer le n nécessaire pour l'atteindre
5. Tracer la fonction $\frac{F_n}{F_{n-1}}$ et la droite NBOR avec le module `matplotlib` pour visualiser la convergence de cette estimation.

Syracuse

Soit la suite de Syracuse (cf TD 1.3.15) donnée par $u_0 = A$ et par la relation de récurrence :

$$u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair} \\ 3 * u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

Écrire des fonctions pour :

1. Calculer la durée de vol pour atteindre 1 à partir d'un A donné en paramètre
2. Demander A à l'utilisateur et afficher la durée de vol, en appelant la première fonction
3. Version modifiée qui boucle et redemande une valeur de A jusqu'à ce que l'utilisateur refuse de rejouer
4. Tester toutes les valeurs de A entre 1 et une borne B reçue en paramètre, et renvoyer celle qui a la plus longue durée de vol
5. Tester toutes les valeurs de départ A (jusqu'à une borne X reçue en paramètre) et afficher pour chacune la durée de vol associée

Bonus avec affichage graphique :

6. Afficher les durées de vol pour chaque A de départ sous la forme d'un histogramme de rectangles avec le module `turtle`. *Bonus : calculer la bonne largeur des rectangles en fonction de la borne max pour que l'histogramme contienne juste dans la fenêtre.*
7. Afficher ce même graphique mais sous la forme d'une courbe avec le module `mathplotlib`.

2.5.4 On travaille en ligne !

Quand vous aurez fini les exercices précédemment proposés, vous pouvez travailler en ligne sur le site du PLM : `https://plm.telecomnancy.univ-lorraine.fr/#/`. PLM signifie en anglais "The Programmer's Learning Machine", c'est une plateforme en ligne avec des exercices de programmation.

Pas 1 : Il faut d'abord choisir Python avant de commencer les exercices. Sur le coin haut à droite de votre écran, changez Java en Python.

Pas 2 : Cliquez sur le bouton en haut à gauche et vous aurez la liste de sujets proposés. Vous pouvez commencer par les exercices correspondant à de "boucles tant que" (while) et "fonctions".

2.6 TP6 : listes

Notions pratiquées : listes (initiation), types mutables, chaînes de caractères.

2.6.1 Consultation de listes

Dans cette partie, nous allons d'abord nous concentrer sur l'utilisation de listes existantes. Nous verrons dans la deuxième partie du TP comment créer ou modifier des listes..

Note : Les codes python présentés ici sont dans une archive téléchargeable sur Caseine : `http://caseine.org/mod/resource/view.php?id=3326`. Récupérez la et décompressez la dans votre répertoire de TP avant de commencer.

Listes sous PythonTutor

Essayez de prévoir le comportement des deux exemples suivants puis testez-les pas à pas sur `www.PythonTutor.com`. (Correspondent aux fichiers `pythontutor1.py` et `pythontutor2.py` de l'archive.)

```
maliste = [1, 3, 9, 6, 4]

print(maliste)

a = maliste[2]
b = maliste[0]
c = maliste[4]
d = maliste[-1]
e = maliste[7]
```

```
maliste = [1, 3, 9, 6, 4]

print(maliste)

i = 0
while i < len(maliste):
    element = maliste[i]
    print ("indice", i, " contient: ", element)
    i = i+1
```

Regardez également le fichier nommé `intro-listes.py`. Vous devez être en mesure de comprendre tout ce qui y est fait. Testez le sous Idle. Vous pouvez le modifier à votre guise pour vérifier votre compréhension.

```
#!/usr/bin/env python3

liste_desord = [2, 12, 7, 9, 3, 4]
liste_croiss = [2, 3, 4, 5, 7, 9, 12]

def parcours (liste):
    i = 0
    while i < len(liste):
        print ("Element a l'indice", i, ":", liste[ii])= i+1
        i = i+1

parcours(liste_desord)

def parcours_envers (liste):
    i = len(liste)-1
    while i >= 0:
        print ("Element a l'indice", i, "(envers) :", liste[i])
        i = i-1
```

```
parcours_envers(liste_desord)


def croissante (liste):
    cr = True
    i = 1
    while i < len(liste):
        cr = cr and (liste[i-1] <= liste[i])
    return cr


print (liste_desord, "est croissante:", croissant
print (liste_croiss, "est croissante:", croissant
```

Recherche dans une liste

Dans cet exercice, vous allez devoir écrire plusieurs fonctions "de base" fonctionnant sur des listes d'entiers. Essayez d'avoir un code propre définissant d'abord les fonctions, puis qui teste sur plusieurs exemples vos fonctions dans un programme principal ("main").

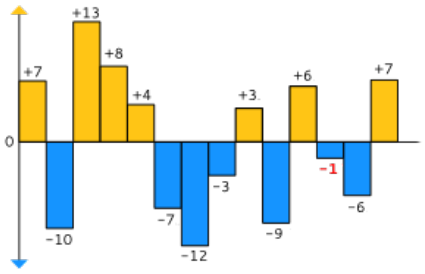
1. Écrivez une fonction `minimum(liste)` qui renvoie l'élément le plus petit de `liste` ;
2. Écrivez une fonction `contient(liste, elem)` qui renvoie True si `liste` contient `elem` et False sinon. Pensez à tester les cas limites (p.ex. élément au début ou à la fin).
3. Écrivez une fonction `minimum2(liste)` qui renvoie le deuxième élément le plus petit de `liste` (i.e., celui juste au-dessus du minimum) ;

   Suggestion : définissez une fonction auxiliaire `minimum_plus_grand_que(liste, petit)`.

Températures

Note : cet exercice est extrait du site CodinGame.

Dans cet exercice, vous devez analyser des enregistrements de température pour trouver la plus proche de zéro.



Dans l'exemple, -1 est la plus proche de 0.

Vous devez écrire une fonction `proche_zero` qui prend en argument une liste de températures (entiers) et renvoie la plus proche de zéro. S'il n'y a aucune température, renvoyer 0. En cas d'égalité entre une température positive et une négative, on considère que c'est la positive qui est la plus proche de zéro (par exemple, si les températures sont 5 et -5, il faut renvoyer 5).

Le fichier `temperature.py` contient un squelette de code prêt à tester votre fonction (note : la fonction *assert* vérifie si son argument est vrai et fait échouer le programme sinon).

Duels de chevaux de course

Note : cet exercice est également extrait du site CodinGame. Il est plus complexe que tous les autres exercices vus jusqu'à présent : il est donc nécessaire de prendre un moment pour bien comprendre le problème et réfléchir posément à comment vous allez le résoudre.

À l'hippodrome de Casablanca, on organise des courses de chevaux d'un type particulier : les duels ! Durant un duel, seuls deux chevaux participent à la course. Pour que la course soit intéressante, il faut sélectionner deux chevaux de force similaire. On considère que l'on connait la "force" des chevaux, donnée par un entier.

Écrire une fonction `plus_proches(liste)` qui, étant donnée une `liste` des forces des chevaux, identifie les forces les plus proches et renvoie la différence entre ces deux forces (un entier positif ou nul).

Le fichier `course.py` contient un squelette de code prêt à tester votre fonction. Vous êtes encouragés à écrire d'autres fonctions auxiliaires si vous le jugez nécessaire.

2.6.2 Avec modifications !

Vous allez maintenant modifier des listes, soit directement des éléments (valeurs contenues dans la liste), soit par ajout/suppression d'éléments.

Familiarisation

- Essayez vous à modifier les éléments d'une liste dans l'interpréteur Python : que fait `maliste[5] = 12` ? Que se passe-t'il si la liste n'existe pas ? Et si elle est de longueur trop petite ? etc.

- Écrivez un programme qui demande des nombres strictement positifs à l'utilisateur et les stocke dans une liste. La saisie s'arrête dès que l'utilisateur entre "0" (zéro). Affichez ensuite la moyenne de cette liste.

- Écrivez un programme qui inverse l'ordre des éléments d'une liste. Par exemple `[1, 3, 9, 6, 4]` doit devenir `[4, 6, 9, 3, 1]`.

  Note : il y a plusieurs manières de résoudre ce problème. Vous êtes libres de garder une seule liste du début à la fin, ou de créer des listes temporaires, d'ajouter, supprimer des valeurs, etc.

Les mutables : encore du PythonTutor !

Les programmes suivants permettent d'illustrer le comportement "étrange" des listes par rapport aux variables que vous avez l'habitude de manipuler. Il est essentiel à cette étape de bien comprendre la différence de comportement entre données "mutables" (notamment les listes) et "non-mutables" (entiers, flottants, chaînes de caractères...) en Python.

C'est le moment de tester ces programmes sous PythonTutor (ils sont dans l'ordre croissant de difficulté). **Observez bien les "flèches" de PythonTutor qui montrent à quels objets les variables se réfèrent.** Si vous ne comprenez pas le comportement d'un des programmes, faites appel à votre chargé·e de TP.

pythontutor3.py

```
maliste = [1, 3, 9, 6, 4]

autre = maliste

autre[2] = 42

print (maliste)
print (autre)
```

pythontutor4.py

```
def incremente(i):
    i = i+1


def incr_liste(liste):
    i = 0
    while i < len(liste):
        liste[i] = liste[i] + 1
        i = i+1


x = 12
maliste = [1, 3, 9, 6, 4]

incremente(x)
incr_liste(maliste)


print(x)
print(maliste)
```

pythontutor5.py

```
maliste = [[1], [3], [9], [6], [4]]

print(maliste)

i = 0
while i < len(maliste):
    element = maliste[i]
    print ("indice", i, " contient: ", element)
    i = i+1

print("Avant:", maliste)
maliste[3]=maliste[2]
maliste[2].append(0)
print("Apres:", maliste)
```

2.6.3 Exercices sur les fonctions et chaînes de caractères

1. Fonctions de chiffrement

Position d'une lettre Écrire une fonction qui reçoit une lettre et renvoie sa position dans l'alphabet. Par exemple la position de 'a' est 1, la position de 'E' est 5.

Chiffrement d'un mot Écrire une fonction qui reçoit un mot et renvoie ce mot codé en remplaçant chaque lettre par sa position dans l'alphabet. Les nombres doivent être séparées par des +.

Par exemple "bonjour" est codé comme "2+15+14+10+15+21+18"

Chiffrement d'un texte Écrire une fonction qui reçoit un texte et renvoie ce texte codé comme précédemment. Les espaces et autres ponctuations dans le texte ne sont pas modifiés. Attention, on s'abstiendra d'utiliser des lettres accentuées. Par exemple "bonjour a tous!" est codé en "2+15+14+10+15+21+18 1 20+15+21+19!"

Programme principal Écrire un programme principal qui opère en boucle :

- demande à l'utilisateur un texte,
- affiche ce texte codé comme ci-dessus,
- puis propose à l'utilisateur de rejouer avec un nouveau texte.

Le programme s'arrête quand l'utilisateur refuse de rejouer.

2. Fonctions de comptage et statistiques

Taille des mots Écrire une fonction qui lit un texte et renvoie une liste d'entiers contenant la taille de chacun de ses mots.

Indices : la fonction `split` *permet de séparer une chaîne de caractères (ici le texte) autour d'un séparateur donné (ici l'espace). La fonction* `list()` *convertit une chaîne de caractères en la liste de ses caractères. La fonction* `len()` *donne la taille d'une liste.*

Mot le plus long Écrire une fonction qui lit un texte et renvoie son mot le plus long.

Mots plus longs que la moyenne Écrire une fonction qui lit un texte et affiche tous les mots qui sont plus longs que la longueur moyenne de ses mots.

Indices : le module `statistics` *fournit une fonction* `mean` *qui fait la moyenne des éléments d'une liste.*

Mot comptant le plus de voyelles

- Écrire une fonction qui reçoit un mot (sans espaces) et qui renvoie le nombre de voyelles dans ce mot.
- Utiliser cette fonction pour en écrire une autre qui reçoit un texte et renvoie le mot de ce texte qui compte le plus de voyelles.

ça se complique...

Lettre la plus fréquente Écrire une fonction qui reçoit un texte et renvoie la lettre qui est présente le plus souvent dans ce texte. Attention à ne parcourir le texte **qu'une seule fois** pour compter les lettres !

2.7 TP7 : retour sur les listes

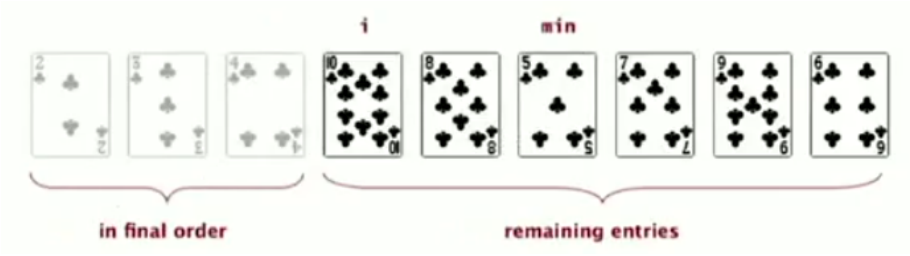
Notions pratiquées : listes, algorithmes de tri, module pyplot

2.7.1 Recyclage et Tri sélectif

a. Tri insertion. Écrivez une fonction `tri_insertion` qui trie une liste d'entiers par insertion successive dans une liste triée. C'est le principe en général appliqué lorsqu'on trie sa "main" dans un jeu de cartes [5].

Source : http://staff.ustc.edu.cn/

b. Tri sélection. Un autre façon possible de trier une liste est de prendre le minimum et de le mettre au début, puis de recommencer avec le deuxième plus petit, etc. Écrivez une fonction `tri_selection` qui trie une liste d'entiers par sélections successives du plus petit élément de la partie non triée de la liste.



Source : http://x-wei.github.io/

2.7.2 Le retour des courses (difficile)

Reprenez le même exercice que pour les duels de chevaux de courses (TP précédent). Cette fois, vous devez apparier les n chevaux existants pour former $n/2$ courses intéressantes.

1. Faites une fonction qui étant donnée une liste des forces de chevaux, renvoie une liste des couples telle que la différence des forces entre chaque couple soit la plus petite possible. *Indices : Encore une fois, cet exercice nécessite de bien comprendre et modéliser le problème avant de se lancer dans l'écriture du code. Il est très difficile de le résoudre d'un seul bloc, il est donc fortement recommandé de "découper" le problème en sous-tâches plus simple, et de faire autant de fonctions auxiliaires que nécessaire.*
2. (Vraiment difficile) Avez-vous vu que l'ordre dans lequel les paires étaient faites était important ? On peut parfois avoir des courses peu intéressantes dans un ordre alors qu'un autre serait meilleur. Par exemple, pour les 4 chevaux [12, 13, 11, 15], si on commence par faire (12,13), il nous reste (11,15) qui n'est pas très intéressante. Alors que le mieux aurait été (11, 12) et (13, 15). Avez-vous des idées pour améliorer votre algorithme ?

2.7.3 Course à pied

Initialisation. Écrire une fonction qui crée une liste de 365 éléments correspondant à la distance courue chaque jour de l'année passée par un coureur. Les éléments sont initialisés au hasard entre 0 et 100, avec 2 chiffres après la virgule.
Indices : le module random fournit une fonction uniform(a,b) qui tire un nombre réel au hasard entre les bornes a et b incluses. La fonction round(nb,x) arrondit le nombre nb avec x chiffres après la virgule.
Premier test : écrire un programme principal qui appelle cette fonction et affiche la liste obtenue.

Moyenne par jour de semaine

- Écrire une fonction qui reçoit une liste de distances, et un numéro de jour de la semaine (1 pour lundi, 7 pour dimanche). Cette fonction calcule alors la moyenne de kilomètres effectués ce jour de la semaine sur toute l'année. *(Par exemple si l'entier reçu est 2, il faut calculer la moyenne des distances de tous les mardis de l'année.)* **Remarque** : pour simplifier, on considère que l'année commence un lundi.
- Écrire ensuite une fonction qui reçoit la liste de distances, calcule la moyenne pour chacun des 7 jours de la semaine, et renvoie une liste de 7 valeurs moyennes.

5. Voir une démo : `https://www.youtube.com/watch?v=3QwCnoa_6FY`

Histogramme turtle. Écrire une fonction qui utilise le module `turtle` pour tracer un histogramme des distances moyennes des 7 jours de la semaine. N'oubliez pas de tester vos fonctions au fur et à mesure en les appelant dans votre programme principal.

2.7.4 Graphiques scientifiques

Le module matplotlib et pyplot

Le module matplotlib.pyplot permet de tracer des courbes facilement. Pour l'utiliser on ajoutera au début du fichier la ligne suivante : `import matplotlib.pyplot as plt`
On pourra ensuite appeler les fonctions de ce module en les préfixant par plt. Par exemple :

- la fonction plt.plot() reçoit la liste des valeurs de x, puis la liste des valeurs de f(x), et trace la courbe de la fonction f.
- la fonction plt.show() affiche le graphique, il faut l'appeler pour voir la fenêtre graphique
- les fonctions plt.xlabel et plt.ylabel permettent d'ajouter un nom aux 2 axes
- la fonction plt.title permet de changer le titre de la fenêtre
- la fonction plt.axis() permet de spécifier les bornes des 2 axes

Vous pouvez trouver la liste des fonctions de pyplot sur `https://matplotlib.org/api/pyplot_summary.html`

Graphique de distance

Utiliser ce module et les fonctions de l'exercice précédent pour afficher sur un même graphique les moyennes de distance par jour de 3 coureurs différents. On pourra changer la couleur, le tracé, et les marqueurs des lignes tracées, en rajoutant une chaîne de caractères en 3e argument de la fonction plot.

- Tracés possibles : - ou – ou -. ou :
- Couleurs possibles : b, g, r, c, m, y, k, w
- Marqueurs possibles : o + . x * ∧

Par exemple en ajoutant la chaîne "r–", on trace une ligne pointillée rouge.

Module numpy

Pour tracer des fonctions plus complexes, on va aussi utiliser le module numpy. Rajouter en haut de votre fichier la ligne : `import numpy as np`

Le module numpy fournit en particulier une fonction `linspace` qui crée une liste de valeurs entre les 2 bornes données, contenant un nombre de valeurs égal au 3e paramètre. Cette fonction va nous servir à préparer l'argument de la fonction `plot`. Plus le 3e argument est grand (plus de valeurs dans la liste), plus la courbe est lissée.

Par exemple pour tracer la fonction cosinus on procède comme suit :

```
x = np.linspace(0, 2*np.pi, 300)
y = np.cos(x)
plt.plot(x, y)
plt.show()
```

Voir aussi la documentation :
`https://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html`
et un cours sur : `http://www.courspython.com/introduction-courbes.html`

Polynôme du second degré

Écrire une fonction qui reçoit en paramètres des valeurs a,b,c et affiche la courbe correspondant au polynôme : $a * x^2 + b * x + c$

2.8 TP8 : propagation d'une nouvelle

Notions appliquées : listes, boucles while et for.

2.8.1 Contexte du problème

On veut modéliser et simuler la propagation d'une nouvelle dans une population donnée. On considère une population de N personnes. Pour simplifier, chaque personne est représentée par un numéro unique compris dans l'intervalle $[0; N-1]$. Il y a 3 statuts possibles pour une personne :

- **C** : la personne **C**onnaît la nouvelle et la raconte aux personnes qu'elle rencontre
- **I** : la personne **I**gnore la nouvelle et ne peut donc pas la raconter
- **M** : la personne, dite "**M**uette", connaît la nouvelle, mais pense que tout le monde la connaît ; elle ne la raconte donc plus aux personnes rencontrées.

Initialement, seule la personne numéro 0 est au courant de la nouvelle. Puis chaque jour a lieu une rencontre aléatoire entre 2 personnes de numéros X et Y de la population considérée, rencontre représentée par le couple (X, Y).

La nouvelle est propagée différemment d'une personne à l'autre selon leur "statut" respectif relativement à la nouvelle. Une rencontre (X, Y) a pour effet un éventuel changement de statut de X et de Y, changement qui obéit aux règles de bon sens résumées dans le tableau 2.1. Dans les rencontres du type (C, M) ou (M, C), C raconte la nouvelle à son interlocuteur M qui répond : "je la connaissais", de sorte que C en est "vexé" et devient M. De même, dans une rencontre du type (C, C) chaque C devient M.

$X \backslash Y$	C	I	M
C	M, M	C, C	M, M
I	C, C	I, I	I, M
M	M, M	M, I	M, M

TABLE 2.1 – Évolution des statuts à la suite d'une rencontre

On voit que :

1. Le statut d'une personne ne peut évoluer que dans le sens : I $\rightarrow$ C $\rightarrow$ M
2. Lors d'une rencontre, il n'y a aucun changement de statut pour les deux personnes lorsqu'aucune des deux n'a le statut C avant la rencontre
3. Donc la simulation n'évolue plus une fois que plus personne n'a le statut C.

2.8.2 Exercice préliminaire

Travail "manuel" de compréhension sur un exemple :

On considère une population de 5 personnes ($N = 5$), où initialement seule la personne numéro 0 connaît la nouvelle, puis une rencontre et une seule a lieu chaque jour. On considère cette suite de 9 rencontres :

$$\{(2,4), (1,3), (3,0), (2,1), (0,4), (3,4), (2,3), (4,3), (4,0)\}$$

Dessinez à la main un tableau représentant l'évolution des statuts des 5 personnes après chacune des 9 rencontres consécutives considérées (utilisez le modèle suivant) :

			statuts après la rencontre				
jour	rencontre	statuts initiaux	pers.1	pers.2	pers.3	pers.4	pers.5
0			C	I	I	I	I
1	2, 4	I, I	...	...	...	...	...
...	...	...	...	...	...	...	...
9	...	...	...	...	...	...	...

Vérifiez qu'à l'issue des 9 rencontres, les statuts des personnes numéro 0 à 4 sont respectivement : M, I, I, M, M, et qu'ils sont stables (c'est à dire qu'aucun d'eux ne peut changer lors d'une nouvelle rencontre).

2.8.3 Exercice 1

Simulation d'une suite de rencontres

Il vous est à présent demandé d'écrire un programme `propagation.py` qui, à partir d'un nombres de personnes N saisi au clavier, effectuera une simulation de la propagation d'une nouvelle parmi cette population de personnes (numérotées de 0 à $N-1$), suite à des rencontres **aléatoires**.

Dans l'état initial, la personne numéro 0 connaît la nouvelle (son statut est C) dont on veut simuler la propagation. Les personnes numéros 1 à $N-1$ ignorent cette nouvelle (leur statut est I). Les statuts des personnes (des caractères) seront mémorisés dans une liste.

Chaque étape de la simulation (appelée "jour") correspondra à une rencontre entre 2 personnes. Chaque rencontre sera obtenue par un tirage au hasard des numéros des deux personnes entrant en contact l'une avec l'autre. Il va de soi que deux personnes qui se rencontrent sont nécessairement distinctes l'une de l'autre. Le tirage aléatoire d'une rencontre est obtenu au moyen de la fonction `random.randint()`. Pour chaque rencontre, il s'agit de tirer au hasard 2 numéros de personnes X et Y, tels que $0 \leq X \leq N-1$, $0 \leq Y \leq N-1$ et $X \neq Y$.

Le traitement d'une rencontre consistera à déterminer si le statut de l'une ou l'autre des deux personnes est modifié par la rencontre, compte tenu du statut que chacune d'elle avait avant la rencontre. Il faudra pour cela utiliser la table 2.1 présentée en début d'énoncé (page 80). Dans l'affirmative il faudra mémoriser dans la liste `personnes` cette (ou ces) modification(s) de statut.

La simulation doit s'**arrêter** dès que plus aucune propagation n'est possible, c'est à dire lorsque plus aucune des N personnes n'a le statut C. On ne sait donc pas à l'avance combien de jours la propagation va durer.

Exemple d'affichage que devra produire le programme :

```
jour rencontre nature  st.1 st.2 st.3 st.4 st.5
  0                      C    I    I    I    I
  1    2,  3    I, I     -    -    -    -    -
  2    1,  4    I, I     -    -    -    -    -
  3    0,  1    C, I     -    C    -    -    -
  4    2,  1    I, C     -    -    C    -    -
  5    0,  2    C, C     M    -    M    -    -
  6    2,  0    M, M     -    -    -    -    -
  7    4,  0    I, M     -    -    -    -    -
  8    2,  4    M, I     -    -    -    -    -
  9    3,  1    I, C     -    -    -    C    -
 10    3,  1    C, C     M    M    M    M    I
Nombre d'ignorants :  1
Nombre de muets :  4
```

2.8.4 Exercice 2

On veut maintenant que le programme n'affiche plus que le résultat final de la simulation, c'est à dire le nombre de jours NJ qu'a duré la propagation de la nouvelle, et le nombre d'ignorants NI restants à la fin de la simulation. Créez une copie du programme précédent appelée `propagation2.py`. Modifiez cette copie pour obtenir le résultat demandé. Le test du programme consistera à lancer successivement plusieurs exécutions avec différentes valeurs de N (par exemple 100, 1000, etc.).

2.8.5 Exercice 3

Il s'agit maintenant de modifier le programme de simulation de façon à ce que pour une même valeur de N il réalise NS simulations successives (NS saisi au clavier) et calcule les moyennes des nombres de jours NJ et des nombres d'ignorants NI obtenus à chaque simulation. Créez une copie du programme `propagation2.py` appelée `propagation3.py` et modifiez cette copie de façon à obtenir le résultat attendu. Comparez vos résultats avec ceux obtenus par d'autres étudiants pour une même valeur de N.

2.8.6 Exercice 4

On veut à présent représenter sous forme *semi-graphique* l'évolution des statuts des personnes pendant le déroulement de la simulation. On appelle NC, NM et NI les nombres de personnes ayant respectivement les statuts C, M et I. Chaque jour l'état global de la population est représenté par le triplet (NC, NM, NI), avec : $NC + NM + NI = N$. Initialement (le jour 0), ce triplet est égal à $(1, 0, N - 1)$. Cette nouvelle version du programme devra afficher pour chaque rencontre une suite de caractères représentant la répartition de la population en connaisseurs, muets et ignorants. Le caractère `'#'` sera utilisé pour représenter les connaisseurs, le caractère `'*'` pour représenter les muets et le caractère `'.'` pour représenter les ignorants. Par exemple, pour $N = 10$, $NC = 3$, $NM = 2$ et $NI = 5$, la répartition des statuts des personnes sera représentée par la suite de caractères :`##***.....` On constate que sont affichés dans l'ordre : les connaisseurs, puis les muets, et enfin les ignorants.

Si on reprend l'exemple de simulation pris dans l'exercice 1, l'affichage obtenu avec cette nouvelle version du programme serait :

```
jour rencontre nature st.1 st.2 st.3 st.4 st.5 repartition
  0                    C    I    I    I    I   #....
  1    2,  3    I, I    -    -    -    -    -   #....
  2    1,  4    I, I    -    -    -    -    -   #....
  3    0,  1    C, I    -    C    -    -    -   ##...
  4    2,  1    I, C    -    -    C    -    -   ###..
  5    0,  2    C, C    M    -    M    -    -   #**..
  6    2,  0    M, M    -    -    -    -    -   #**..
  7    4,  0    I, M    -    -    -    -    -   #**..
  8    2,  4    M, I    -    -    -    -    -   #**..
  9    3,  1    I, C    -    -    -    C    -   ##**.
 10    3,  1    C, C    M    M    M    M    I   ****.
```

Créer une copie du programme `propagation.py` appelée `propagation4.py`, puis modifiez cette copie afin que le programme obtenu affiche le résultat souhaité.

2.8.7 Bonus : visualisation avec matplotlib

On veut tracer deux graphiques :

- Pour l'exercice 3, un histogramme [6] montrant pour différentes tailles de population de départ les valeurs moyennes de NJ et NI en fin de NS simulations.
- Pour l'exercice 4, un graphique camembert [7] montrant les proportions NC, NM et NI dans la population totale, et leur évolution au fil des jours de simulation.

6. Voir un exemple à : `https://pythonspot.com/en/matplotlib-histogram/`
7. Voir un exemple à : `https://pythonspot.com/en/matplotlib-pie-chart/`

2.9 TP9 : dictionnaires et traduction

Notions pratiquées : dictionnaires.

2.9.1 Exercice préliminaire : manipulation de dictionnaire français-anglais

Dans ce TP on va manipuler les dictionnaires, une structure de données qui permet d'associer des clés à des valeurs. On va pour cela travailler sur des dictionnaires de langues, et pour commencer sur un dictionnaire français-anglais, qu'on utilisera pour traduire des mots et des textes.

Créer un fichier `traduc1.py` pour cet exercice.

1. Créer une fonction `ajouteMot` qui reçoit en paramètres un dictionnaire $dfren$, un mot français mfr, et sa traduction en anglais wen, et qui permet d'ajouter ce mot dans le dictionnaire $dfren$. La fonction modifie le dictionnaire reçu, et ne renvoie rien.

2. Créer une fonction `supprimeMot` qui reçoit en paramètres un dictionnaire et un mot, et qui permet de supprimer ce mot de ce dictionnaire. Cette fonction modifie le dictionnaire reçu, et ne renvoie rien.

3. Créer une fonction `afficheDico` qui reçoit en paramètre un dictionnaire et qui l'affiche, avec sur chaque ligne un mot suivi de sa traduction sous la forme *"chat = cat"*.

4. Créer une fonction `afficheDicoLettre` qui reçoit en paramètres un dictionnaire et une lettre, et qui n'affiche que les mots commençant par cette lettre, sous le même format que ci-dessus.

5. Créer une fonction `afficheDicoLongueur` qui reçoit en paramètres un dictionnaire et un entier (une longueur), et qui n'affiche que les mots de la longueur donnée (longueur du mot français), sous le même format que ci-dessus.

Écrire ensuite un programme principal avec les étapes suivantes :

6. Créer une variable `fr` contenant un dictionnaire associant à quelques mots français leur traduction en anglais.

7. Appeler la fonction `ajouteMot` définie ci-dessus pour ajouter quelques mots dans le dictionnaire, puis l'afficher avec la fonction `afficheDico` pour vérifier.

8. Utiliser la fonction `supprimeMot` définie ci-dessus pour supprimer un mot du dictionnaire, et l'afficher avec `afficheDico` pour vérifier.

2.9.2 Exercice 2 : mini-jeu de traduction

Créer un nouveau fichier `traduc2.py` pour cet exercice.

Traduction français-anglais

1. Écrire une fonction `traduire` qui reçoit en paramètre le dictionnaire, et un mot français à traduire, et qui renvoie sa traduction en anglais.

2. Écrire une fonction `jouerUnMot` qui reçoit un dictionnaire, choisit au hasard un mot français de ce dictionnaire, et qui demande à l'utilisateur sa traduction en anglais. Cette fonction affiche un message de succès ou d'erreur, et renvoie un booléen indiquant si la réponse de l'utilisateur était correcte ou incorrecte.

3. Écrire un programme principal qui utilise ces deux fonctions pour jouer une partie : chaque tour de jeu correspond à la traduction d'un mot choisi au hasard ; puis le joueur peut choisir de continuer ou d'arrêter ; à la fin de la partie son score est affiché (nombre de traductions correctes sur nombre total de tours).

Traduction dans les 2 sens *(optionnel, non nécessaire pour la suite)*

Modifier ce programme pour pouvoir demander des traductions dans les 2 sens (soit du français vers l'anglais, soit de l'anglais vers le français). Le sens de traduction pourra être donné comme paramètre supplémentaire optionnel des fonctions définies (valeur par défaut : sens français vers anglais).

Que se passe-t-il quand on demande la traduction en français d'un mot anglais qui a plusieurs traductions en français ?

Ce problème se pose-t-il dans l'autre sens de traduction ?

2.9.3 Exercice 3 : multi-joueurs

Créer un nouveau fichier `traduc3.py` pour cet exercice.

1. Écrire une nouvelle version du programme qui demande initialement le nombre de joueurs et le nom de chacun. Un dictionnaire est initialisé avec les scores de chaque joueur.

2. Le programme demande le nombre de tours de la partie, c'est-à-dire le nombre de mots à faire deviner.

3. Les joueurs jouent les mêmes mots (au choix : chaque joueur joue tous ses mots avant de passer au suivant ; ou bien chaque mot est joué par tous les joueurs à tour de rôle).

4. Le score de chaque joueur est calculé en pourcentage de mots correctement traduits par rapport au nombre total de mots de la partie (score entre 0 et 100 %), arrondi à l'entier inférieur.

5. Écrire une fonction qui reçoit le dictionnaire des scores des joueurs et renvoie le nom du meilleur joueur et son score (2 valeurs de retour).

6. Écrire une fonction qui reçoit le dictionnaire des scores des joueurs et affiche le tableau des meilleurs scores : les 3 meilleurs joueurs, un par ligne, avec leur score. En cas d'égalité ils seront affichés par ordre alphabétique mais avec le même numéro de rang. Par exemple :

```
1. Tom    99\%
2. Lea    97\%
2. Nina   97\%
```

7. Tester ces fonctions de score, soit en faisant des parties avec au moins 4 joueurs, soit en entrant manuellement des scores dans le dictionnaire.

2.9.4 Exercice 4 : traduction de texte

Créer un nouveau fichier `traduc4.py` pour cet exercice.

Traduction partielle

1. Écrire une fonction qui reçoit en paramètre une chaîne de caractères (un texte) ainsi qu'un dictionnaire, et qui traduit ce texte de la manière suivante : les mots qui sont dans le dictionnaire sont remplacés par leur traduction, les autres sont laissés tels quels. La fonction renvoie la traduction sous forme d'une chaîne de caractères.
   Rappel : on peut diviser un texte en la liste de ses mots avec la fonction `split`, et on peut reformer une chaîne de caractères à partir d'une liste de mots avec la fonction `join`.

2. Écrire un programme principal qui demande à l'utilisateur de saisir un texte, et affiche sa (semi-)traduction.

Traduction interactive

1. Modifier la fonction de traduction ci-dessus pour qu'elle procède de la manière suivante pour chaque mot du texte à traduire :

   - Si le mot est connu (présent dans le dictionnaire) il est traduit
   - Si le mot n'est pas dans le dictionnaire, le programme demande à l'utilisateur la traduction de ce mot
     - Si l'utilisateur connaît la traduction, elle est ajoutée au dictionnaire, et le mot est traduit dans le texte.
     - Si l'utilisateur ne connaît pas la traduction de ce mot, le mot est laissé tel quel dans la traduction du texte (comme dans la version 1) et le dictionnaire n'est pas modifié.

   La fonction renvoie finalement le texte (complètement) traduit. On notera que cette fonction a aussi pour effet de bord de modifier le dictionnaire reçu en paramètre.

2. Écrire un programme principal qui demande à l'utilisateur un texte, appelle cette fonction pour le traduire, et affiche ensuite le texte traduit et le dictionnaire modifié.

2.9.5 Exercice 5 : multi-lingue

1. Créer des dictionnaires pour une ou plusieurs autres langues (espagnol, allemand, japonais, chinois, etc), sur le même modèle que le dictionnaire français-anglais.
2. Créer un dictionnaire de dictionnaires : les clés sont les noms des langues, et les valeurs sont les dictionnaires français-langue étrangère correspondants
3. Écrire une fonction qui reçoit le dictionnaire de dictionnaires, un nom de langue LG2, un mot XX à traduire, et qui affiche sa traduction YY dans la langue demandée sous la forme : `''La traduction de XX de français vers LG2 est YY''`
4. Écrire un programme principal qui demande à l'utilisateur un mot, puis qui affiche la liste des langues disponibles (les clés du dictionnaire de dictionnaires), lui demande d'en choisir une (l'utilisateur doit taper une chaîne de caractères), et appelle la fonction ci-dessus pour afficher la traduction demandée.
5. Compléter ce programme principal pour demander à l'utilisateur un deuxième mot et afficher sa traduction dans toutes les langues disponibles.
6. Bonus : écrire une fonction qui cherche un mot (donné en argument) dans les dictionnaires de toutes les langues, et qui renvoie la langue de ce mot. Si le mot est présent dans plusieurs langues, la fonction renvoie la liste de ces langues.

2.9.6 Bonus en utilisant les fichiers

Se renseigner sur la librairie `pickle` : `https://docs.python.org/3/library/pickle.html` qui permet de sérialiser des données, c'est-à-dire de les convertir dans un format qui permet de les stocker dans un fichier puis de retrouver leur structure d'origine.

- Enregistrer les dictionnaires de chaque langue dans un fichier dédié.
- Charger le dictionnaire de la langue demandée par l'utilisateur à partir du bon fichier.

2.10 TP10 : fichiers et Python Tutor

Notions pratiquées : manipulation de fichiers, entrées-sorties, chaînes de caractères.

2.10.1 Un peu de PythonTutor

Soit les programmes suivants extraits du fichier `pythontutor.py`, à télécharger.

```
def ajouter_zero(dico, clef):
    copie = dict(dico)
    copie[clef] = 0
    return copie

d = {'a': 12,'b': 25}
d_avec_zero = ajouter_zero(d, 'c')

print("Avant: ", d)
print("Apres: ", d_avec_zero)
```

```
def ajouter_zero_liste(dico, clef):
    copie = dict(dico)
    copie[clef].insert(0, 0)
    return copie

e = {'a': [1,2,3],'b':[4,5,6]}
e_avec_zero = ajouter_zero_liste(e, 'a')

print("Avant: ", e)
print("Apres: ", e_avec_zero)
```

1. Avant d'exécuter les programmes, prédire ce qui sera affiché. Identifiez les différences entre les deux programmes et leur influence sur le comportement attendu.

2. Copier le contenu du fichier `pythontutor.py` sur PythonTutor et observer son exécution pas à pas.

3. Les programmes se comportent-t-ils comme prévu ? Si non, que feriez-vous pour corriger ce problème ?

2.10.2 Chaînes de caractères - Compléments

Dans cette partie, nous allons voir des fonctions relatives aux chaines de caractères qui peuvent être particulièrement utiles lorsque l'on travaille avec des fichiers. Pour cette partie, vous travaillerez sur la console d'IDLE.
Exécutez les instructions suivantes et observez le résultat :

```
>>> message1 = 'Hello world\n'
>>> print(message1)
>>> message2 = 'Hello world\n\n'
>>> print(message2)
>>> print(message2.strip())
>>> message4 = '    hi    '
>>> print(message4.strip())
```

```
>>> 'HELLO'.lower()
>>> 'hello'.upper()
>>> 'HeLLo'.lower()
>>> 'a b c'.split()
>>> 'a:b:c'.split(':')
>>> ','.join(['a','b','c'])
>>> ':'.join(['a','b','c'])
```

1. Question : À quoi sert chacune des fonctions suivantes : `strip()`, `lower()`, `upper()`, `split()` et `join()` ?

   Pour en savoir plus, n'hésitez pas à consulter la documentation :

   Section 4.7.1 *String methods* `https://docs.python.org/3.6/library/stdtypes.html#string-methods`

2. Complétez les programmes ci-dessous :

```
>>> s1='SaLuT'
>>> [          ]
>>> s2
'salut'
```

```
>>> s1='  ok    \n'
>>> [          ]
>>> s2
'ok'
```

```
>>> s1=['yes', 'no', 'maybe']
>>> [          ]
>>> s2
'yes-no-maybe'
```

```
>>> s1='Albus.Severus.Potter'
>>> [          ]
>>> s2
['Albus', 'Severus', 'Potter']
```

2.10.3 Travailler avec des fichiers texte - premiers pas

Nous allons maintenant apprendre à utiliser des fichiers dans nos programmes. Avant de commencer nous allons préparer notre "espace de travail".

1. Créez un dossier appelé "TP8" sur votre bureau. Au long de ce TP, on appellera ce dossier votre "répertoire de travail". Vous devez sauvegarder tous vos programmes, ainsi que les fichiers téléchargés, dans ce même répertoire.

Proverbe du jour

Le fichier `proverbs.txt`, à télécharger, contient un grand nombre de proverbes en anglais en une seule ligne, séparés par des points ".", ce qui le rend assez difficile à lire. On veut d'abord reformater ce fichier pour le rendre plus lisible.

1. Téléchargez le fichier `proverbs.txt` vers votre répertoire de travail.

2. Écrire un programme qui crée un nouveau fichier nommé `proverbs_readable.txt` qui contient les mêmes proverbes que le fichier original mais avec un seul proverbe par ligne. Vérifiez que le fichier a bien été créé dans votre répertoire de travail, et qu'il est bien au bon format.

3. Écrire un programme qui lit l'un des deux fichiers, et affiche un message d'accueil, suivi d'un proverbe aléatoire (utiliser le module `random`). Exemple :

```
Bonjour, voici votre proverbe du jour:
    "Actions speak louder than words"
```

2.10.4 Champion de Scrabble

Dans cette partie, nous allons utiliser les notions vues précédemment combinées avec les dictionnaires pour développer un programme qui sait jouer au célèbre jeu du Scrabble.

Anagrammes

Pour commencer, on veut créer une fonction qui permet de vérifier si un ensemble de lettres (chaîne de caractères) est un anagramme d'un mot déjà connu. A l'aide des dictionnaires, il est assez facile de faire cette vérification. En effet, si on représente chaque mot par un dictionnaire associant à chaque lettre du mot le nombre de fois où elle apparaît dans le mot, le dictionnaire généré pour un mot et celui généré pour un de ses anagrammes sont identiques.

1. En vous inspirant de l'exercice 4 du TD, écrivez une fonction `mot_vers_dictionnaire(mot)`, qui prend un mot (chaîne de caractères) et renvoie le dictionnaire correspondant.

2. En utilisant la fonction `mot_vers_dictionnaire` écrire une fonction `est_anagramme(lettres, mot)` qui envoie `True` si la chaîne de caractères `lettres` est un anagramme du mot `mot`, `False` sinon.

```
>>> est_anagramme("tusla", "salut")
True
>>> est_anagramme("salus", "salut")
False
```

Solution parfaite

En utilisant les fonctions précédemment définies, on peut déjà construire une première version de notre programme. Cette première version va prendre les lettres que l'utilisateur a tirées, et lui afficher les mots français qui peuvent être formés en utilisant TOUS les jetons qu'il a en main.

Le fichier `mots.txt` à télécharger est un fichier texte contenant tous les mots français du petit Larousse. Il contient un mot par ligne, sans accents, et en majuscule.

1. Téléchargez le fichier `mots.txt` vers votre répertoire de travail.

2. Écrire un programme principal qui demande à l'utilisateur une chaîne de caractères, et qui affiche tous les mots du fichier `mots.txt` qui peuvent être formés à partir de ces lettres. (Utiliser la fonction `est_anagramme`). L'utilisateur peut saisir des majuscules ou des minuscules. Utiliser les fonctions vues dans la première partie du TP pour que le programme fonctionne quelle que soit la casse.

```
Quelles lettres avez vous en main ? taslu
Voici les mots que vous pouvez jouer:
SALUT
TALUS
```

Solution exhaustive

Comme il n'est pas toujours possible de former un mot en utilisant toutes les lettres disponibles, on aimerait que notre programme soit également capable de nous renvoyer tous les mots que l'ont peut former en utilisant seulement une partie des jetons à notre disposition. Cela requiert une méthode un peu plus élaborée que celle des anagrammes.

Si on considère toujours la représentation d'un mot par un dictionnaire, pour vérifier qu'une chaine `C` permet de former un mot `M`, il suffit de vérifier que chaque lettre du mot `M` apparait dans la chaine `C` au moins une fois. Autrement dit, pour chaque lettre `L` du mot `M`, le nombre d'apparitions de `L` dans `C` doit être supérieur ou égal au nombre d'apparitions de `L` dans `M`.

1. Ecrire une fonction `peut_ecrire(lettres, mot)` qui retourne True si le mot `mot` peut être écrit en utilisant des lettres de la chaine de caractères `lettres`, False sinon.

```
>>> peut_ecrire("tatxslua", "salut")
True
>>> peut_ecrire("lasautaionna", "salutation") # pas assez de "t"
False
```

2. Écrire maintenant le programme qui permet d'afficher tous les mots français pouvant être formés par les lettres saisies par l'utilisateur. On n'affichera que les mots de longueur 3 ou plus.

```
Quelles lettres avez vous en main ? taslu
Voici les mots que vous pouvez jouer:
ALU AULT LAS LATS LUT SAL SALUT SAUT TALUS TAS TAU USA
```

Solution optimale

Dans le jeu du Scrabble, chaque lettre est associée à un nombre de points. Ce sont ces points qui permettent de calculer le score de chaque joueur à la fin de la partie. On aimerait que notre programme soit capable de choisir, parmi les mots qu'on peut jouer, celui qui a le plus grand total de points.

Pour ce faire, on stocke les points correspondant à chaque lettre dans un fichier texte appelé `points.txt`. Chaque ligne de ce fichier contient une lettre et son nombre de points, séparés par des espaces.

1. Téléchargez le fichier `points.txt` vers votre répertoire de travail.
2. Ecrire une fonction `lire_points()` qui lit le fichier `points.txt` et renvoie un dictionnaire faisant correspondre à chaque lettre son nombre de points (entier).
3. Ecrire une fonction `score_mot(points, mot)` qui prend en argument un dictionnaire faisant correspondre à chaque lettre son nombre de points, et un mot, et qui renvoie le score.
4. Modifier le programme de l'exercice précédent pour qu'il n'affiche que le meilleur mot à jouer, ainsi que le score correspondant.

```
Quelles lettres avez vous en main ? taslu
Voici le meilleur mot que vous pouvez jouer:
SALUT pour un score de: 9
```

Avec Jokers

Dans le jeu du scrabble, il existe des jetons blancs appelés Jokers. Ces jetons peuvent être utilisés pour remplacer n'importe quelle lettre de l'alphabet. Les jokers valent 0 points. Dans notre programme, le joker sera représenté par le caractère '_'.

1. Reprenez le programme défini dans l'exercice 4 en modifiant la fonction `peut_ecrire` afin qu'elle prenne en compte les jokers dans la chaine saisie par l'utilisateur.

```
Quelles lettres avez vous en main ? r_sg
Voici les mots que vous pouvez jouer:
...
GARS GPRS GRAS GRE GRES GRIS GROS ...
```

2. Pouvez-vous modifier le programme de l'exercice 5 de sorte à ce que les jokers ne rentrent pas dans le calcul du score ?

2.11 TP11 : révisions

Notions pratiquées : tout le programme

2.11.1 Retour au journal de M. Bizarre

Avec listes

1. Les fonctions écrites précédemment ne transforment qu'un seul mot. Écrire un programme qui les utilise (ainsi que la fonction `split()` qui permet de découper une chaîne) pour transformer tout un texte mot par mot.

2. Plus difficile : écrire ensuite un programme qui reçoit un texte transformé et qui détermine le jour de la semaine dont on a appliqué la règle. *Il s'agit de trouver pour chaque jour une condition booléenne que vérifie une chaîne ainsi transformée.*

Avec fichiers

3. Reprendre l'exercice initial, mais le programme doit lire le texte initial dans un fichier "journal.txt", et écrire les résultats de chaque transformation dans un fichier par jour de la semaine (lundi.txt, mardi.txt, etc).

2.11.2 Un menu et des lettres

Écrire les fonctions suivantes, qui reçoivent un caractère en argument et qui :

1. Vérifie s'il est en minuscule ou majuscule

2. Vérifie si c'est une voyelle ou une consonne

3. Calcule sa position dans l'alphabet

4. Calcule son code ASCII

Écrire ensuite un programme principal qui affiche le menu en proposant les 4 options codées ci-dessus. L'utilisateur doit taper '@' pour quitter. Tant qu'il ne quitte pas, le programme ré-affiche le menu après le résultat de chaque fonction demandée.

2.11.3 Tri à bulle

Le tri à bulle est une méthode de tri par propagation [8] : on échange progressivement les éléments mal positionnés, ce qui fait remonter les éléments les plus grands vers la fin de la liste.

1. Comprendre l'algorithme du tri à bulle en déroulant un exemple sur papier

2. Écrire une fonction qui reçoit une liste en paramètre et la modifie pour la trier avec cet algorithme

3. Utiliser le module `turtle` pour afficher les éléments de la liste sous forme de rectangles (même largeur, hauteur dépendant de leur valeur), afin d'animer graphiquement le tri.

2.11.4 Dictionnaire

On considère un dictionnaire de randonnées sous la forme suivante :

```
randos = { 1 : {'deniv': 1200, 'distance': 17.8, 'temps': 5.5}, 2: {...}}
```

Les clés sont des numéros identifiants uniques, et les valeurs sont des dictionnaires indiquant pour chaque randonnée son dénivelé en mètres, sa distance en kilomètres, et un temps de parcours en heures.

1. Écrire une fonction qui crée et renvoie un dictionnaire de randonnées en demandant le dénivelé, la distance et le temps à l'utilisateur. L'identifiant unique est généré automatiquement dans l'ordre de saisie.

2. Écrire une fonction qui crée et renvoie un dictionnaire de randonnées aléatoire : pour chaque randonnée, choisir aléatoirement une distance et un dénivelé crédible, puis estimer le temps de parcours (on pourra écrire une fonction auxiliaire), et créer la nouvelle entrée dans le dictionnaire avec l'identifiant suivant.

8. Voir `https://en.wikipedia.org/wiki/Bubble_sort`

3. Écrire une fonction qui reçoit en paramètre une chaîne de caractères (valant soit 'distance', soit 'deniv', soit 'temps'), une valeur maximum, et qui renvoie une liste de tous les numéros de randonnées dont la caractéristique nommée est inférieure ou égale à la valeur donnée. Si la clé passée en paramètre n'est pas correcte, renvoyer la liste vide.

4. Écrire une fonction qui reçoit en paramètre le dictionnaire et un numéro de randonnée, et qui affiche son descriptif de manière lisible, sous la forme : "Randonnée numéro XX : XX km, XX m de D+, environ XXhXXmn". *On remarque qu'il faudra convertir le nombre réel d'heures (par exemple 5.5) en nombre d'heures et de minutes (par exemple 5h30mn).*

5. Écrire un programme principal qui demande à l'utilisateur son critère de choix, puis sa valeur maximale, puis lui affiche la liste des randonnées correspondant à ce critère (avec la fonction d'affichage précédente), une par ligne.

6. Écrire une fonction qui permet de modifier le dictionnaire pour y ajouter un champ 'chrono' une fois une randonnée réalisée par l'utilisateur. *Toutes les entrées n'auront donc pas forcément de champ chrono, ce qui permettra de reconnaître les randonnées déjà réalisées.*

7. Modifier la fonction de recherche précédente, pour y ajouter un paramètre optionnel `new`, valant par défaut `False`. Si ce paramètre vaut `True` alors la fonction de recherche ne doit renvoyer dans la liste que des randonnées pas encore réalisées par l'utilisateur (champ 'chrono' inexistant).

8. Écrire une fonction qui permet de supprimer le champ 'chrono' d'une randonnée donnée par son numéro.

2.11.5 Jeu du pendu

Créer un jeu de pendu

- Le programme choisit un mot au hasard dans une liste et donne X erreurs maximum à l'utilisateur pour le deviner, lettre par lettre. Il affiche un tiret par lettre du mot.
- L'utilisateur choisit une lettre, le programme lui répond si elle apparaît ou non dans le mot. Si elle apparaît, le squelette du mot à trouver est affiché avec toutes les occurrences de cette lettre. Si elle n'apparaît pas, le nombre d'erreurs encore autorisées diminue de 1.
- Le jeu se termine quand l'utilisateur a trouvé toutes les lettres du mot (victoire) ou quand il a consommé toutes ses erreurs (défaite).
- Améliorations possibles : afficher à chaque tour le nombre d'erreurs encore autorisées, et les lettres déjà tentées (et qui n'apparaissent pas) ; afficher le nombre d'occurrences d'une lettre correctement devinée

Exemple d'exécution

```
Devine mon mot : _ _ _ _ _ _
Tu as droit a 3 erreurs
a
Il n'y a pas de a
Devine mon mot : _ _ _ _ _ _
Tu as droit a 2 erreurs (a)
p
Il y a 1 p
Devine mon mot : p _ _ _ _ _
e
Il n'y a pas de e
Devine mon mot : p _ _ _ _ _
Tu as droit a 1 erreur (a e)
y
Il y a 1 y
Devine mon mot : p y _ _ _ _
Tu as droit a 1 erreur (a e)
u
Il n'y a pas de u
Tu as perdu
Le mot etait : python
```

Chapitre 3

Annales

INF101/INF131 : devoir surveillé

Jeudi 26 Octobre 2017

- **Durée 2 heures.**
- **Lisez toutes les questions avant de commencer !** Le sujet fait quatre (4) pages.
- Respectez strictement les consignes de l'énoncé (noms de variables, format d'affichage...)
- Pensez à commenter vos programmes pour les rendre plus lisibles.
- Les exercices sont indépendants, et en ordre croissant de difficulté.
- Aucun document autorisé. L'utilisation d'une calculette, d'un téléphone, etc est interdite.
- Le barème est indicatif. La qualité de la rédaction et de la présentation sera prise en compte.

1 EXERCICE 1 : BOOLEENS (2.5 points)

On suppose que toutes les variables nécessaires sont bien initialisées. Écrire les expressions booléennes vraies si:

- a est un entier strictement positif et multiple de 3
- a est soit positif pair, soit négatif impair
- c est une lettre de l'alphabet en minuscule ou en majuscule (**remarque** : on interdit d'utiliser la fonction `isalpha` de Python; attention c pourrait être une chaîne de plusieurs caractères)
- x, y, z sont dans l'ordre croissant (non strict) et tous strictement négatifs
- x est un nombre réel non entier

2 EXERCICE 2 : CONVERSIONS (2.5 points)

On veut écrire un programme pour aider un coureur à calculer son allure de course. Chaque coureur connaît sa vitesse maximale en km/h (par exemple 21.2 km/h). Son entraîneur lui demande de courir à un certain pourcentage de cette vitesse maximale (par exemple 75%). Sa montre n'affiche pas la vitesse mais l'allure en mn/km (par exemple 5 mn/km correspond à 12 km/h). Écrire un programme qui réalise les opérations suivantes :

1. Demander au coureur de saisir sa vitesse maximale en km/h, qui doit être strictement positive, et inférieure ou égale à 30. Le programme filtre la valeur saisie jusqu'à ce qu'elle respecte cette contrainte (variable vmax).
2. Demande au coureur de saisir le pourcentage de cette vitesse auquel il doit courir (entier entre 0 et 100), et filtre cette valeur jusqu'à ce qu'elle soit bien comprise entre 0 et 100 (variable pc).
3. Calcule la vitesse (variable v) correspondant à ce pourcentage et l'affiche.
4. Calcule l'allure (variable a) correspondant à la vitesse v et l'affiche. (Attention on veut l'allure en minutes et secondes par kilomètres. Le nombre de minutes doit être un entier, le nombre de secondes un réel qu'on pourra arrondir à 2 chiffres après la virgule avec la fonction `round`).

On demande de respecter **strictement** l'affichage de l'exemple suivant (à l'espace près) :

```
Quelle est ta vitesse maximale en km/h ? 15.4
Quel pourcentage (0-100) ? 180
Quel pourcentage (0-100) ? 80
La vitesse correspondant à 80 pourcents de ton max est : 12.32 km/h
L'allure correspondant à 12.32 km/h est : 4'52''/km
```

Rappel : on peut spécifier le séparateur voulu dans l'affichage en précisant sep=...

3 EXERCICE 3 : PYRAMIDE (4 points)

Écrire un programme qui :

1. demande à l'utilisateur un nombre de lignes (variable nbl) et une lettre de départ (variable dep). On suppose que l'utilisateur saisit toujours une minuscule, on n'a pas besoin de le vérifier.
2. affiche une pyramide de lettres à nbl lignes, sous la forme suivante. Si on dépasse la fin de l'alphabet on veut recommencer à la lettre 'a' après le 'z'.

Par exemple: à gauche pour 4 lignes (nbl=4) et départ à 'a' (dep='a'); à droite pour 3 lignes (nbl=3) et départ à 'u' (dep='u').

```
   a                 u
  b c d             v w x
 e f g h i         y z a b c
j k l m n o p
```

Rappels :

- la fonction `ord(caractere)` renvoie le code ASCII du caractère reçu en paramètre. La fonction `chr(code)` renvoie le caractère ayant le code ASCII reçu en paramètre.
- `print` a 2 paramètres optionnels, `sep` qui permet de modifier le séparateur entre les chaînes affichées (espace par défaut), et `end` qui permet de modifier la fin de l'affichage (retour à la ligne par défaut).

4 Exercice 4 : mélange de liste (3 points)

1. Écrire une fonction *melange_liste* qui reçoit en paramètre une liste, et renvoie une liste contenant les mêmes éléments dans le désordre.
   Attention : on ne veut PAS modifier la liste reçue en paramètre mais renvoyer une nouvelle liste.
   Indice : on peut faire une copie de la liste reçue, dans une nouvelle variable qui pourra elle être modifiée.
2. Écrire ensuite un programme principal qui :
   (a) crée une liste vide
   (b) demande à l'utilisateur de saisir des entiers, et les ajoute à cette liste. L'utilisateur doit taper -1 pour terminer la saisie (-1 n'est pas ajouté à la liste)
   (c) affiche la liste ainsi constituée de tous les éléments saisis (dans l'ordre de saisie)
   (d) mélange cette liste en appelant la fonction écrite ci-dessus
   (e) affiche la liste mélangée

Indice : on utilisera la fonction `randint` du module `random`. **Remarque** : le module `random` contient aussi déjà une fonction `shuffle` qui permet de mélanger une liste, mais on demande de ne **PAS** l'utiliser ici, le but de l'exercice est de la réécrire. (Par ailleurs cette fonction modifie la liste manipulée, alors qu'on demande ici de renvoyer une copie mélangée de la liste de départ sans modifier celle-ci.)
Rappel de quelques fonctions utiles sur les listes : `len` (longueur d'une liste) ; `append` (ajouter un élément à la fin d'une liste) ; `pop` (récupérer l'élément à un certain indice et le supprimer de la liste).

5 Exercice 5 : message secret (8 points)

5.1 Fonctions

Définir les fonctions suivantes permettant de coder un mot de différentes manières selon le jour de la semaine :

1. La fonction `lundi` reçoit un mot, choisit un entier n au hasard entre 2 et 7, et renvoie le mot répété n fois, séparés par des +. Par exemple si le mot est "salut" et que l'entier tiré est 3, le résultat est "salut+salut+salut". On remarquera qu'il n'y a **pas** de + à la fin de la chaîne.

2. La fonction `mardi` reçoit un mot, et renvoie ce mot écrit à l'envers. Par exemple si le mot est "bonjour", le résultat est "ruojnob".
   Rappel : si c est une chaîne de caractères alors list(c) permet d'obtenir la liste de toutes ses lettres.

3. La fonction `mercredi` reçoit un mot, choisit un entier x au hasard entre 1 et 10, et opère un décalage de x positions sur chaque lettre du mot (en rebouclant sur 'a' après le 'z' ou sur 'A' après le 'Z'). Par exemple le mot "Salut" avec un décalage de 7 sera codé en "Zhsba" car 'Z' est situé 7 lettres après 'S' dans l'alphabet en majuscules, 'h' est situé 7 lettres après 'a' dans l'alphabet en minuscules, etc. Pour 'u' et 't' on voit qu'on dépasse le 'z' et on revient donc au début de l'alphabet. On pourra écrire une fonction auxiliaire qui opère le décalage sur une lettre (en considérant bien les majuscules et les minuscules).

4. La fonction `jeudi` reçoit un mot, supprime toutes ses voyelles (majuscules comme minuscules), et renvoie le mot constitué des lettres (consonnes) restantes. Par exemple "salut" devient "slt", "Alice" devient "lc". On pourra écrire une fonction auxiliaire qui teste si une lettre est une voyelle.

5. La fonction `vendredi` reçoit un mot, change les minuscules en majuscules, et vice-versa. Par exemple "Salut" devient "sALUT". On pourra écrire une fonction auxiliaire qui transforme une minuscule en majuscule et vice versa.
   Remarque : on **interdit** ici d'utiliser les fonctions `upper`, `lower`, `isupper`, `islower`, déjà définies dans Python; on veut les reprogrammer.

6. La fonction `samedi` reçoit un mot, mélange ses lettres, et renvoie le mot mélangé.
   Remarque: on pourra cette fois utiliser la fonction `shuffle` du module `random`. **Rappel** : la fonction `shuffle` reçoit une liste, la mélange, et ne renvoie rien; c'est la liste reçue qui est modifiée pour être mélangée.

7. La fonction `dimanche` reçoit un mot mais ne fait aucun codage et le renvoie tel quel.

5.2 Programme principal

8. Définir ensuite un programme principal qui :
   (a) Opère en boucle en demandant un mot à l'utilisateur, jusqu'à lire le mot "stop".
   (b) Pour chaque mot saisi par l'utilisateur (sauf "stop"), lui demande un jour de la semaine et affiche le résultat du codage de ce mot selon la fonction correspondant au jour choisi.
   (c) Recommence ensuite avec le mot suivant.

9. **Question bonus :** Écrire un nouveau programme principal qui, au lieu de lire un seul mot, lit un texte entier. Ensuite le programme demande un jour de la semaine, code chaque mot du texte selon la fonction correspondante, et affiche le texte résultant. De la même manière qu'à la question 8, ce programme s'arrête si le texte saisi est "stop".

INF101/INF131 : examen final

Mardi 19 Décembre 2017

- **Durée 3 heures.**
- **Lisez toutes les questions avant de commencer !** Le sujet fait 5 pages.
- Respectez **strictement** les consignes de l'énoncé (noms des fonctions et variables, format d'affichage...)
- Les exercices sont indépendants et ne sont **pas** dans l'ordre de difficulté. Vous pouvez sauter des exercices et des questions. Vous pouvez supposer existantes les fonctions des questions précédentes et les utiliser, même si vous n'avez pas répondu à la question correspondante.
- Il est inutile de filtrer les entrées lorsque ce n'est pas explicitement demandé (elles seront supposées correctes).
- **Aucun document autorisé. L'utilisation d'une calculette, d'un téléphone, etc, est interdite.**
- **Chaque question vaut 1 point**. L'examen est donc noté sur 22. Le barème est indicatif. La qualité de la rédaction et de la présentation sera prise en compte. Pensez à commenter vos programmes pour les rendre plus lisibles. Attention à l'indentation. Répondez à **chaque exercice sur une feuille séparée**.

1 Nombres narcissiques (5 points)

Un nombre narcissique (ou nombre d'Armstrong de première espèce) est un entier naturel n non nul qui est égal à la somme des puissances p-ièmes de ses chiffres en base dix, où p désigne le nombre de chiffres de n :

$$n = \sum_{k=0}^{p-1} x_k 10^k = \sum_{k=0}^{p-1} (x_k)^p \quad avec \quad x_k \in \{0, \ldots, 9\} \quad et \quad x_{p-1} \neq 0.$$

Par exemple, tous les entiers de 1 à 9 sont narcissiques ; 153 est narcissique car $153 = 1^3 + 5^3 + 3^3$; 548834 est narcissique car $548834 = 5^6 + 4^6 + 8^6 + 8^6 + 3^6 + 4^6$.

1. Écrire une fonction `nombreChiffres` qui calcule et renvoie le nombre p de chiffres d'un entier naturel non nul x reçu en argument.
2. Écrire une fonction `sommePuissances` qui reçoit en argument un entier naturel x, et calcule et renvoie la somme des puissances p-ièmes de ses chiffres, où p est son nombre de chiffres (qui sera calculé avec la fonction précédente).
3. Écrire une fonction `narcissique` qui reçoit en argument un entier naturel x, et renvoie un booléen indiquant si ce nombre est narcissique.
4. Écrire une fonction `listeNarcissiques` qui reçoit en argument une borne maximum, et qui construit et renvoie la liste des nombres narcissiques inférieurs ou égaux à cette borne. On rappelle qu'un entier narcissique doit être non nul. Si la borne reçue en argument est inférieure ou égale à 0, la liste renvoyée doit donc être vide.
5. Écrire un programme principal qui demande une borne maximum à l'utilisateur, puis affiche la liste des nombres narcissiques inférieurs ou égaux à cette borne, sur une seule ligne, séparés par des virgules, avec un retour à la ligne final. Par exemple :

```
bmax? 10
1,2,3,4,5,6,7,8,9
bmax? 200
1,2,3,4,5,6,7,8,9,153
```

Tournez la page

2 Estimation du nombre π (4 points)

Il existe plusieurs formules pour calculer des estimations du nombre π. On considère en particulier dans cette exercice les deux formules suivantes :

- $\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - ...$: cette formule de la somme alternée des inverses des nombres impairs, s'approche très très lentement du nombre π.
- $\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + ...$: cette somme des inverses des carrés des entiers converge plus rapidement que la formule précédente.

En Python, on peut obtenir la valeur de π grâce à `math.pi`, ce qui nous permettra de tester la précision du calcul des estimations. On appelle précision la différence en valeur absolue entre l'estimation calculée par la formule, et la valeur de π donnée par `math.pi`. *Par exemple : estimPi*1(1) *calcule une estimation de* π *valant 4. La précision de cette estimation est* $|4 - \pi|$ *c'est-à-dire environ 0.8585.*

1. Écrire une fonction `estimPi1` qui reçoit en paramètre une borne max (l'entier impair constituant le dénominateur de la dernière fraction de la somme), calcule et renvoie l'estimation correspondante de π selon la première formule. Attention la formule permet de calculer $\pi/4$ et pas π. *Par exemple :* estimPi1(7) estime $\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7}$ puis calcule et renvoie π à partir de cette valeur.
2. Écrire une fonction `estimPi2` qui reçoit en paramètre une borne max n (l'entier dont le carré constitue le dénominateur de la dernière fraction de la somme), calcule et renvoie l'estimation correspondante de π selon la deuxième formule. Attention la formule permet de calculer $\pi^2/6$ et pas π. On rappelle que `math.sqrt` permet de calculer la racine carrée. *Par exemple :* estimPi2(6) estime $\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \frac{1}{6^2}$, puis calcule et renvoie π à partir de cette valeur.
3. Écrire une fonction `estimPrecision` qui reçoit en paramètre une précision souhaitée dans l'approximation de π (c'est-à-dire l'écart maximum souhaité, en valeur absolue, entre l'estimation calculée et la valeur de π obtenue avec `math.pi`), et appelle la fonction `estimPi2` ci-dessus pour calculer les estimations successives de π (avec des valeurs croissantes de n) jusqu'à obtenir la précision souhaitée. La fonction renvoie alors la borne max n nécessaire, et l'estimation de π obtenue. *Par exemple : estimPrecision(0.1) renvoie (10, 3.04936163598207) ; estimPrecision(0.01) renvoie (96, 3.131681463952958)*
4. Écrire un programme principal qui demande à l'utilisateur une précision p (nombre réel), puis affiche la borne max nécessaire pour atteindre cette précision, et l'estimation correspondante. Le programme doit ensuite proposer à l'utilisateur de rejouer avec une nouvelle précision, jusqu'à ce que l'utilisateur refuse.

Tournez la page

3 Jeu de la vie de Conway (8 points)

Le jeu de la vie de Conway est un automate cellulaire dont les règles sont extrêmement simples. On considère une grille carrée de cellules qui peuvent être soit vivantes soit mortes. L'évolution de l'état d'une cellule dépend de son nombre de voisines vivantes parmi les 8 cellules qui l'entourent (cf figure 1):

- Une cellule morte qui a exactement 3 voisines vivantes devient vivante (elle naît);
- Une cellule vivante qui a exactement 2 ou 3 voisines vivantes reste vivante, sinon elle meurt.

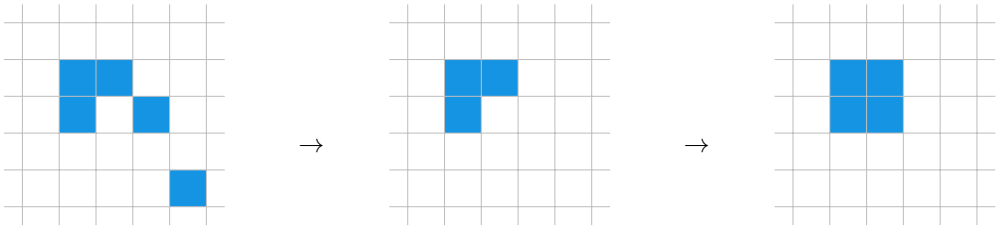


Figure 1: Un exemple d'évolution sur trois étapes d'un extrait de grille dans le jeu de la vie. Les cellules mortes sont en blanc, les cellules vivantes en bleu.

On choisit ici de représenter cette grille de la manière suivante :

- L'état de chaque cellule est un booléen : True pour vivante, False pour morte.
- La grille est une liste de listes : chaque élément de la liste est une liste contenant les états de toutes les cellules de cette ligne.
- On appelle taille de la grille le nombre de cellules par côté. Une grille de taille N est donc une liste de N listes de N éléments.

Par exemple la grille de gauche sur la figure 1 ci-dessus est de taille 5 et est représentée par la liste suivante :

```
[[False, False, False, False, False],
 [False, True,  True,  False, False],
 [False, True,  False, True,  False],
 [False, False, False, False, False],
 [False, False, False, False, True ]]
```

1. Écrire une fonction `initGrilleM(N)` qui reçoit en argument la taille N de la grille carrée, et initialise et renvoie une grille carrée de cette taille (N cellules par N cellules) contenant uniquement des cellules mortes. **Attention** à créer des listes différentes pour chaque ligne, et non pas des références à la même liste.

2. Écrire une fonction `initGrilleV(N,V)` qui appelle la fonction précédente pour créer une grille de taille N de cellules mortes, puis qui place **exactement** V cellules vivantes (à des positions au hasard), et renvoie cette grille. On suppose que V a une valeur correcte c'est-à-dire positif et inférieur ou égal à N*N. *Indice: il s'agit, pour chaque cellule vivante à placer, de tirer au hasard des coordonnées dans la grille, jusqu'à trouver une cellule morte, puis de rendre cette cellule vivante.*

3. Écrire une fonction `afficheSolution` qui reçoit en paramètre une grille et l'affiche de la manière suivante : les cellules vivantes sont représentées par le caractère '*' (étoile), et les cellules mortes par le caractère '.' (point), sans espace entre les valeurs. Par exemple la grille de booléens

ci-dessus (qui correspond à la grille de gauche sur la figure) sera affichée sous la forme suivante :

```
.....
.**..
.*.*.
.....
....*
```

4. Écrire une fonction `grilleEtendue` qui reçoit une grille de taille N, et qui calcule et renvoie une grille de taille N+2 en rajoutant des cellules mortes (valeur False) tout autour de la grille reçue. Cela permettra d'éviter les effets de bord dans le comptage des voisins vivants des cellules. *On rappelle les fonctions* `insert` *et* `append` *pour insérer un élément dans une liste ou à la fin d'une liste respectivement.* Par exemple pour la grille de taille 5 ci-dessus, cette fonction renvoie la grille de taille 7 suivante :

```
.......
.......
..**...
..*.*..
.......
.....*.
.......
```

5. Écrire une fonction `compteVoisins(grille,i,j)` qui reçoit la grille de taille N, utilise la fonction ci-dessus pour calculer la grille étendue de taille N+2, puis compte et renvoie le nombre de voisins vivants de la cellule de la ligne d'indice i et colonne d'indice j de la grille. **Attention** : les indices reçus sont supposés corrects, et sont exprimés dans la grille de taille N, avant extension : ils sont donc compris entre 0 et N-1. Grâce à l'extension de la grille, toutes les cellules ont exactement 8 voisines et il n'est donc pas nécessaire de gérer des cas particuliers dans les coins ou sur les bords.
Par exemple pour la grille ci-dessus et les indices (0,1), la fonction compte les voisins vivants de la cellule de la 1e ligne et 2e colonne de la grille initiale (numérotées à partir de 0), c-à-d de la 2e ligne et 3e colonne dans la grille étendue, et renvoie 2 (une cellule vivante dessous, et une cellule vivante dessous à droite).

6. Écrire une fonction `evoluerGrille` qui reçoit la grille en argument, et crée et renvoie une nouvelle grille contenant le nouvel état de toutes les cellules après une évolution. La grille initiale n'est pas modifiée. Toutes les cellules évoluent **simultanément**, c'est-à-dire qu'on compte le nombre de voisins vivants dans l'ancienne grille (celle reçue en paramètre), afin de calculer le nouvel état qui sera stocké dans la nouvelle grille (celle renvoyée par la fonction). Pour créer la nouvelle grille, on pourra utiliser la fonction `initGrilleM` définie ci-dessus.

7. Écrire une fonction `nombreVivantes` qui reçoit une grille en argument, et parcourt cette grille pour calculer et renvoyer le nombre de cellules vivantes qu'elle contient.

8. Écrire un programme principal qui appelle les fonctions définies ci-dessus pour :

   - Initialiser une grille carrée de taille 5 par 5, avec 7 cellules vivantes, et afficher cette grille initiale.
   - Faire évoluer cette grille pendant 10 tours, en affichant à chaque tour le numéro du tour, la nouvelle grille, et le nombre de cellules vivantes.
   - Ce programme s'arrête après les 10 tours, ou avant dès qu'il n'y a plus aucune cellule vivante dans la grille.

Tournez la page

4 Dictionnaire de cryptage (5 points)

On s'intéresse dans cet exercice à des dictionnaires dont les clés sont les lettres de l'alphabet en minuscule, et dont les valeurs sont aussi des lettres minuscules de l'alphabet. Un tel dictionnaire associe chaque lettre à une lettre (elle-même ou une autre), chaque lettre n'apparaissant comme valeur que d'une seule clé, réalisant ainsi une permutation de l'alphabet. Cela permet donc de coder un mot ou un texte en remplaçant chaque lettre par une autre.

1. Écrire une fonction `initDico` qui crée et renvoie un dictionnaire qui associe aléatoirement à chaque lettre de l'alphabet une autre lettre. **Attention** : chaque lettre ne doit être utilisée qu'une seule fois. *Par exemple* si la clé "e" est associée à la valeur "a", alors aucune autre clé ne doit être associée à cette même valeur. Ce dictionnaire constitue donc une permutation de l'alphabet. On pourra par exemple utiliser la fonction `shuffle`.

2. Écrire une fonction `codeMot` qui reçoit un mot et un dictionnaire dont les clés et les valeurs sont des lettres de l'alphabet. Cette fonction calcule et renvoie ce mot codé par ce dictionnaire, c'est-à-dire dans lequel chaque lettre est remplacée par sa valeur dans le dictionnaire. On suppose que le mot reçu ne contient que des lettres minuscules.

3. Écrire une fonction `plusLointaine` qui reçoit en argument un tel dictionnaire, et qui calcule et renvoie quelle lettre (clé) est associée dans ce dictionnaire avec la lettre la plus lointaine, et la valeur de l'écart. Pour cela on calculera l'écart en valeur absolue entre la clé et la valeur associée (par exemple si la clé 'e', 5e lettre, est associée à la valeur 'a', 1e lettre, l'écart est de 4; si la clé 'i', 9e lettre, est associée à la valeur 'z', 26e lettre, l'écart est de 17). On rappelle les fonctions `ord` et `chr` permettant de convertir un caractère en code ASCII et vice-versa.

4. Écrire une fonction `dicoDecode` qui reçoit en argument un dictionnaire associant chaque lettre à son codage par une autre lettre, et qui génère et renvoie le dictionnaire permettant de **décoder** les mots codés avec ce dictionnaire. Par exemple à partir du dictionnaire 'a':'e','b':'c','c':'d','d':'a','e':'b', cette fonction renvoie le dictionnaire inversé 'a':'d','b':'e','c':'b','d':'c','e':'a'. Ainsi si on code un mot avec le premier dictionnaire, puis qu'on code le résultat avec le deuxième dictionnaire, on doit retomber sur le mot initial.

5. Écrire un programme principal qui :

   - Génère un dictionnaire aléatoire,
   - Calcule dans une variable `ll` la lettre qui est associée à la lettre la plus lointaine, et dans une variable `em` l'écart correspondant, puis affiche ces deux valeurs avec un texte clair
   - Demande à l'utilisateur un mot, le code avec ce dictionnaire et affiche le résultat,
   - Puis calcule le dictionnaire inverse, décode le mot et affiche le résultat.

FIN

INF101/INF131 : examen 2e session

Lundi 18 Juin 2018

- **Durée 2 heures.**
- **Lisez toutes les questions avant de commencer !** Le sujet fait 2 pages.
- Respectez **strictement** les consignes de l'énoncé (noms des fonctions et variables, format d'affichage...)
- Les exercices sont indépendants et ne sont **pas** dans l'ordre de difficulté. Vous pouvez sauter des exercices et des questions. Vous pouvez supposer existantes les fonctions des questions précédentes et les utiliser, même si vous n'avez pas répondu à la question correspondante.
- Il est inutile de filtrer les entrées lorsque ce n'est pas explicitement demandé (elles seront supposées correctes).
- **Aucun document autorisé. L'utilisation d'une calculette, d'un téléphone, etc, est interdite.**
- **Chaque question vaut 1 point**. Le barème est indicatif. La qualité de la rédaction et de la présentation sera prise en compte. Pensez à commenter vos programmes pour les rendre plus lisibles. Attention à l'indentation. Répondez à **chaque exercice sur une feuille séparée**.

1 Triplets de Pythagore (5 points)

Un triplet de Pythagore est un ensemble de 3 entiers naturels, $a < b < c$, tels que $a^2 + b^2 = c^2$. Par exemple, $3^2 + 4^2 = 9 + 16 = 25 = 5^2$.

1. Écrire une fonction `triplets` qui calcule et renvoie la liste des triplets de Pythagore tels que a est inférieur ou égal à une borne maximale reçue en paramètre.
2. Écrire une fonction `tripletsSomme` qui reçoit en paramètre un entier s, et affiche le triplet de Pythagore tel que $a + b + c = s$ s'il existe, ou un message d'erreur sinon. On se servira de la fonction précédente. *Indice: quelle est la valeur maximum de a pour une somme s donnée ?*
3. Écrire un programme principal qui demande à l'utilisateur une somme, et utilise la fonction précédente pour afficher le triplet de Pythagore dont la somme correspond à celle demandée (s'il existe).

2 Codage de mots (4 points)

Dans cet exercice, on appelle 'mot' une chaîne de caractères constituée uniquement de lettres minuscules de l'alphabet.

- Écrire une fonction `codage1` qui reçoit en paramètre un mot, et renvoie la somme des positions de ses lettres dans l'alphabet. Par exemple pour le mot 'salut', on renverra 73 (19+1+12+21+20).
- Écrire une fonction `codage2` qui reçoit en paramètre un mot et renvoie le mot obtenu en remplaçant chaque lettre par la suivante (le 'z' devient un 'a'). Par exemple 'salut' devient 'tbmvu'.
- Écrire une fonction `codage3` qui reçoit en paramètre une entier strictement positif, et qui renvoie le mot constitué des lettres dont la position correspond à chaque chiffre, en comptant à partir de 0 (il ne peut donc y avoir que des lettres entre a, position 0, et j, position 9). Par ex : 1203 devient 'bcad'; 2708 devient 'chai'. *Attention au sens !*
- Écrire une fonction `code4` qui reçoit en paramètre un mot, et renvoie un mot composé des mêmes lettres mais mélangées dans un ordre aléatoire. *Indice: on pourra utiliser la fonction `shuffle` du module `random`, sans oublier de l'importer.*

Tournez la page

3 Démineur (6 points)

On s'intéresse à une grille de démineur carrée, représentée par une liste de listes d'entiers. Une case vide est représentée par l'entier 0, une case minée par l'entier 1. Pour éviter les effets de bord, on stocke une grille de démineur de taille n dans une liste de n+2 listes de n+2 entiers (les entiers placés sur les bordures de la grille valent nécessairement 0).

1. Écrire une fonction `initGrille` qui reçoit en paramètre un entier n, calcule et renvoie une liste de listes de taille n+2, ne contenant que des entiers 0.
2. Écrire une fonction `placeMines` qui reçoit en paramètre une liste de listes et un entier x (nombre de mines). Cette fonction ajoute exactement x mines dans la grille, en évitant les bordures. Cette fonction modifie directement la grille reçue en paramètre, elle ne renvoie rien. *Indice: on pourra tirer une position vide au hasard avec* `random.randint(inf,sup)`.
3. Écrire une fonction `affiche` qui reçoit en paramètre une grille d'entiers, et l'affiche sous forme d'un carré, les entiers de chaque ligne séparés par un espace, avec un retour à la ligne entre chaque ligne.
4. Écrire une fonction `lignePlusMinee` qui reçoit en paramètre une grille, cherche la ligne contenant le plus de mines, et renvoie le numéro de cette ligne et le nombre de mines qu'elle contient.
5. Écrire une fonction `ligneNonMinee` qui reçoit en paramètre une grille carrée, cherche la première ligne ne contenant aucune mine, et renvoie son numéro, ou -1 si une telle ligne n'existe pas.
6. Écrire un programme principal qui demande à l'utilisateur la taille de la grille de démineur et le nombre de mines, utilise les fonctions ci-dessus pour initialiser la grille, y placer le bon nombre de mines, et l'afficher sous forme d'un carré. Le programme affiche ensuite un message indiquant le numéro de la ligne la plus minée et son nombre de mines, et le numéro de la première ligne non minée si elle existe.

4 Dictionnaire de traduction (5 points)

1. Écrire une fonction `initListeMots` qui ne reçoit aucun paramètre, demande à l'utilisateur de saisir des mots français en terminant par 'stop', enregistre tous ces mots dans une liste (sauf 'stop'), et renvoie cette liste.
2. Écrire une fonction `initDicoAnglais` qui reçoit une liste de mots français, demande à l'utilisateur la traduction de chaque mot en anglais, et initialise un dictionnaire français-anglais (clé = mot français, valeur = mot anglais).
3. Écrire une fonction `supprimerDoublons` qui reçoit en paramètres un dictionnaire français-anglais et une liste de mots français; elle parcourt les mots français pour vérifier qu'il n'y a pas plusieurs mots ayant la même traduction dans le dictionnaire reçu. Si c'est le cas, alors les doublons sont supprimés du dictionnaire (on ne garde que le premier mot ayant cette traduction). Cette fonction ne renvoie rien, elle modifie le dictionnaire reçu en paramètre (effet de bord). *Indice: on a besoin de parcourir la liste des mots plutôt que les clés du dictionnaire, car on n'a pas le droit de modifier le dictionnaire pendant qu'on le parcourt.*
4. Écrire une fonction `inverseDico` qui reçoit en paramètre un dictionnaire français-anglais, et l'inverse pour créer un dictionnaire anglais-français *(on suppose qu'il n'y a pas plusieurs mots ayant la même traduction).*
5. Écrire un programme principal qui utilise les fonctions précédentes pour :
   - Créer un dictionnaire nommé `fr2en` associant des mots français (saisis par l'utilisateur) à leur traduction anglaise (saisie par l'utilisateur).
   - Créer un dictionnaire nommé `en2fr` en inversant le dictionnaire ci-dessus. On prendra soin de supprimer les doublons avant l'inversion.
   - Demander à l'utilisateur de saisir un mot et un langue ('en' ou 'fr'), puis utiliser le bon dictionnaire pour traduire le mot dans l'autre langue, et afficher le résultat.
   - Lui proposer de rejouer avec un autre mot et langue, jusqu'à ce qu'il refuse de continuer.
   - Afficher 'fin du programme" avant de se terminer.

FIN

Projet 2016 : la réussite des alliances

1 Organisation du projet

Pour ce projet, vous vous inscrirez en binômes sur CaseInE. Les deux membres d'un binôme doivent être dans le même groupe de TD.

Le projet sera partiellement évalué sur CaseInE mais la note finale sera donnée par votre chargé de TP. CaseInE permettra surtout de valider un certain nombre de fonctions.

Date limite de rendu sur CaseInE : vendredi 28 avril 2017 à 18h59. Vous devrez compléter le fichier `student.py` de l'activité associée, et rendre un rapport au format pdf. Ce rapport, d'un maximum de deux pages donnera le mode d'emploi de la fonction 4.5.2, précisera les extensions que vous avez faites, et présentera de manière synthétique les résultats expérimentaux obtenus : moyennes, maximum et minimum des valeurs obtenues (en précisant le nombre de simulations), graphiques divers que vous aurez générés avec votre programme. Ce compte-rendu sera aussi déposé sur CaseIne, à la même date de rendu que le programme.

Votre programme devra être clair et commenté. Toutes les fonctions (y compris celles non demandées) devront être documentées.

Le rendu et les inscriptions en binômes sur CaseInE seront disponibles aux alentours du 3 avril.

2 Le thème du projet : la réussite des alliances

Dans ce projet, nous nous intéressons à une réussite peu connue qui se joue avec un jeu de 32 ou de 52 cartes : la réussite des alliances. On commence par donner les règles de cette réussite.

On mélange les cartes, on obtient un tas de cartes qui va constituer la pioche. On prend les cartes une à une dans la pioche et on les pose côte à côte de gauche à droite. Quand on a placé trois cartes, on considère les cartes 1 et 3 (en les numérotant de gauche à droite). Si elles sont de même couleur ou de même valeur (ce qu'on appellera une alliance), on prend la carte située entre elles (ici la carte numéro 2) et on la pose sur la carte 1 (sinon on ne fait rien et on continue à piocher). C'est ce qu'on va appeler un "saut". On n'a donc potentiellement plus que deux tas.

Par exemple, si les 3 premières cartes piochées sont dans l'ordre un 8 de cœur, un 7 de carreau et un roi de trèfle, aucun saut n'est possible :

$$8\heartsuit \quad 7\diamondsuit \quad R\clubsuit$$

Par contre, si les 3 premières cartes piochées sont dans l'ordre un 8 de cœur, un 7 de carreau et un roi de cœur :

$$8\heartsuit \quad 7\diamondsuit \quad R\heartsuit$$

on devra faire "sauter" le 7 de carreau sur le 8 de cœur avant de piocher la carte suivante, et on obtiendra :

$$7\diamondsuit \quad R\heartsuit$$

On continue alors à piocher des cartes et à les poser une par une sur la droite de la réussite. Dès qu'il y a une alliance entre la dernière carte posée et l'antépénultième, le tas placé entre les deux est décalé et posé sur le tas à sa gauche ("saut"). Ainsi à tout moment de la partie, on a des tas de cartes, posés de gauche à droite. Pour un tas donné, le nombre de cartes du tas importe peu dans la mesure où il n'y a que la carte visible sur le dessus du tas qui compte pour la suite de la partie. On parlera donc de tas, même si le tas en question est composé d'une seule carte.

Par exemple, si on continue la partie suivante, en supposant qu'on pioche successivement l'as de cœur, le 10 de carreau et le 9 de pique, aucun saut ne sera possible, et on aura :

$$7\diamondsuit \quad R\heartsuit \quad A\heartsuit \quad 10\diamondsuit \quad 9\spadesuit$$

On pioche donc la carte suivante, un 9 de carreau :

$$7\diamondsuit \quad R\heartsuit \quad A\heartsuit \quad 10\diamondsuit \quad 9\spadesuit \quad 9\diamondsuit$$

Ici un saut est possible car le tas du 9 de pique est situé entre deux carreaux, et on pose donc le tas du 9 de pique sur le tas du 10 de carreau :

$$7\diamondsuit \quad R\heartsuit \quad A\heartsuit \quad 9\spadesuit \quad 9\diamondsuit$$

Parfois, lorsqu'on fait un saut, cela rend d'autres sauts possibles, ce n'est pas le cas ici. Si au moins un autre saut est possible, il faut le faire avant de piocher la carte suivante. Si plusieurs sauts sont possibles, il faut décider lequel on fait en premier. On décide de faire toujours en premier le saut le plus à gauche (ce qui peut en entraîner un autre encore plus à gauche).

Pour illustrer ces sauts en cascade, imaginons par exemple qu'à un moment de la partie, on ait le jeu suivant :

$$9\clubsuit \quad V\diamondsuit \quad 10\heartsuit \quad A\spadesuit \quad V\clubsuit \quad D\clubsuit \quad D\spadesuit$$

Ici, il n'y a toujours aucun saut possible. On pioche la carte suivante, un 7 de trèfle, et on la place à droite :

$$9\clubsuit \quad V\diamondsuit \quad 10\heartsuit \quad A\spadesuit \quad V\clubsuit \quad D\clubsuit \quad D\spadesuit \quad 7\clubsuit$$

Ce 7 de trèfle est un trèfle tout comme la carte situé deux rangs avant lui, donc on prend le tas de la dame de pique qui est entre le 7 de trèfle et la dame de trèfle et on le pose sur le tas de la dame de trèfle. Après ce saut, on obtient :

$$9\clubsuit \quad V\diamondsuit \quad 10\heartsuit \quad A\spadesuit \quad V\clubsuit \quad D\spadesuit \quad 7\clubsuit$$

Ici c'est intéressant car on a deux sauts possibles : on peut faire sauter le tas du valet de trèfle ou le tas de la dame de pique. Mais si on pose d'abord la dame de pique sur le valet de trèfle on ne pourra plus faire l'autre saut, d'où l'idée de toujours commencer par le saut le plus à gauche. Donc on fait sauter la carte (ou le tas de cartes) situé entre l'as de pique et la dame de pique, et c'est donc le tas du valet de trèfle qui sera posé sur l'as de pique :

$$9\clubsuit \quad V\diamondsuit \quad 10\heartsuit \quad V\clubsuit \quad D\spadesuit \quad 7\clubsuit$$

Maintenant, avant de faire l'autre saut qu'on avait repéré, il faut vérifier s'il n'y en a pas un nouveau plus à gauche. On repart du début du jeu à gauche pour chercher les sauts possible. Ici un nouveau saut est apparu entre le valet de carreau et le valet de pique. On pose donc le tas du 10 de cœur sur le valet de carreau :

$$9\clubsuit \quad 10\heartsuit \quad V\clubsuit \quad D\spadesuit \quad 7\clubsuit$$

Le prochain saut à faire est de poser le tas du 10 de cœur sur le 9 de trèfle :

$$10\heartsuit \quad V\clubsuit \quad D\spadesuit \quad 7\clubsuit$$

Et maintenant le premier saut à faire est celui que nous avions repéré il y a quelques étapes déjà : on pose le tas de la dame de pique sur le valet de trèfle.

$$10\heartsuit \quad D\spadesuit \quad 7\clubsuit$$

Ainsi sans piocher d'autre carte après le 7 de trèfle, on est passé d'un jeu à 8 tas à un jeu à 3 tas.

Ici aucun autre mouvement n'est possible. On pioche la carte suivante et on continue la réussite.

La partie s'arrête quand on a épuisé la pioche et qu'on ne peut plus faire diminuer le nombre de tas. On compte alors le nombre de tas restants à la fin de la réussite. On a gagné si le nombre de tas est inférieur à un certain seuil fixé par la règle du jeu. On testera plusieurs seuils dans ce projet. On a toujours au moins 2 tas à la fin (et le dernier n'a qu'une carte), c'est la fin "parfaite" mais c'est suffisamment rare pour qu'on soit plus tolérant pour estimer que la réussite est gagnée, surtout si on joue avec 52 cartes.

3 Représentation informatique de cette réussite

Dans ce projet, on va programmer cette règle de réussite et faire plusieurs études pour estimer par exemple la probabilité de gagner une partie.

Pour modéliser cette réussite, on va considérer qu'à tout moment de la partie, la réussite est modélisée par une liste de cartes qui sont les cartes visibles sur les dessus des tas. Ainsi le nombre de tas à la fin de la partie sera tout simplement le nombre d'éléments de cette liste.

Pour coder une carte dans le programme, on va utiliser un dictionnaire à deux clés : `'valeur'` et `'couleur'`.

Pour la valeur de la carte, un as sera représenté par `'A'`, un deux par l'entier `2`, un trois par l'entier `3` et un dix par `10`. Pour les figures, on écrira respectivement `'V'`, `'D'` et `'R'` pour valet, dame et roi.

Pour la couleur, on aimerait utiliser l'initiale majuscule de chaque couleur (Cœur, Pique, Carreau et Trèfle) mais comme cœur et carreau ont la même initiale, on choisit la lettre `'K'` pour représenter carreau.

Ainsi le dictionnaire {`'valeur' :7, 'couleur' : 'P'`} représentera le 7 de pique.

La liste `[{'valeur':7, 'couleur':'P'}, {'valeur':10, 'couleur':'K'}, {'valeur':'D', 'couleur':'K'}]` correspondra à l'état suivant de la réussite :

7♠ 10♢ D♢

La pioche sera également modélisée par une liste de cartes, la carte à l'indice 0 étant la première carte à piocher.

On va maintenant pouvoir écrire les fonctions permettant de faire jouer l'ordinateur à cette réussite.

4 Travail demandé

Dans cette partie, on a réparti les différentes fonctions à écrire dans plusieurs catégories. Autant que possible, testez vos fonctions au fur et à mesure que vous les écrivez.

Attention : pour toutes les fonctions demandées dans ce projet, quand il y a des listes en argument, ces listes ne doivent pas être modifiées par la fonction, sauf si c'est explicitement demandé. N'hésitez pas à faire une copie de la liste si nécessaire.

4.1 Affichage de la réussite

4.1.1 Fonction `carte_to_chaine`

Pour afficher la réussite sur la console Python, on souhaite avoir un affichage plus explicite que `"DK"` par exemple pour la dame de carreau. On va afficher les cartes de façon assez similaire à ce qui est fait dans les exemples de la partie 2.

Pour cela, on doit savoir représenter les caractères spéciaux ♡ ♠ ♢ ♣. En fait, en Python, chaque caractère correspond à un entier qui code ce caractère. Pour savoir quel est le code de la lettre `'a'` par exemple, il faut faire afficher `ord(a)`. Vous trouverez alors que l'entier associé au caractère `'a'` est 97. Réciproquement, pour obtenir le caractère correspondant à un entier donné, il faut utiliser la fonction `chr`. Ainsi par exemple `chr(97)`

retournera `'a'`. Voici les codes qui vous seront utiles :

Caractère	Code
♠	9824
♡	9825
♢	9826
♣	9827

Ainsi, par exemple, pour créer la chaîne de caractères permettant d'afficher la dame de carreau, il faudra faire : `'D' + chr(9826)`

Ecrire la fonction `carte_to_chaine` qui prend en argument un dictionnaire représentant une carte, et retourne une chaîne de caractères permettant l'affichage de la carte correspondante, sur 3 caractères, avec éventuellement un espace à gauche. Un espace est nécessaire pour toutes les cartes qui ne sont pas un 10. Attention, cette fonction ne produit aucun affichage.

Exemples : (les espaces ont été matérialisés par le caractère ␣)

- `carte_to_chaine({'valeur':7, 'couleur':'P'})` retourne la chaîne "␣7♠"
- `carte_to_chaine({'valeur':10, 'couleur':'K'})` retourne la chaîne "10♢"
- `carte_to_chaine({'valeur':'R', 'couleur':'C'})` retourne la chaîne "␣R♡"

4.1.2 Fonction `afficher_reussite`

Ecrire une fonction `afficher_reussite` qui prend en argument une liste de cartes correspondant à l'état de la réussite à un moment donné et affiche la réussite. On utilisera la fonction `carte_to_chaine`. On affichera les différentes cartes sur une même ligne, séparées par un espace (en plus de celui éventuellement mis par `carte_to_chaine`).

Exemple :
`afficher_reussite([{'valeur':7, 'couleur':'P'}, {'valeur':10, 'couleur':'K'}, {'valeur':'A', 'couleur':'T'}])` affichera "␣7♠␣10♢␣␣A♣" suivi d'un saut de ligne (2 retours à la ligne).

4.2 Entrées / Sorties avec des fichiers

4.2.1 Fonction `init_pioche_fichier`

Cette fonction lit une suite de cartes dans le fichier dont le nom est passé en argument, et renvoie la liste de cartes correspondante. On suppose que les valeurs dans le fichier sont correctes, c'est-à-dire qu'elles correspondent toutes à une carte existante, et qu'il y a le bon nombre de cartes (32 ou 52). Dans le fichier, on a la liste des cartes dans l'ordre où elles seront piochées, les cartes étant séparées par des espaces. Pour chaque carte, il est écrit sa valeur puis sa couleur, séparées par un tiret. Par exemple, si on a "7-K", cela correspondra au 7 de carreau et "D-P" représentera la dame de pique. Pour cette question, un fichier d'exemple de pioche `data_init.txt` est fourni sur CaseInE.

4.2.2 Fonction `ecrire_fichier_reussite`

On souhaite pouvoir sauvegarder dans un fichier l'ordre des cartes de la pioche d'une réussite donnée, par exemple pour mémoriser le mélange qui a donné le meilleur résultat parmi plusieurs mélanges.

Ecrire pour cela la fonction `ecrire_fichier_reussite` qui prend en argument un nom de fichier `nom_fich` dans lequel il faut écrire la liste des cartes `pioche` fournie en deuxième argument de la fonction. Le fichier ainsi écrit doit pouvoir être lu par la fonction `init_pioche_fichier`. Cette fonction ne renvoie rien.

4.3 Générer une pioche mélangée aléatoirement

Ecrire la fonction `init_pioche_alea`. Cette fonction a un seul argument, optionnel, nommé `nb_cartes` qui vaut 32 par défaut, mais qui peut aussi valoir 52. Cette fonction renvoie une liste contenant toutes les cartes du jeu, et les mélange (pour mélanger une liste, vous pourrez utiliser la fonction `shuffle` du module `random`).

4.4 Programmer les règles de la réussite des alliances

4.4.1 Fonction `alliance(carte1,carte2)`

Ecrire une fonction `alliance(carte1,carte2)` qui prend en argument deux cartes, et qui renvoie vrai si les cartes ont la même valeur ou la même couleur et faux sinon.

Exemples :

- `alliance({'valeur':7, 'couleur':'P'},{'valeur':7, 'couleur':'C'})` renvoie vrai
- `alliance({'valeur':7, 'couleur':'P'},{'valeur':8, 'couleur':'C'})` renvoie faux.

4.4.2 Fonction `saut_si_possible`

Cette fonction prend en argument une liste `liste_tas` correspondant à la liste des cartes visibles sur les tas de la réussite et un entier `num_tas`. La fonction vérifie s'il est possible de faire sauter le tas d'indice `num_tas` sur le tas qui le précède (selon les règles de la réussite). Si le saut est possible, la fonction le fait. La fonction retourne vrai si le saut a été fait et faux sinon. Elle n'affiche rien.

Attention : cette fonction modifie donc la liste qui lui est donnée en argument.

4.4.3 Fonction `une_etape_reussite`

Cette fonction prend en argument une liste `liste_tas` correspondant aux cartes visibles des tas de la réussite, une liste `pioche` contenant les cartes restant dans la pioche ainsi qu'un argument optionnel booléen `affiche` valant `False` par défaut. Cette fonction doit effectuer une étape de la réussite, c'est-à-dire :

- tirer la première carte de `pioche` et la placer dans `liste_tas` pour qu'elle corresponde au tas le plus à droite de la réussite,
- faire le saut s'il est possible entre la carte qu'on vient de poser et le tas situé deux rangs plus à gauche (c'est-à-dire faire sauter la carte entre ces deux tas),
- si un saut a été fait, il faudra vérifier si cela a rendu possible d'autres sauts. Pour cela on partira de la gauche de `liste_tas` et on fera le premier saut possible. On recommencera cette action tant qu'un changement aura lieu.
- en plus de tout ça, si l'argument `affiche` vaut vrai, la fonction affichera chacun des états de la réussite pour l'étape en cours. C'est-à-dire qu'à chaque changement (pioche ou saut) il faudra faire un affichage de la réussite. Et si `affiche` vaut faux, rien ne sera affiché.

Attention : cette fonction ne retourne rien mais elle modifie les deux listes qui lui sont données en argument (car on a enlevé la première carte de la `pioche` pour la mettre dans `liste_tas`).

Il est recommandé d'utiliser des fonctions auxiliaires pour programmer correctement cette fonction.

4.5 Faire une partie

4.5.1 Fonction `reussite_mode_auto`

Cette fonction prend en argument une liste de cartes `pioche` correspondant à l'ensemble du jeu de cartes déjà mélangé et un booléen optionnel `affiche` ayant les mêmes spécifications qu'à la question précédente. Si `affiche` vaut vrai, on commence par afficher le contenu de la pioche avant de jouer la réussite en affichant les étapes. Cette fonction joue l'ensemble de la réussite en affichant chaque étape si `affiche` vaut vrai. Le second affichage correspond alors à la première carte de la pioche retournée. La fonction ne doit pas modifier la liste `pioche` (vous pouvez en faire une copie) et elle doit retourner la liste des cartes visibles sur les tas restant à la fin de la réussite.

4.5.2 Fonction `reussite_mode_manuel`

On souhaite maintenant laisser l'utilisateur prendre lui-même les décisions comme si c'était lui qui jouait quitte à ce qu'il rate des sauts, mais sans le laisser tricher.

Pour cela, écrire une fonction `reussite_mode_manuel` qui, comme la fonction précédente, prend en argument une liste de cartes `pioche` correspondant à l'ensemble du jeu de cartes déjà mélangé. Ici, il n'y a pas d'argument `affiche` car on veut forcément afficher les étapes pour que le joueur puisse voir où il en est. Par contre, la fonction prendra un argument optionnel `nb_tas_max` (par défaut, réglé à 2) qui fixe le nombre maximum de tas qu'il peut rester à la fin pour que la partie soit déclarée comme gagnante. L'ordinateur proposera un menu au joueur : il lui proposera par exemple de découvrir une carte de la pioche, ou de saisir un saut à faire. Il faudra aussi que le joueur puisse quitter s'il le souhaite. Si le joueur demande à faire un saut non autorisé, l'ordinateur doit refuser de le faire en précisant que le coup est impossible.

Si le joueur abandonne avant d'avoir retourné toutes les cartes, les cartes restant dans la pioche sont découvertes une à une et posées sur la réussite, sans qu'aucun saut ne soit fait. Dans tous les cas, à la fin, la fonction affiche au joueur s'il a gagné ou perdu, et retourne la liste des cartes visibles sur les tas restant à la fin de la réussite.

Cette question est volontairement assez ouverte. A vous de trouver le format qui rendra le jeu le plus agréable, et qui proposera éventuellement d'autres options.

4.5.3 Fonction `lance_reussite`

Écrire une fonction `lance_reussite` qui prend en argument une chaîne de caractères `mode` qui vaudra soit `'manuel'` soit `'auto'`, ainsi que trois arguments optionnels `nb_cartes` (par défaut, 32) et un booléen `affiche` (par défaut, False) et un entier `nb_tas_max` (par défaut, 2). Cette fonction doit mélanger aléatoirement un jeu contenant le bon nombre de cartes et lancer la réussite en mode manuel ou en mode auto. L'argument `affiche` ne sert que pour le mode auto, et l'argument `nb_tas_max` ne sert que pour le mode manuel. La fonction doit renvoyer la liste des tas restants à la fin de la partie.

4.6 Extensions

Dans cette partie, on propose un certain nombre d'extensions ou d'améliorations possibles. Vous n'êtes pas obligés de toutes les faire mais plus vous réussirez à en faire, meilleure sera votre note. Ici, les instructions sont moins directives que dans la partie précédente, à vous de prendre des décisions pour faire au mieux. Vous pouvez aussi imaginer d'autres extensions.

Pour chaque extension faite, vous ajouterez dans le programme principal du code très bien commenté permettant d'illustrer au mieux cette nouvelle fonctionnalité et vous préciserez ce qu'elle fait et les résultats qu'elle donne dans votre compte-rendu.

4.6.1 Fonction verifier_pioche

Jusqu'à maintenant, on n'a jamais vérifié si la liste des cartes donnée comme pioche en entrée des fonctions `reussite_mode_auto` ou `reussite_mode_manuel` correspondait bien à un vrai jeu de cartes non truqué : pas de carte en double, toutes les cartes présentes.

La fonction `verifier_pioche` prend en argument la liste des cartes constituant la pioche et un entier `nb_cartes` qui vaut 32 ou 52 (argument optionnel, 32 par défaut) et elle vérifie si cette pioche contient bien toutes les cartes d'un jeu à `nb_cartes`, sans doublon. Elle renvoie vrai si c'est le cas et faux sinon.

4.6.2 Estimer le nombre moyen de tas

- Ecrire la fonction `res_multi_simulation` qui prend en argument un nombre entier `nb_sim` et en argument optionnel un entier `nb_cartes` correspondant au nombre de cartes dans le jeu (32 par défaut). La fonction génère `nb_sim` mélanges aléatoires d'une pioche d'un jeu de `nb_cartes` cartes, effectue les réussites en mode automatique, sans affichage, et retourne la liste des nombres de tas obtenus à la fin de chaque réussite. (par exemple, si `nb_sim` vaut 3, la fonction retourne [5, 3, 12] si la première réussite s'est finie avec 5 tas, la deuxième avec 3 tas et la troisième avec 12 tas).

- Ecrire la fonction `statistiques_nb_tas` qui prend en argument un nombre entier `nb_sim` et en argument optionnel un entier `nb_cartes` correspondant au nombre de cartes dans le jeu (32 par défaut). La fonction calcule et affiche avec des messages clairs la moyenne, le minimum et le maximum du nombre de tas obtenus sur `nb_sim` réussites automatiques correspondant à `nb_sim` mélanges aléatoires de la pioche. Vous pouvez également afficher d'autres indicateurs qui vous sembleront pertinents.

Pour cette partie, n'oubliez pas de résumer les résultats numériques obtenus dans votre compte-rendu, en précisant le nombre de simulations et le nombre de cartes du jeu.

4.6.3 Estimer la probabilité de gagner et faire un graphique

Dans les règles officielles de cette réussite, on a gagné si on a seulement deux tas à la fin. Vous avez dû vous rendre compte que cela arrive assez rarement, surtout quand on a 52 cartes. On souhaiterait pouvoir quantifier ce "assez rarement" et voir comment augmentent nos chances de gagner si on autorise plus de tas (par exemple si on décide que finalement une réussite sera gagnante si elle se finit avec au maximum 4 tas). Ces probabilités semblent difficiles à obtenir par un calcul théorique, par contre, il est relativement aisé de les estimer grâce à des simulations.

En faisant un nombre important de simulations de mélange aléatoire, faites en sorte d'estimer la probabilité de gagner pour chaque nombre maximum de tas possiblement autorisé par la règle (entre 2 et 32 pour un jeu de 32 cartes).

Vous pouvez stocker ces probabilités dans une liste. Créez cette liste à partir d'une fonction.

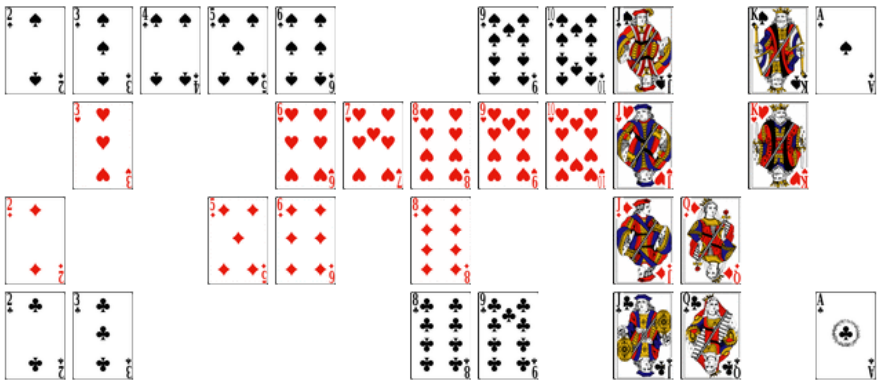
Pour mieux visualiser ces données, vous pouvez faire un graphique en utilisant le module `pyplot` de la librairie `matplotlib`. Pour eviter d'avoir à écrire le nom `matplotlib.pyplot` devant chaque fonction de ce module que vous utiliserez, vous pouvez l'importer ainsi : `import matplotlib.pyplot as plt` et ainsi il suffira d'écrire `plt` devant chaque fonction utilisée. Pour visualiser la fenêtre graphique, après avoir donné à l'ordinateur toutes les instructions pour créer le graphique, il ne faudra pas oublier de faire `plt.show()`. Vous trouverez la documentation de `pyplot` ici :

`http://matplotlib.org/api/pyplot_api.html` *ou tapez pyplot tutorial sans un moteur de recherche*

4.6.4 Interface graphique avec Turtle

Récupérez le fichier `affichage.py` ainsi que le répertoire `imgs` (qui contient les images des cartes). Exécutez et observez ce programme d'affichage.

En vous inspirant de ce programme, proposez un affichage graphique de la réussite en cours d'exécution.



4.6.5 Améliorer un mélange en trichant "un peu"

Si vous avez un peu pratiqué cette réussite, vous vous serez rendus compte que l'on a souvent l'impression qu'à pas grand-chose près, en trichant juste un peu, en échangeant discrètement deux cartes consécutives qui ne sont vraiment pas arrivées dans le bon ordre, le résultat aurait été très différent. On va tester si cette impression est justifiée.

Ecrivez une fonction `meilleur_echange_consecutif` qui prend en paramètre la liste des cartes correspondant à la pioche et cherche la meilleure façon d'améliorer cette réussite en faisant un seul échange, entre deux cartes consécutives. Cette fonction a deux valeurs de retour : elle renvoie la liste des cartes correspondant à la pioche donnée en entrée dans laquelle on a fait le meilleur échange possible entre deux cartes consécutives et elle renvoie aussi le nombre de tas que nous fait gagner cet échange (3 par exemple si la réussite initiale donnait 7 tas à la fin et que celle après avoir fait le meilleur échange donne 4 tas à la fin). Si plusieurs échanges conduisent au résultat optimal (nombre de tas minimum), on choisira de faire l'échange qui est le plus loin dans la pioche (le plus grand indice dans la liste `pioche`).

Etudiez en quoi ça améliore les statistiques de vos parties (nombre moyen, minimum et maximum de tas).

Vous pouvez éventuellement générer de nouveaux graphiques d'estimations de probabilité quand on s'autorise un tel échange de deux cartes dans la pioche avant la partie.

4.6.6 Améliorer un mélange en trichant un peu plus

Sur le modèle de l'extension précédente, à vous d'imaginer d'autre façons de modifier un peu l'ordre de la pioche pour améliorer le score à la réussite. Etudiez si ça améliore beaucoup les résultats.

Chapitre 4

Sujet de projet

INF101 : Projet - Jeu de Uno

Dans ce projet on se propose de réaliser un jeu de Uno. Certaines fonctions sont guidées, mais vous pouvez (et devez) aussi laisser libre cours à votre imagination pour ajouter des fonctionnalités, une interface, etc.

1 Partie A - gestion du paquet de cartes et des mains des joueurs

1.1 Les cartes du jeu

Le paquet de cartes du Uno contient les 108 cartes, réparties dans 4 couleurs différentes (rouge, bleu, jaune, vert) sauf 8 cartes multicolores. Dans chaque couleur il y a 19 cartes numérotées (une carte 0 et deux cartes pour chaque chiffre de 1 à 9), et 6 cartes spéciales : 2 cartes "+2", 2 cartes "Changement de sens", 2 cartes "Passe ton tour". Les 8 cartes multicolores sont : 4 cartes "Joker", et 4 cartes "+4". On représentera chacune de ces cartes par un entier entre 1 et 108 (qu'on appellera indice de la carte dans le paquet), et le paquet par une liste d'entiers.

1.2 Initialisation et mélange du paquet

Écrire une fonction qui initialise le paquet en mélangeant les 108 entiers (entre 1 et 108).

Fonctions utiles : range, random.randInt

1.3 Conversions

Écrire des fonctions de conversion :

- une fonction pour obtenir la couleur d'une carte donnée par son indice (entre 1 et 108),
- une fonction qui permet d'obtenir le numéro d'une carte (entre 0 et 9 pour les chiffres, 10 pour le +2, 11 pour le "change sens", 12 pour le "passe tour") donnée par son indice dans le paquet (entre 1 et 108).
- une fonction qui reçoit l'indice d'une carte et renvoie le texte complet correspondant (par exemple "2 bleu" ou "change tour jaune" ou "Joker")

1.4 Pioche

Écrire une fonction qui reçoit une liste (le paquet), y pioche une carte (ce qui doit modifier la liste) et renvoie la carte piochée.

1.5 Main du joueur

On représente maintenant la main d'un joueur (les cartes qu'il a en main) par une liste d'indices de cartes (liste d'entiers entre 1 et 108).

Écrire une fonction piocher qui reçoit 3 arguments (la pioche, la main du joueur, le nombre de cartes à piocher) et qui modifie les 2 listes en faisant piocher ce nombre de cartes (qui doivent donc être supprimées de la pioche et ajoutées à la main).

1.6 Pioche vide ?

Le jeu ne s'arrête pas quand la pioche est vide, mais on mélange la pile de jeu pour reconstituer la pioche. Écrire une fonction qui reçoit en argument la pioche et la pile de jeu, et mélange la pile de jeu dans la pioche. Attention la dernière carte reste au-dessus du jeu pour savoir quoi jouer, il ne faut pas la remettre dans la pioche.

Cette fonction sera appelée quand la pioche ne contient plus assez de cartes pour piocher le nombre de cartes voulues.

2 Partie B - les règles de base du jeu

Les joueurs jouent à tour de rôle une carte de leur main sur la pile centrale. On représente le jeu par une liste, dont la dernière carte est celle visible sur la table.

Les règles du jeu précisent quelle carte peut être jouée (par le joueur dont c'est le tour) selon celle qui est en haut de la pile:

- une carte de la même couleur (n'importe quel numéro, un "+2", "passe tour" ou "change sens") que la carte courante
- une carte identique à la carte courante (même numéro ou type de carte spéciale) de n'importe quelle couleur
- un Joker sur n'importe quelle carte de n'importe quelle couleur, sauf sur un +2 et un +4
- un +4 sur n'importe quelle autre carte

2.1 Vérification des règles

Écrire une ou des fonctions pour vérifier si on peut poser une carte sur la carte courante, selon les différentes règles ci-dessus.

2.2 Joue ou pioche

Si un joueur ne peut poser aucune carte à son tour de jeu, il doit à la place piocher une carte.

Écrire une fonction qui vérifie si un joueur peut jouer (au moins une de ses cartes est jouable), et éventuellement une fonction qui renvoie l'indice de la (première) carte jouable (ou -1 si aucune).

Un cas particulier se produit si la carte centrale est un +2 ou un +4 : si le joueur ne peut pas jouer il doit piocher le nombre de cartes égal à la somme de tous les +2 et +4 de la séquence en cours. Par exemple si 3 cartes +2 et une carte +4 ont été posées et que le joueur n'a aucune carte à poser dessus, il doit piocher 3*2+4=10 cartes. Pour gérer ce cas écrire une fonction qui renvoie le nombre de cartes à

piocher par le joueur selon la carte centrale et sa main. Si ce nombre vaut 0 cela signifie que le joueur peut jouer, sinon il doit piocher.

2.3 Choix de la carte à jouer

Écrire une fonction qui demande au joueur quelle carte il veut jouer, filtre jusqu'à ce que la carte choisie soit bien jouable selon les règles, et renvoie la carte choisie.

Attention il ne faudra appeler cette fonction que s'il y a bien une carte jouable dans la main du joueur, sinon elle ferait une boucle infinie.

3 Partie C - multi-joueur et tours de jeu

On veut maintenant gérer les tours de jeu pour savoir quel joueur doit jouer en prochain. Par défaut le sens de rotation se fait dans le sens des aiguilles d'une montre, mais les cartes "change sens" peuvent modifier la rotation. Il faut donc enregistrer le sens de rotation courant et le modifier selon la carte jouée.

3.1 Gestion des joueurs

Il faut une structure de données pour enregistrer les mains de tous les joueurs. L'utilisateur doit aussi pouvoir saisir le nombre de joueurs au départ, et le nombre de cartes dans la main de départ.

On propose d'utiliser un dictionnaire pour stocker pour chaque joueur son nom et le contenu de sa main.

Écrire une fonction d'initialisation en début de partie, qui demande à l'utilisateur le nombre de joueur, lui demande le nom de chaque joueur, lui demande le nombre de cartes dans les mains de départ, puis initialise les mains de chaque joueur avec le bon nombre de cartes et les enregistre dans le dictionnaire.

3.2 Sens de rotation et prochain joueur

Écrire une fonction qui détermine le numéro ou le nom du joueur dont c'est le tour à partir du numéro ou du nom du joueur précédent, du sens de rotation courant, et de la dernière carte jouée.

Il faudra donc gérer les effets des cartes spéciales :

- si la dernière carte est un changement de sens, il faut mémoriser le nouveau sens de rotation avant de passer au joueur suivant dans ce sens modifié
- si la dernière carte est un passe-tour, il faut sauter le joueur suivant et passer à celui d'après

3.3 Test de victoire et fin de partie

Un joueur gagne quand il n'a plus aucune carte en main. Écrire une fonction qui teste si le joueur courant (reçu en argument) a gagné, et renvoie un booléen.

3.4 Tour de jeu

Écrire une fonction qui reçoit en particulier le numéro du joueur dont c'est le tour, et réalise les étapes suivantes qui constituent son tour :

- teste s'il peut jouer ou combien de cartes il doit piocher
- s'il doit piocher : vérifie s'il reste assez de cartes dans la pioche, sinon y mélange la pile de jeu, avant de le faire piocher
- s'il peut jouer : lui demande quelle carte il veut jouer, et la joue (et applique éventuellement les effets de la carte)
- vérifie si le joueur a gagné ou détermine le numéro du prochain joueur, et renvoie cette information

3.5 Programme principal

Écrire un programme principal qui gère une partie complète jusqu'à ce qu'un joueur ait gagné, et affiche le nom du gagnant et les mains restantes des autres joueurs.

4 Partie D - interface graphique et bonus

En utilisant par exemple le module turtle, fournir une interface graphique qui affiche entre autres :

- la carte centrale sur laquelle on joue
- le joueur courant et sa main
- le sens de rotation
- des instructions éventuelles (tel joueur pioche tant de cartes)
- des indications pour aider le joueur (quelles cartes il peut jouer)

Pour aller plus loin on peut aussi :

- permettre au joueur de jouer en cliquant dans l'interface plutôt qu'en tapant dans la console
- écrire une IA qui joue toute seule, et proposer un menu permettant de définir le nombre de joueurs, de demander leur nom, et de dire lesquels sont joués par l'IA ou par un joueur humain.

Mémo Python - UE INF101 / INF131 / INF204

Opérations sur les types

`type()` : pour connaître le type d'une variable
`int()` : transformation en entier
`float()` : transformation en flottant
`str()` : transformation en chaîne de caractères

Définition d'une fonction

```
def nomFonction(arg) :
    instructions
    return v
```

Fonction qui renvoie la valeur ou variable v.

Infini

`float('inf')` : valeur infinie positive ($+\infty$)
`float('-inf')` : valeur infinie négative ($-\infty$)

Écriture dans la console

`print(a1,a2,...,an, sep=xx, end=yy)`

- Pour imprimer une suite d'arguments de a1 à an
- sep : permet de définir le séparateur affiché entre chaque argument (optionnel, par défaut " ")
- end : permet de définir ce qui sera affiché à la fin (optionnel, par défaut : saut de ligne)

Lecture dans la console

`res = input(message)`

- Pour lire une suite de caractères au clavier terminée par $< Enter >$
- Ne pas oublier de transformer la chaîne en entier (int) ou réel (float) si nécessaire.
- La chaîne de caractères résultante doit être affectée (ici à la variable `res`).
- L'argument est optionnel : c'est un message explicatif destiné à l'utilisateur

Opérateurs booléens

`and` : et logique
`or` : ou logique
`not` : négation

Opérateurs de comparaison

```
== égalité              != différence
<  inférieur,           <= inférieur ou égal
>  supérieur,           >= supérieur ou égal
```

Instructions conditionnelles

```
if condition :
    instructions

if condition :
    instructions
else :
    instructions

if condition1 :
    instructions
elif condition2 :
    instructions
else :
    instructions
```

Opérateurs arithmétiques

```
+ : addition,          - : soustraction
* : multiplication,    ** : puissance,
/ : division,          // : quotient div entière,
% : reste de la division entière (modulo)
```

Caractères

`ord(c)` : renvoie le code ASCII du caractère c
`chr(a)` : renvoie le caractère de code ASCII a

Chaînes de caractères

`len(s)` : renvoie la longueur de la chaîne s
`s1+s2` : concatène les chaînes s1 et s2
`s*n` : construit la répétition de n fois la chaîne s
exemple : "ta"* 3 donne "tatata"
`list(chaine)` : renvoie la liste des caractères de la chaîne
`ch.split(arg)` : retourne la liste des sous-chaînes de ch, en coupant à chaque occurrence de `arg` (par défaut `arg=" "`)
`ch.join(liste)` : concatène les chaînes de `liste`, en utilisant `ch` comme séparateur, et renvoie la chaîne résultante
`ch.upper()`: passe `ch` en majuscules
`ch.lower()`: passe `ch` en minuscules

Itération tant que

```
while condition :
    instructions
```

Itération for, et range

```
for e in conteneur :
     instructions

for var in range (deb, fin, pas) :
     instructions
```

Itère les instructions avec e prenant chaque valeur dans le conteneur (liste, chaîne ou dictionnaire) ; ou avec var prenant les valeurs entre `deb` et `fin` avec un `pas` donné.

`range(a)` : séquence des valeurs [0, a[
`range (b,c)` : séquence des valeurs [b, c[(pas=1, $c > b$)
`range (b, c, g)` : idem avec un pas = g
`range(b,c,-1)`: valeurs décroissantes de b (incl.) à c (excl.), pas=-1 ($c < b$)

Listes

`maListe = []`: création d'une liste vide
`maListe = [e1,e2,e3]` : création d'une liste, ici à 3 éléments e1, e2, et e3

`maListe[i]`: obtenir l'élément à l'index i ($i >= 0$).
Les éléments sont indexés à partir de 0. Si $i < 0$, les éléments sont accédés à partir de la fin de la liste. Ex : `maListe[-1]` permet d'accéder au dernier élément de la liste

`maListe.append(elem)`: ajoute un élément à la fin
`maListe.extend(liste2)`: ajout de tous les élements de la liste `liste2` à la fin de la liste `maListe`
`maListe.insert(i,elem)`: ajout d'un élément à l'index i

`res = maListe.pop(index)`: retire l'élément présent à la position `index` et le renvoie, ici dans la variable `res`
`maListe.remove(element)`: retire l'élément donné (le premier trouvé)

`len(maListe)`: nombre d'éléments d'une liste
`elem in maListe`: teste si un élément est dans une liste (renvoie True ou False)

`l2 = maListe`: crée un synonyme (2ème nom pour la liste)
`l3 = list(maListe)`: crée une copie de surface (un clone)
`l4 = copy.deepcopy(maListe)`: crée une copie profonde (récursive)

Aléatoire

`random.randint(inf,sup)`: entier aléatoire entre bornes inf et sup incluses
`random.shuffle(maListe)`: mélange la liste (effet de bord), ne renvoie rien
`random.choice(maListe)`: renvoie un élément au hasard de la liste

Dictionnaires

`monDico = {}` : création d'un dictionnaire vide
`monDico = { c1:v1, c2:v2, c3:v3 }` : création d'un dictionnaire, ici à 3 entrées (clé c1 avec valeur v1, etc)

`e = monDico[c1]` : les valeurs du dictionnaire sont accessibles par leurs clés. Ici, e prendra la valeur v1. Provoque une erreur si la clé n'existe pas.

`monDico[c3] = v3`: ajoute une nouvelle valeur au dictionnaire (ici v3) avec une clé (ici c3). Si la clé existe déjà, la valeur associée est modifiée.

`del monDico[C3]`: supprime une association dans le dictionnaire. La clé doit exister.

`c in monDico`: vérifie l'existence d'une clé dans le dictionnaire, renvoie True ou False.

`dic2 = monDico`: crée un synonyme (2ème nom au dico)
`dic3 = dict(monDico)`: crée une copie de surface (clone)
`dic4 = copy.deepcopy(monDico)`: crée une copie profonde (récursive)

Gestion des fichiers

`f=open('data.txt')`: ouvrir un fichier en lecture seule
`f=open('data.txt','w')`: ouvre un fichier en écriture (attention s'il existe il est écrasé, sinon il est créé)
`f=open('data.txt','a')`: ouvre un fichier en écriture (ajoute le texte à la fin)

`texte = f.read()`: lire tout le fichier en une seule fois
`lignes = f.readlines()`: lire en 1 fois toutes les lignes du fichier et les stocker dans une liste (un élém=une ligne)

```
for ligne in f:
    instructions
```

Lire le fichier ligne par ligne dans une boucle for

`f.write(texte)`: écrire dans un fichier (`texte` doit obligatoirement être une `string`).
Ne saute pas de ligne automatiquement à la fin du texte. `'\n'` code un saut de ligne.

`f.close()`: ferme un fichier