Cahier d'exercices[1] - Module Python

Licence 1 – Année 2022/2023

G. Simon

Table des matières

1NB : certains exercices sont issus ou adaptés d'un cahier d'exercices du module de Python à l'ISEL rédigé par M. De Boysson ainsi que du site « Exercices pratiques en NSI ».

Partie 1: Remise en route sur les bases

Exercice 1.1 : calcul d'une valeur approchée de PI

Sachant que la somme : 1 – 1/3 + 1/5 – 1/7 + 1/9 – 1/11 + 1/13 – 1/15 + … tend vers π/4, écrire un programme permettant de calculer une valeur approchée de π. On pourra écrire deux versions : selon que l'on demande à l'utilisateur le nombre de termes souhaités, ou la valeur du dénominateur du dernier terme pris en compte dans le calcul.

Exercice 1.2 : calcul des termes de la suite de Fibonacci

Écrire un algorithme, puis un programme, qui permet d'afficher le n ième terme (n étant saisi au clavier) de la suite de fibonnaci : 1 1 2 3 5 8 13 21…
Rappel : la définition mathématique de la suite de Fibonnaci est la suivante :

- ✗ $U_0 = 1$
- ✗ $U_1 = 1$
- ✗ $U_n = U_{n-1} + U_{n-2}$ pour n>1

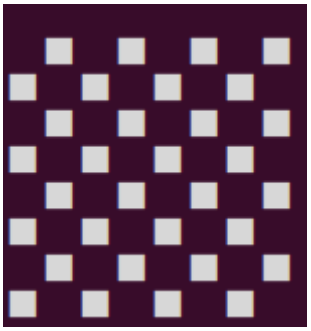
Exercice 1.3 : calcul des tables de multiplication

Écrire un programme, permettant d'afficher à l'écran la table de multiplication des entiers de 1 à 10. On n'utilisera ici aucune structure de données. L'affichage doit être le suivant :

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Exercice 1.4 : le damier

Écrire un programme qui affiche à l'écran un damier. La taille du coté du damier sera demandée à l'utilisateur. Pour les cases noires, on affichera un espace. Pour les cases blanches, on affichera le caractère correspondant à un carré blanc (taper au clavier dans un terminal : CTRL+SHIFT+u2410 pour obtenir le caractère) .



Exercice 1.5 : le distributeur automatique

Un distributeur automatique dispose, pour rendre la monnaie, de 20 billets de 5 euros et de 100 pièces de 1 euro. Ecrire un programme, qui :

- demande à l'utilisateur le prix de l'article à payer (en euros entiers) et la somme qu'il va introduire dans le distributeur ;
- calcule la monnaie à rendre à l'utilisateur en billets de 5 euros et pièces de 1 euro ;
- déduit de ses propres ressources le nombre de billets et pièces rendues ;
- propose un nouvel achat jusqu'à ce qu'il n'y ait plus de monnaie à rendre.

Indication : la caisse contenant la monnaie à rendre et celle recevant l'argent des clients sont distinctes. Il arrivera donc un moment où il ne sera plus possible d'effectuer un achat.

```
Montant restant :  200
Prix de l'article : 50
Montant fourni : 75
Je vous rends la monnaie : 25 euros
5 billets de 5 euros et 0 pieces de 1 euro
Montant restant :  175
Prix de l'article : 20
Montant fourni : 200
Je n'ai pas assez de monnaie !
Montant restant :  175
Prix de l'article : 20
Montant fourni : 195
Je vous rends la monnaie : 175 euros
15 billets de 5 euros et 100 pieces de 1 euro
```

Exercice 1.6 : jeu du nombre à deviner

Ecrire un programme permettant de jouer au jeu du nombre mystérieux. La règle en est la suivante : après que la machine ait choisi un nombre aléatoire compris entre 1 et 1000, vous devez, en proposant des nombres, retrouver le nombre de la machine. Pour cela à chaque fois que vous proposez un nombre, la machine vous répond si votre nombre est plus grand ou plus petit que le nombre à trouver. Quand vous aurez trouvé le nombre mystérieux, la machine devra préciser en combien de coups vous avez réussi.

Exercice 1.7 : jeu du nombre à deviner inversé

Inverser le principe de l'exercice précédent en faisant trouver par l'ordinateur un nombre choisi par l'utilisateur. Pour cela, le programme va tirer aléatoirement un nombre dans un intervalle qui sera réduit à chaque tentative en fonction des tirages précédents. Par exemple, supposons que le nombre à trouver est 35. Le programme va commencer par tirer aléatoirement un nombre entre 1 et 1000. Supposons qu'il choisisse 50. L'utilisateur va indiquer que le nombre proposé est trop grand. Le prochain tirage du programme se fera donc entre 1 et 50...
A chaque tentative de la machine, on affichera le nombre qu'elle propose ainsi que le nombre de tentatives déjà effectuées.

Exercice 1.8 : propriétés des nombres

On recherche les nombres à quatre chiffres ' abcd' tels que, si l'on décompose ces nombres, ils vérifient la propriété suivante : ab + cd = bc.
Par exemple, 3956 vérifie cette propriété. En effet, 39 + 56 = 95.
Ecrire un programme qui affiche tous les nombres vérifiant cette propriété combien il y en a.
Remarque : pour ce type de problème,on peut effectuer une recherche exhaustive sur l'intervalle des nombres à quatre chiffres.

Exercice 1.9 : nombres à construire

Soit l'équation aa * bb = ccdd avec chaque lettre représentant un chiffre
Ecrire un programme qui doit trouver les nombres correspondant à ccdd.
Pour chacun, le programme doit fournir la multiplication de la forme aa * bb dont il est issu.

- dans une première version, les 4 chiffres a,b,c et d sont quelconques
- dans une seconde version, les 4 chiffres a, b, c et d doivent être tous différents

Exercice 1.10 : contrôle de saisie

Il s'agit ici d'écrire un programme permettant de faire saisir à l'utilisateur consécutivement 4 chiffres. Tout ce qui n'est pas un chiffre doit être refusé. Dès qu'un chiffre est validé, il doit être affiché et le programme doit passer à la saisie du chiffre suivant. Une fois les 4 chiffres saisis, le programme doit afficher le nombre correspondant aux 4 chiffres saisis ainsi que son double.
Rappels:
- la fonction chr renvoie le caractère dont le code ASCII est passé en paramètre.
- la fonction ord renvoie le code ASCII du caractère passé en paramètre
NB : il faut aussi prévoir et refuser le cas ou l'utilisateur saisit plusieurs caractères au lieu d'un seul.

Exercice 1.11 : code Hamming

Il s'agit d'écrire un programme permettant de calculer la distance de Hamming entre deux mots. Cette notion est utilisée dans de nombreux domaines comme les télécommunications, le traitement du signal, . . . Elle est définie, pour deux mots de même longueur, comme le nombre de positions où les deux mots ont un caractère différent. Le programme doit saisir 2 mots et calcule la distance de Hamming entre les deux mots lorsqu'ils ont la même longueur. Si les mots n'ont pas la même longueur, le programme doit afficher -1.
Par exemple, pour les mots « aaba » et « aaha », le programme doit afficher 1. Pour les mots « poire » et « pomme », le programme doit afficher 2. Enfin pour les mots « stylo » et « bouteille », le programme doit afficher -1.

Partie 2: Les fonctions

Exercice 2.1 : retour sur les tables de multiplications

Reprendre l'exercice sur le calcul des tables de multiplication de la partie 1 et introduire une fonction qui affiche la table de multiplication par un nombre passé en paramètre.

Exercice 2.2 : somme des nombres dans un intervalle

Il s'agit d'écrire un programme permettant à l'utilisateur de saisir 2 entiers (le second devant être strictement supérieur au premier) et d'afficher la somme des nombres situés dans l'intervalle fermé définir par les 2 entiers saisis. On définira les fonctions suivantes :

- `saisirEntier()` permettant de saisir quelque chose, de vérifier que ce qui a été saisi correspond bien à un entier, de le renvoyer en résultat si c'est le cas ou de recommencer la saisie sinon.
- `saisieEntierSuivant(premier)` prenant en paramètre le premier entier saisi et permettant de saisir le second entier, de vérifier qu'il s'agit bien d'un entier, de vérifier qu'il est bien supérieur à premier et de le renvoyer en résultat si c'est le cas ou de recommencer la saisie sinon.
- `somme(premier,second)` permettant de calculer et de renvoyer la somme des entiers situés dans l'intervalle `[premier,second]`.

Exercice 2.3 : les anagrammes

Deux chaînes de caractères **de même longueur** sont des anagrammes s'il est possible d'écrire l'une en utilisant tous les caractères de l'autre, quitte à les déplacer.Par exemple les chaines `chien` et `niche` sont des anagrammes, alors que `louve` et `poule` ne le sont pas.

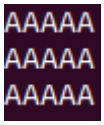
Ecrire un programme permettant de saisir deux chaînes de caractères et de déterminer si ce sont des anagrammes. On introduira une fonction analyser(ch1, ch2) prenant en paramètre 2 chaînes de caractères en renvoyant `True` si les 2 chaînes sont des anagrammes et `False` sinon.

NB : On peut trier les 2 chaînes de caractères avant de les comparer.

Exercice 2.4 : forme géométriques ASCII

L'objectif est de dessiner des formes géométriques (rectangle et triangle) pleines ou creuses en utilisant un caractère. Le caractère et la taille sont choisis par l'utilisateur. Ces formes sont dessinées ligne par ligne. Voici un exemple de chaque forme visée :

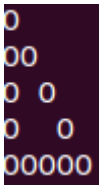
- rectangle plein : caractère « A », longueur 5, largeur 3



- rectangle creux : caractère « # », longueur 5, largeur 4

- triangle plein : caractère « #», hauteur 5

- triangle creux, caractère « 0 », hauteur 5



Pour cela, on introduira les fonctions suivantes :

- `ligne_pleine(car,nb)` : r**envoie une chaine de caractères** correspondant à une ligne pleine de nb caractères de type car. Attention : nb doit être supérieur ou égal à 1.

- `ligne_creuse(car,nb)` : r**envoie une chaine de caractères** correspondant à une ligne creuse de longueur nb faite avec le caractère car. Là encore nb doit être supérieur ou égal à 1.

- `rectangle_plein(car,longueur,largeur)` : affiche à l'écran un rectangle plein fait de caractères car avec la longueur et la largeur passés en paramètres.

- `rectangle_creux(car,longueur,largeur)` : affiche à l'écran un rectangle creux fait de caractères car avec la longueur et la largeur passés en paramètres.

- `triangle_plein(car,hauteur)` : affiche à l'écran un triangle plein fait de caractères car dont la hauteur est celle passée en paramètre.

- `triangle_creux(car,hauteur)` : affiche à l'écran un triangle creux fait de caractères car dont la hauteur est celle passé en paramètre.

Exercice 2.5 : correspondances de mots

Dans cet exercice :

- un mot est une chaîne de caractères composée uniquement de lettres de l'alphabet.
- Un mot à trous comporte également zéro, une ou plusieurs fois le caractère `"."`.

On dira que `mot_complet` correspond à `mot_a_trous`, si on peut remplacer chaque `"."` de `mot_a_trous` par une lettre de façon à obtenir `mot_complet`.

Exemples

- `"INFO.MA.IQUE"` est un mot à trous,
- `"INFORMATIQUE"` est un mot qui lui correspond,
- `"AUTOMATIQUE"` est un mot qui ne lui correspond pas.

Ecrire un programme permettant de saisir un mot complet et un mot à trous et de déterminer si les 2 mots peuvent être mis en correspondance. On définira pour cela une fonction `correspondance(mot_complet,mot_a_trous)` renvoyant vrai si les 2 mots peuvent être mis en correspondance et faux sinon.

Exercice 2.6 : fonction à nombre variable de paramètres

Définir une fonction à nombre variables de paramètres ayant le comportement suivant :
- si aucun paramètre n'est passé, la fonction doit renvoyer la chaîne « etcetc »
- si la fonction reçoit un paramètre, la fonction doit renvoyer une chaîne de caractères composée de la chaîne de caractères passée en paramètre écrite deux fois à suivre
- si la fonction a au moins deux paramètres, la fonction doit renvoyer une chaîne de caractères composée de la suite des différentes chaînes de caractères passées en paramètres.

Partie 3: Les tableaux / listes

Exercice 3.1 : calcul du maximum d'une liste de nombres

Il s'agit d'écrire un programme permettant de recherche dans une liste d'entiers la plus grande valeur. La liste pourra être définie au début du programme ou saisie par l'utilisateur ou encore tirée aléatoirement. Le programme commencera par afficher cette liste. Plusieurs versions de la recherche devront être développées :
- trouver la valeur maximale de la liste
- trouver l'indice de la dernière occurrence de la valeur maximale de la liste
- trouver l'indice de la première occurrence de la valeur maximal de la liste
- trouver toutes les occurrences de la valeur maximal de la liste et pour chacune d'elles son indice
Pour chacune des versions, on introduira une fonction prenant en charge la recherche associée.

Exercice 3.2 : séparer les nombre pairs et impairs dans une liste

Ecrire un programme permettant à l'utilisateur de saisir les éléments d'une liste d'entiers L1, puis de créer à partir de L1 2 autres listes L2 et L3: L2 doit contenir tous les éléments pairs de L1 et L3 doit contenir tous les éléments impairs de L1. Enfin, le programme doit afficher L1 puis L2 et L3.

Exercice 3.3 : insertion dans une liste triée

Ecrire un programme reposant sur une fonction permettant d'insérer un entier dans une liste d'entiers déjà triée. L'objectif est qu'apès l'insertion la liste soit toujours triée.

Exemples :

- si on insère 7 dans la liste [2,5,7,10], on doit obtenir [2,5,7,7,10]

- si on insère 7 dans [5,8,12,20], on doit obtenir [5,7,8,12,20]

Exercice 3.4 : fusion de listes triées

Ecrire un programme reposant sur une fonction permettant de fusionner 2 listes déjà triées. Il s'agit d'obtenir une troisième liste qui doit elle aussi être triée et contenir tous les éléments des 2 listes initiales.

Exemples :

- si on fusionne [1,4,5] et [2,3,7], on doit obtenir la liste [1,2,3,4,5,7]

- si on fusionne [1,4,5] et [], on doit obtenir [1,4,5]

- si on fusionne [] et [1,4,5], on doit obtenir [1,4,5]

Exercice 3.5 : première et dernière occurrences dans une liste

Ecrire un programme reposant sur une fonction permettant de trouver l'indice de la première occurrence et de la dernière occurrence d'un élément dans une liste. La fonction prend en paramètre

l’élément et la liste dans laquelle recherche l’élément et renvoie un tableau à 2 éléments contenant l ‘indice de la première et de la dernière occurrence de l’élément dans la liste. Voici quelques exemples :

- premEtDern(2,[1,2,3,2,4,2]) doit renvoyer [1,5]

- premEtDern(3,[1,2,3,4]) doit renvoyer [2,2]

- premEtDern(5,[1,2,3,4]) doit renvoyer [-1,-1]

Exercice 3.6 : enlever les redondances dans une liste

Ecrire un programme permettant d’enlever dans une liste d’entiers les éléments présents plus d’une fois. On introduira une fonction prenant en paramètre le tableau initial **trié** et renvoyant un nouveau tableau dans lequel chaque élément n’est présent qu’une fois.

Exercice 3.7 : le dentiste

Chez le dentiste, la bouche grande ouverte, lorsqu'on essaie de parler, il ne reste que les voyelles. Même les ponctuations sont supprimées. Vous devez écrire une fonction `dentiste` qui prend une chaine de caractères `texte` et qui renvoie une autre chaine ne contenant que les voyelles de `texte`, placées dans l'ordre. Pour cela, on déclarera un tableau dans lequel seront stockées les différentes voyelles de l’alphabet français. Le programme devra permettre de saisir une chaine de caractères qu’on supposera en minuscules et sans accent et devra afficher la prononciation de cette chaîne façon dentiste.
Exemple : « il fait chaud » donnera « iaiau ».

Exercice 3.8 : décomposition en facteurs premiers

Il s’agit d’écrire un programme permettant de saisie un entier positif, de calculer sa décomposition en facteurs premiers et enfin de l’afficher. Si l’entier saisi est un nombre premier, le programme doit le signaler. Par exemple, si on saisit 24, le programme doit afficher en résultat 2*2*2*3.
Pour cela, on introduira les fonctions suivantes :
- divPar2(nb) : cette fonction prend le nombre choisi en paramètre et divisant autant que possible ce nombre par 2 jusqu’à obtenir un nombre impair. Cette fonction doit renvoyer un tableau dont la première valeur doit être le nombre impair obtenu et la seconde valeur doit être le nombre de divisions par 2 qui ont été effectuées
- divParImpair(tab) : cette fonction prend en paramètre un tableau calculé par la fonction précédente . Elle doit diviser le nombre impair stocké comme premier élément de tab par les nombres impairs successifs. La fonction doit diviser autant que possible par chaque nombre impair puis passer au nombre impair suivant.Le processus s’arrête lorsque le reste est inférieur ou égal à 1. La fonction doit renvoyer un tableau dont le premier élément est le tout dernier reste et dont les éléments suivants sont les différents diviseurs utilisés. Si un diviseur a été utilisé plusieurs fois, il doit alos figurer plusieurs fois dans le tableau renvoyé
- affichage(n,tab) : cette fonction prend en paramètre le nombre à décomposer et le tableau renvoyé par la fonction divParImpair. Elle doit afficher la décomposition en nombres premiers du nombre n sous la forme d’une multiplication. Si le nombre n est premier alors la fonction doit faire un affichage le signalant.

Exercice 3.9 : écrêtage des valeurs d'un tableau

L'écrêtage d'un signal consiste à limiter l'amplitude du signal entre deux valeurs `x_min` et `x_max`. On peut également appliquer cela aux valeurs d'un tableau. Voici par exemple un tableau `valeurs` que l'on a écrêté entre -150 et 150 pour donner le tableau `valeurs_ecretees` :

```
valeurs_depart =   [34, 56, 89, 134, 152, 250, 87, -34, -187, -310]
valeurs_ecretees = [34, 56, 89, 134, 150, 150, 87, -34, -150, -150]
```

On définira les fonctions suivantes :
- la fonction limite_amplitude(val,x_min,x_max) qui limite la valeur val à l'intervalle [x_min,x_max] et renvoie la valeur résultante.
- la fonction ecrete(tab,x_min,x_max) qui limite chaque valeur du tableau tab à l'intervalle [x_min,x_max]. Elle renvoie le tableau résultant.

Exercice 3.10 : inclusion de texte

L'ADN peut être représenté par une chaine de caractères formée avec les lettres `A`, `T`, `G`, `C`.

- Un **brin** est un petit morceau d'ADN, que l'on retrouve parfois dans
- un **gène** qui est une grande séquence d'ADN.

On veut écrire un programme permettant de savoir si un brin d'ADN est inclus ou non dans un gène, les 2 étant saisis sous forme de chaines de caractères par l'utilisateur. Pour cela, on définira deux fonctions :

- la fonction `correspond(motif, chaine, position)` qui renvoie `True` si on retrouve `motif` exactement à partir de `position` dans `chaine` et `False` sinon.

- la fonction `est_inclus` prend en paramètres deux chaines de caractères `brin` et `gene` et renvoie vrai si le brin est inclus dans le gène.

Exemples :

- le brin « GT » est inclus dans le gène « AAGGTTCC »

le brin « AGA » n'est pas inclus dans le gène « AAGGTTCC »

NB : dans cet exercice, on n'utilisera pas de fonction prédéfnie de traitement de chaines.

Exercice 3.11 : recherche d'un élément par dichotomie dans un tableau trié

On veut écrire un programme qui fait une recherche d'un élément par dichotomie dans un tableau d'entiers trié. On va pour cela introduire une fonction **récursive** (ie. qui s'appelle elle-même) rechercheDichotomie qui prend en paramètres l'élément à rechercher (elem), le tableau tab dans lequel rechercher, l'indice de début de recherche (deb) et l'indice de fin de recherche (fin) et qui doit renvoyer True si l'élément fait bien partie du tableau. Lors du premier appel de lafonction, deb est égal à 0 et fin est égal à l'indice du dernier élément du tableau. Le principe de cette fonction est le suivant :

- si l ‘élément se trouve à l’indice milieu, renvoyer True
- si deb > fin alors l’élément ne figure pas dans le tableau, renvoyer False
- si deb = fin alors renvoyer True si l’élément recherché se trouve dans tab[deb]
- si deb<fin alors calculer l’indice du milieu ((deb+fin)//2) et relancer la recherche récursivement entre les indices deb et milieu et, si l’élément n’a pas été trouvé, relancer la recherche récursivement entre les indices milieu et fin.

Exercice 3.12 : le triangle de Pascal

L’objectif est d’écrire un programme capable d’afficher le triangle de Pascal d’une certaine taille. L’utilisateur saisit au clavier le nombre de lignes voulu pour le triangle. Le triangle de Pascal sera représenté par un tableau de tableaux (chaque sous-tableau représente une ligne du triangle de Pascal). On s’appuiera sur 2 fonctions :

- construireTabCoeff(tab,n) : cette fonction prend en paramètre un tableau de tableaux dont les 2 premières lignes doivent avoir été initialisées au préalable. La fonction complète les lignes du tableaux avec les lignes du triangle de Pascal jusqu’à arriver à un nombre de ligne égal à n

- affichage(tab) : cette fonction prend en paramètre un tableau de tableaux représentant un triangle de Pascal et doit l’afficher à l’écran.

Exemple : voici l’affichage que l’on doit obtenir pour un triangle de Pascal de 5 lignes

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
```

Exercice 3.13 : jeu du Mastermind

Il s’agit de faire deviner à l’utilisateur une combinaison de 6 couleurs Bleu, Rouge, Vert, Jaune, Marron et Gris. Chaque couleur sera désignée dans les combinaisons par son initiale. Le programme commencera par choisir au hasard une combinaison de ces 6 couleurs (une même couleur peut être présente plusieurs fois). Par exemple le programme pourra choisir la combinaison BRGJRB. Le but est de deviner la combinaison en proposant des conbinaisons. A chaque étape le programme lui indiquera le nombre de couleurs bien placées et le nombre de couleurs mal placées. Par exemple, pour le cas précédent, si l’utilisateur propose RBBGMB, le programme devra lui répondre « 1 couleur bien placée et 3 mal placées ». Lorsque l’utilisateur trouve la combinaison, le jeu s’arrête et le programme doit afficher le nombre d’essais effectués par l’utilisateur.

Dans le programme, les couleurs disponibles seront stockées dans un tableau sous forme de chaines de caractères contenant leur initiale. Une combinaison sera saisie sous forme d’une chaîne de caractères. On écrira les fonctions suivantes :

- une fonction creer_combinaison qui renvoie sous forme d'un tableau de caractères la combinaison choisie. A chaque étape, la couleur choisie est tirée au hasard dans le tableau des couleurs
- une fonction evaluer_proposition prenant en paramètre une proposition de l'utilisateur sous forme de chaine de caractères et renvoyant en résultat un tuple dont le premier élément est le nombre de couleurs bien placées et dont le second élément est le nombre de couleurs mal placées
- une fonction lire_proposition qui saisie une proposition de l'utilisateur puis vérifie qu'elle ne contient bien que des initiales de couleurs. La fonction doit ré-interroger l'utilisateur tant que la saisie n'est pas correcte. Lorsque la saisie est correcte, elle doit être renvoyée en résultat de la fonction.
- Une fonction jouer() qui gère le déroulement du jeu

Partie 4: Les tuples

Exercice 4.1 : construction d'un tuple

Ecrire un programme qui permet à l'utilisateur de saisir un entier n ≥ 10 (avec contrôle de la saisie) et qui fait appel à une fonction prenant en paramètre l'entier n et renvoyant en résultat un tuple composé du nombre correspondant à la dizaine précédant n puis du nombre correspondant à la dizaine juste après n. Le programme doit ensuite afficher ce tuple

Exemple : pour n = 32, la fonction doit renvoyer le tuple (30,40)

Exercice 4.2 : chercher un élément dans un tuple

Ecrire un programme définissant et utilisant une fonction prenant en paramètre un entier et un tuple d'entiers. Cette fonction doit renvoyer l'indice de la première occurrence de l'élément dans le tuple. Si l'élément n'appartient pas au tuple, la fonction doit renvoyer -1. Le programme doit afficher l'élément, le tuple et le résultat de la fonction.

Exercice 4.3 : créer un tuple à partir d'un autre

Ecrire un programme permettant de saisir une suite d'entiers, puis de construire un tuple contenant cette suite d'entiers. La saisie s'arrêtera si l'utilisateur saisit un nombre négatif. On définira une fonction qui prend en paramètre une liste des entiers saisis et qui renvoie en résultat un tuple contenant la suite de nombres de la liste. Le programme doit afficher le tuple obtenu.

Exercice 4.4 : simuler une collection de jeux

Dans cet exercice, on veut pouvoir représenter une collection de jeux. Chaque jeu va être représenté par un tuple composé du nom du jeu (chaîne de caractères), le nombre minimal de joueurs et le nombre maximal de joueurs. La collection va être représentée par une liste de tuples.

Définir une fonction prenant en paramètre un nombre de joueurs et qui renvoie en résultat la liste des noms des jeux auxquels il est possible de jouer avec ce nombre de joueurs. Dans le programme, on commencera par définir la collection de jeux, puis on fera saisir un nombre de joueurs à l'utilisateur et enfin on affichera la liste des jeux auxquels il est possible de jouer.

Exercice 4.5 : tuples et chaîne de caractères

Ecrire un programme permettant à l'utilisateur de saisir une chaîne de caractères et, en utilisant une fonction, de calculer un tableau de tuples permettant de recenser les caractères présents dans la chaine de caractères. Plus exactement, pour chaque caractère présent dans la chaîne de caractères, le tableau devra contenir un tuple dont le premier élément est le caractère et le deuxième élément est le nombre de fois où le caractère est présent dans la chaîne. Le programme doit afficher ce tableau.

Exemple : pour la chaîne de caractères « bonjour jules », le tableau affiché doit être :

```
[('b',1),('o',2),('n',1),('j',2),('u',2),('r',1),(' ',1),('l',1),('e',1),('s',1)]
```

Exercice 4.6 : jeux de cartes

Il s'agit ici d'utiliser des tuples et des listes pour représenter des mains de cartes dans le cadre d'un jeu de carte. Chaque carte va être représentée par un tuple de deux éléments dont le premier élément est le rang de la carte (de 1 à 10) et le second élément est la couleur ('trefle','pique','coeur' ou 'carreau'). On va représenter l'ensemble des cartes disponibles par un tableau de tuples tjeu. Puis le programme va demander le nom des chacun des 2 joueurs. Il va ensuite choisir aléatoirement des cartes dans tjeu pour constituer le jeu de 5 cartes de chacun des joueurs. Le principe doit être de tirer une carte pour chacun des jeux alternativement. Puis le programme doit demander une couleur de carte à l'utilisateur. Enfin, il doit afficher le nom du joueur ayant la plus forte carte dans cette couleur. Voici un exemple d'affichage du programme :

```
nom du joueur1: titou
nom du joueur2: yann
jeu joueur1: [(3, 'carreau'), (8, 'coeur'), (5, 'trèfle'), (4, 'pique'), (2, 'trèfle')]
jeu joueur2: [(12, 'trèfle'), (1, 'coeur'), (7, 'trèfle'), (8, 'trèfle'), (1, 'carreau')]
38
Choisir un symbole:
trèfle
yann a la plus grande carte : 12 trèfle
```

Partie 5: les ensembles

Exercice 5.1 : ensembles de couleurs d'articles

On se place dans le cadre de deux magasins vendant le même type d'article. On suppose que l'on mémorise dans 2 ensembles coul1 et coul2 les couleurs d'articles disponibles dans chaque magasin. Chaque couleur est représentée par une chaine de caractères. Définir les fonctions suivantes :

- une fonction `afficherCouleurs` prenant en paramètre un ensemble de couleurs et permettant d'afficher les couleurs de l'ensemble.

- une fonction `dispo` prenant une couleur et un ensemble de couleurs en paramètre et permettant de savoir si la couleur appartient à l'ensemble de couleurs ou pas.

- une fonction enCommun prenant 2 ensembles de couleurs en paramètres et renvoyant l'ensemble des couleurs communes aux 2 ensembles.

- une fonction `ajouter` prenant en paramètres une couleur et un ensemble de couleurs coul et permettant d'ajouter la couleur à l'ensemble de couleurs coul

- une fonction `collection` prenant en paramètre deux ensembles de couleurs et renvoyant en résultat l'ensemble résultant de l'union des deux ensembles

- une fonction `diff` prenant en paramètres 2 ensembles de couleurs coul1 et coul2 et renvoyant l'ensemble des couleurs présentes dans coul1 mais pas dans coul2.

En utilisant les fonctions précédentes, le programme doit :

- afficher la liste des couleurs disponibles pour chaque magasins

- afficher si la couleur « vert » est disponible dans chacun des magasins

- afficher les couleurs disponibles dans les deux magasins (chaque couleur en un seul exemplaire)

- saisir une couleur d'un nouvel article et l'ajouter au stock du premier magasin. Afficher le nouveau stock

- afficher les couleurs disponibles dans le magasin 1 mais pas dans le magasin 2

Partie 6: les dictionnaires

Exercice 6.1 : gestion de likes

Sur le réseau social TipTop, on s'intéresse au nombre de « *like* » des abonnés. Les données sont stockées dans un dictionnaire où les clés sont les pseudos et les valeurs correspondantes sont les nombres de « *like* » comme ci-dessous :

```
{'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50}
```

Écrire une fonction `top_likes` qui :

- prend en paramètre un dictionnaire `likes` non-vide structuré comme dans l'exemple (un nombre de likes peut être nul),
- renvoie un tuple dont :
    - la première valeur est le pseudo ayant le plus de likes; en cas d'égalité sur plusieurs clés, on choisira la plus petite suivant un classement alphabétique,
    - la seconde valeur est le plus grand nombre de likes du dictionnaire.

Ecrire un programme qui définit un dictionnaire du type de celui de l'exemple et utilise la fonction top_likes pour afficher le pseudo ayant le plus de likes et son nombre de likes.

Exercice 6.2 : gestion d'un refuge d'animaux

On considère une base (une liste de dictionnaires Python) qui contient des enregistrements relatifs à des animaux hébergés dans un refuge.

Les propriétés déclarées pour chaque animal hébergé sont :

- `nom`,
- `espece`,
- `age`,
- `enclos`.

Les valeurs associées à `nom` et `espece` sont des chaines de caractères, celles associées à `age` et `enclos` des entiers.

Voici un exemple d'une telle liste :

```
animaux = [ {'nom': 'Medor', 'espece': 'chien', 'age': 5, 'enclos': 2},
            {'nom': 'Titine', 'espece': 'chat', 'age': 2, 'enclos': 5},
            {'nom': 'Tom', 'espece': 'chat', 'age': 7, 'enclos': 4},
            {'nom': 'Belle', 'espece': 'chien', 'age': 6, 'enclos': 3},
            {'nom': 'Mirza', 'espece': 'chat', 'age': 6, 'enclos': 5}]
```

On garantit que chaque enregistrement contient l'ensemble des informations (aucune clé ne manque dans un dictionnaire).

Ecrire une fonction `selection_enclos` qui :

- prend en paramètres :
  - une liste `table_animaux` contenant des enregistrements relatifs à des animaux (comme dans l'exemple ci-dessus),
  - un numéro d'enclos `num_enclos` ;
- renvoie une liste contenant les enregistrements de `table_animaux` correspondant aux animaux se trouvant dans l'enclos num_enclos. Dans cette liste, les enregistrements seront donnés dans le même ordre que dans la liste initiale.

Ecrire un programme qui déclare une base comme dans l'exemple, permet à l'utilisateur de saisir un numéro d'enclos et affiche la liste des noms des animaux se trouvant dans cet enclos. Si le numéro d'enclos ne figure pas dans la base, le programme doit afficher « numéro d'enclos inconnu ! ».

Exercice 6.3 : dictionnaire français-anglais

Écrire un programme :

- créant une structure de type dictionnaire "Dico", comportant quelques mots français associés à leur traduction en anglais ("français _ anglais"). Les clefs et les valeurs sont donc de type chaînes de caractères

- proposant à l'utilisateur d'ajouter une entrée dans le dictionnaire jusqu'à ce qu'il ne le souhaite plus (il stoppera la saisie avec le mot « fin »). Pour chaque nouvelle entrée saisie par l'utilisateur, le programme devra s'assurer qu'elle n'est pas déjà présente avant de l'ajouter.

- affichant le contenu du dictionnaire,

- affichant le contenu du dictionnaire par ordre alphabétique des entrées,

- créant un second dictionnaire "anglais - français" à partir du précédent et l'affichant,

-proposant à l'utilisateur de retirer une entrée de ce second dictionnaire (l'utilisateur saisie un mot anglais),

- affichant le contenu du dictionnaire "anglais-français".résultant

Exercice 6.4 : génération d'une page HTML

On veut générer le code d'une page HTML contenant un tableau à 2 colonnes. Voici la structure de base de la page HTML recherchée :

```
<html>
<head>
<title> Page générée en python </title>
</head>
<body>
<h1> Articles en stock </h1>
<table>
<tr>
      <th> Nom article </th>
      <th> Quantité
```

```
<tr>
...
</table>
</body>
</html>
```

On définira une fonction prenant un dictionnaire en paramètre et écrivant dans la console des lignes de la forme <tr> <td> nomArticle </td> <td> quantité en stock </td> </tr> pour chaque article figurant dans le dictionnaire. Les clefs du dictionnaire doivent être les noms des articles et lesvaleurs associées leur quantité en stock.

Le programme doit définir le dictionnaire puis afficher dans la console le code complet de la page HTML en utilisant la fonction précédente. On pourra tester la page HTML en copiant/collant le résultat du programme dans un éditeur puis en chargeant le fichier correspondant dans un navigateur.

Exercice 6.5 : analyser une chaîne de caractères

Le but est de ré-écrire l'exercice 35 en stockant cette fois le résultat dans un dictionnaire dont les clefs sont les différents caractères présents dans la chaîne de caractères saisie et les valeurs sont le nombre de fois où chaque caractère est présent dans la chaîne de caractères.

On introduira à cette fin une fonction `ajouter(dict, clef)` permettant d'ajouter au dictionnaire dict un couple (clef, frequence) dans le dictionnaire. Si un couple avec la même clef existe déjà dans le dictionnaire alors la fréquence sera l'ancienne augmentée de 1. Si aucun couple avec cette clef n'existe déjà dans le dictionnaire alors la fonction ajoutera un nouveau couple (clef,1).

Partie 7: lecture/écriture de fichiers

Exercice 7.1 : génération d'une page HTML

Repartir de l'exercice 41 et faire en sorte que le code de la page HTML soit écrit directement dans un fichier au lieu d'être écrit dans la console.

Exercice 7.2 : jeu du pendu

Le but de cet exercice est d'écrire un programme permettant de mettre en œuvre le jeu du pendu. On suppose que la liste des mots disponibles est définie dans un fichier texte « baseMots.txt » que vous pouvez récupérer sur Eureka. Ce fichier contient des lignes composées de mots séparés par des virgules. Tous les mots d'une même ligne ont le même nombre de caractères.

On définira les fonctions suivantes :

- `charger_mots` : fonction sans paramètre permettant de charger en mémoire les mots disponibles. Les mots seront stockées dans une liste de listes de mots. Chaque liste de mots contiendra des mots de même longueur. Les mots seront récupérés à partir du fichier « baseMots.txt ». La fonction doit renvoyer la liste de listes de mots

NB : ne pas oublier d'enlever les retour-chariots situés à la fin de chaque ligne du fichier.

- `jouer` : fonction prenant en paramètre le mot à rechercher et permettant de gérer une partie de jeu de pendu. L'utilisateur peut faire au maximum 4 erreurs. A chaque tentative, on affichera le nombre d'erreurs déjà faites ainsi que les lettres déjà proposées. Ensuite, on demandera une lettre à l'utilisateur. Si la lettre appartient bien au mot recherché, on affichera un mot contenant les lettres déjà proposées appartenant au mot recherché et un caractère « _ » pour chaque lettre non encore trouvée. Sinon, on signalera une nouvelle erreur. Ce processus continue jusqu'à ce que le joueur trouve le mot recherché ou jusqu'à ce que le joueur fasse la 4ième erreur (dans ce dernier cas, on lui donnera le mot recherché).

- `initialiser` : fonction sans paramètre qui permet à l'utilisateur de choisir le nombre de lettres du mot à rechercher, récupère la liste des mots de cette longueur, en choisit un au hasard et lance la partie.

Partie 8: Les exceptions

Exercice 8.1 : saisie d'un entier avec gestion des erreurs

Ecrire un programme permettant de saisir un entier. Si l'utilisateur saisit autre chose qu'un entier, afficher un message d'erreur en gérant l'exception `ValueError`. Le programme doit boucler jusqu'à ce que l'utilisateur saisisse bien un entier.

Exercice 8.2 : saisir des entiers et gérer une division par zéro

Ecrire un programme permettant de saisir deux entiers : une distance à parcourir (en km) et une vitesse moyenne (en km/h). Pour chaque saisie, on adoptera le même comportement que dans l'exercice précédent via une fonction prenant en paramètre le message d'invite à afficher.

Dans un deuxième temps, le programme doit calculer le nombre d'heures nécessaires pour parcourir la distance spécifiée à la vitesse spécifiée. Si la vitesse est égale à 0 alors en gérant l'exception `ZeroDivisionError`, le programme doit afficher un message d'erreur. Sinon le programme doit afficher la durée obtenue.

Exercice 8.3 : gestion d'un menu textuel

Ecrire un programme permettant de gérer l'affichage d'un menu et de permettre à l'utilisateur de choisir une des options du menu. Les options du menu seront systématiquement numérotées et l'utilisateur devra saisir un entier correspondant à une des options. Si l'utilisateur ne saisit pas un entier ou ne saisit pas une des options de menu, un message d'erreur (différent dans les deux cas) devra être affiché.

On définir pour cela 2 fonctions :

- une fonction `menu(listeOptions)` prenant en paramètre une liste de chaines de caractères correspondant au texte des options du menu et qui affiche le menu correspondant avec numérotation des options
- une fonction `saisirOption(min,max)` capable de saisir un entier compris entre min et max (compris). Si l'utilisateur ne saisit pas un entier ou saisit un entier hors de l'intervalle, la fonction doit afficher un message d'erreur adapté et demander une nouvelle saisie.

Partie 9: Les objets

Exercice 9.1 : définition de la classe Chien

On souhaite dans cet exercice créer une classe `Chien` ayant deux attributs :

- un nom `nom` de type `str`,
- un poids `poids` de type `float`.

Cette classe possède aussi différentes méthodes décrites ci-dessous (`chien` est un objet de type `Chien`) :

- `chien.donne_nom()` qui renvoie la valeur de l'attribut `nom` ;
- `chien.donne_poids()` qui renvoie la valeur de l'attribut `poids` ;
- `chien.machouille(jouet)` qui renvoie son argument, la chaîne de caractères `jouet`, privé de son dernier caractère ;
- `chien.aboie(nb_fois)` qui renvoie la chaîne `'Ouaf' * nb_fois`, où `nb_fois` est un entier passé en argument ;
- `chien.mange(ration)` qui modifie l'attribut `poids` en lui ajoutant la valeur de l'argument `ration` (de type `float`).

On ajoute les contraintes suivantes concernant la méthode `mange` :

- on vérifiera que la valeur de `ration` est comprise entre 0 (exclu) et un dixième du poids du chien (inclus),
- la méthode renverra `True` si `ration` satisfait ces conditions et que l'attribut `poids` est bien modifié, `False` dans le cas contraire.

Voici un exemple d'appels dans la console Python permettant de tester cette classe :

```
>>> medor = Chien('Médor', 12.0)
>>> medor.donne_nom()
'Médor'
>>> medor.donne_poids()
12.0
>>> medor.machouille('bâton')
'bâto'
>>> medor.aboie(3)
'OuafOuafOuaf'
>>> medor.mange(2.0)
False
>>> medor.mange(1.0)
True
>>> medor.donne_poids()
13.0
>>> medor.mange(1.3)
True
```

Exercice 9.2 : les classes Train et Wagon : premier cas de composition

On souhaite dans cet exercice créer une classe `Train` permettant de relier des objets de type `Wagon`.

Un objet de type `Wagon` possède deux attributs :

- un contenu `contenu` de type `str`,
- un attribut `suivant` de type `Wagon` contenant une référence vers le wagon suivant.

On inclut aussi deux méthodes permettant d'afficher le wagon sous forme d'une chaîne de caractère.

Un objet de la classe `Train` possède deux attributs :

- `premier` contient son premier wagon (de type `Wagon`) ou `None` si le train est vide (il n'y a que la locomotive),
- `nb_wagons` (de type `int`) contient le nombre de wagons attachés à la locomotive.

Lors de sa création, un objet de type `Train` n'a aucun wagon.

Les méthodes de la classe `Train` sont présentées ci-dessous (`train` est un objet de type `Train`) :

- `train.est_vide()` renvoie `True` si `train` est vide (ne comporte aucun wagon), `False` sinon ;
- `train.donne_nb_wagons()` renvoie le nombre de wagons de `train` ;
- `train.transporte_du(contenu)` détermine si `train` transporte du `contenu` (une chaine de caractères). Renvoie `True` si c'est le cas, `False` sinon ;
- `train.ajoute_wagon(wagon)` ajoute un `wagon` à la fin du train. On passe en argument le `wagon` à ajouter ;
- `train.supprime_wagon_de(contenu)` prend en argument une chaîne de caractères `contenu` et supprime le premier wagon de `contenu` du train. Si le `train` est vide ou ne comporte aucun wagon de `contenu`, la méthode renvoie `False`. S'il en contient un et que celui-ci est effectivement supprimé, la méthode renvoie `True`.

On inclut là-aussi aussi deux méthodes permettant d'afficher le train dans la console ou sous forme d'une chaîne de caractères.

Voici un exemple de test de ces classes Python dans la console :

```
>>> train = Train()
>>> w1 = Wagon('blé')
>>> train.ajoute_wagon(w1)
>>> w2 = Wagon('riz')
>>> train.ajoute_wagon(w2)
>>> train.ajoute_wagon(Wagon('sable'))
>>> train
'Locomotive - Wagon de blé - Wagon de riz - Wagon de sable'
>>> train.est_vide()
False
```

```
>>> train.donne_nb_wagons()

3
>>> train.transporte_du('blé')
True
>>> train.transporte_du('matériel')
False

>>> train.supprime_wagon_de('riz')

True
>>> train
'Locomotive - Wagon de blé - Wagon de sable'
>>> train.supprime_wagon_de('riz')
False
```

Exercice 9.3 : gestion d'une bibliothèque

Question 1 : définition d'une classe Auteur

On souhaite pouvoir décrire des auteurs de livres. On va pour cela créer une classe Auteur avec deux variables d'instance : le nom et le prénom. La classe contiendra un constructeur, un accesseur en lecture pour chaque variable d'instance et une méthode __str__ permettant d'obtenir une chaine contenant le prenom suivi du nom. Ecrire un programme qui définit plusieurs auteurs (sans saisie) et qui les affichent via la méthode __str__ de la classe Auteur.

Question 2 : définition d'une classe Livre par composition

Dans ce programme, on repart du programme précédent. On souhaite en plus pouvoir gérer des livres. On va pour cela introduite une classe Livre avec 2 variables d'instance : le titre et l'auteur (de type Auteur). Cette nouvelle classe sera pourvue des méthodes suivantes : un constructeur, un accesseur en lecture pour chacune des variables d'instance et une méthode __str__. Si un livre de Patrick Suskind a pour titre « Le parfum » alors cette dernière méthode doit renvoyer une chaine de caractères du type « Le parfum de Patrick Suskind ».

Ecrire un programme qui définit les mêmes auteurs que dans l'exercice 46, définit ensuite plusieurs livres de ses auteurs puis affiche ces livres via la méthode __str__ de la classe Livre

Question 3: définition d'une classe Bibliotheque par composition

On va à nouveau partir du programme précédent en y ajoutant cette fois une classe Bibliotheque permettant de gérer un collection de livres. Cette classe aura une variable d'instance de type Set contenant des objets de type Livre.

Elle devra disposer des méthodes suivantes :

- `ajouter` : méthode prenant un livre en paramètre et l'ajoutant à la bibliothèque

- `getNbLivres` : méthode sans paramètre renvoyant le nombre de livres de la bibliothèque

- `__str__` : méthode renvoyant une chaîne de caractères listant tous les livres (avec leurs auteur) de la bibliothèque

Ecrire un programme permettant d'ajouter les livres de l'exercice précédent à la bibliothèque, puis d'afficher le nombre de livres de la bibliothèque ainsi que la liste de ses livres.

Question 4 : comparer des livres et des auteurs

On va maintenant ajouter une méthode `__eq__` (*equals*) aux deux classes Auteur, Livre permettant de comparer deux auteurs ou deux livres entre eux. Deux auteurs sont considérés comme identiques s'ils ont le même nom et le même prénom. Deux livres sont considérés comme identiques s'ils ont le même titre et le même auteur.

NB : avant de comparer 2 objets, il faut toujours commencer par vérifier qu'ils sont du même type, ie. instance de la même classe (voir fonction `isInstance`).

Pour que cela fonctionne, il est nécessaire de définir une fonction hash() dans les classes Auteur et Livre. Voici leur code :

```
def __hash__(self):
        return (hash(self.nom)*7)+hash(self.prenom)

def __hash__(self):
        return (hash(self.titre)*7)+hash(self.auteur)
```

Ajouter dans le programme principal différents appels permettant de tester l'égalité entre des auteurs et des livres.

Question 5 : ajout de fonctionnalités à la classe Bibliothèque

En s'appuyant sur les méthodes définies dans la question 2, on souhaite maintenant ajouter des fonctionnalités à la classe Bibliothèque :

- méthode `rechercherTitre` : méthode prenant en paramètre un titre de livre et renvoyant le livre de la bibliothèque ayant ce titre.

- méthode `rechercherMot` : méthode prenant en paramètre une chaine de caractères et renvoyant l'ensemble de livres (Set) de la bibliothèque dont le titre contient cette chaine de caractères.

- méthode `rechercherAuteur` : méthode prenant en paramètre un nom et un prénom et renvoyant l'ensemble des livres (Set) de la bibliothèque écrits par cet auteur.

Ajouter dans le programme principal des appels permettant de tester ces différentes méthodes (ne pas oublier les cas ou ces méthode renvoie un ensemble vide ou aucun objet).

Exercice 9.4 : premier cas d'héritage

Question 1 : la classe Velo

Dans cet exercice on va commencer par construire une classe Velo permettant de décrire un vélo avec les caractéristiques suivantes :

- le modèle : 'VILLE', 'VTT' ou 'VTC'

- le genre : ‘HOMME’ ou ‘FEMME’

Cette classe devra disposer des méthodes suivantes :

- un constructeur permettant d’initialiser le modèle et le genre
- une méthode isFeminin() renvoyant vrai si le vélo est un vélo pour femme
- une méthode __str__ renvoyant une chaine de caractères contenant le modèle et le genre du vélo

On écrira un main permettant de tester l’ensemble de ces fonctionnalités

Question 2 : la classe VeloElectrique

Il s’agit maintenant de définir une nouvelle classe VeloElectrique par héritage de la classe Velo. Cette nouvelle classe aura une propriété supplémentaire : l’autonomie en kms de la batterie du vélo.

Elle aura également une méthode supplémentaire : isLongueDistance qui renvoie vrai si l’autonomie du vélo électrique dépasse 100km.

Ecrire un main permettant de tester cette nouvelle classe.

Exercice 9.5 : retour sur l’exercice sur la bibliothèque : introduction de la classe BD par héritage

On va maintenant ajouter une nouvelle classe permettant de définir des bandes dessinées. Cette classe va être définie par héritage de la classe Livre et aura un attribut spécifique : le nom du dessinateur (chaine de caractères).

Cette classe devra disposer des méthodes suivantes :

- un constructeur prenant en paramètres le titre de la BD, le nom et le prénom de l’auteur ainsi que le nom du dessinateur

- une méthode __str__ permettant de renvoyer une chaine de caractère décrivant la BD

- une méthode __eq__ permettant de déterminer si 2 BD sont égales. Pour cela, il faut que les attributs provenant de la classe Livre soient égaux et qu’en plus les dessinateurs soient égaux.

Voici la méthode hash qui doit être ajoutée à la classe BD :

```
def __hash__(self):
        return super().__hash__()+hash(self.dessinateur)
```

On ajoutera enfin dans la classe Bibliothèque une méthode `rechercherDessinateur` prenant en paramètre une chaîne de caractères et permettant de rechercher dans la bibliothèque les BD dont le dessinateur porte le nom passé en paramètre.

Modifier le programme principal pour ajouter des BD à la bibliothèque et faire des recherches sur ces BD. On vérifiera que les anciens appels de méthodes de la classe Bibliothèque fonctionnent toujours.