MESURES PHYSIQUES 1è année **2010 – 2011**

Informatique Scientifique **version 2.2**

Python 3

Exercices corrigés

Énoncés des exercices

Remarque

☞ Les exercices suivants sont fournis à titre d'exemples et de modèles.
Ils sont soit simples, soit moins simples (notés ▷ dans la marge) soit difficiles (notés ▷▷).

Les scripts du cours

Cours nº 1 : « Premiers pas en Python »

1. Affectez les variables `temps` et `distance` par les valeurs 6.892 et 19.7.
   Calculez et affichez la valeur de la vitesse.
   Améliorez l'affichage en imposant un chiffre après le point décimal.

2. Saisir un nom et un âge en utilisant l'instruction `input()`. Les afficher.
   Refaire la saisie du nom, mais avec l'instruction `raw_input()`. L'afficher.
   Enfin, utilisez la « bonne pratique » : recommencez l'exercice en *transtypant* les saisies effectuées avec l'instruction `raw_input()`

Cours nº 2 : « Contrôle du flux d'instructions »

1. Saisissez un flottant. S'il est positif ou nul, affichez sa racine, sinon affichez un message d'erreur.

2. L'ordre *lexicographique* est celui du dictionnaire.
   Saisir deux mots, comparez-les pour trouver le « plus petit » et affichez le résultat.
   Refaire l'exercice en utilisant l'instruction ternaire :
   ```
   <res> = <a> if <condition> else <b>
   ```

3. On désire sécuriser une enceinte pressurisée.
   On se fixe une pression seuil et un volume seuil : `pSeuil = 2.3`, `vSeuil = 7.41`.
   On demande de saisir la pression et le volume courant de l'enceinte et d'écrire un script qui simule le comportement suivant :
   - si le volume *et* la pression sont supérieurs aux seuils : arrêt immédiat ;
   - si seule la pression est supérieure à la pression seuil : demander d'augmenter le volume de l'enceinte ;
   - si seul le volume est supérieur au volume seuil : demander de diminuer le volume de l'enceinte ;
   - sinon déclarer que « tout va bien ».

   Ce comportement sera implémenté par une alternative multiple.

4. Initialisez deux entiers : `a = 0` et `b = 10`.
   Écrire une boucle affichant et incrémentant la valeur de `a` tant qu'elle reste inférieure à celle de `b`.
   Écrire une autre boucle décrémentant la valeur de `b` et affichant sa valeur si elle est impaire. Boucler tant que b n'est pas nul.

5. Écrire une *saisie filtrée* d'un entier dans l'intervalle 1 à 10, bornes comprises. Affichez la saisie.

6. Affichez chaque caractère d'une chaîne en utilisant une boucle `for`.
   Affichez chaque élément d'une liste en utilisant une boucle `for`.

7. Affichez les entiers de 0 à 15 non compris, de trois en trois, en utilisant une boucle `for` et l'instruction `range()`.

8. Utilisez l'instruction `break` pour interrompre une boucle `for` d'affichage des entiers de 1 à 10 compris, lorsque la variable de boucle vaut 5.

9. Utilisez l'instruction `continue` pour modifier une boucle `for` d'affichage de tous entiers de 1 à 10 compris, sauf lorsque la variable de boucle vaut 5.

10. Utilisez une *exception* pour calculer, dans une boucle évoluant de -3 à 3 compris, la valeur de $\sin(x)/x$.

11. La clause `else` des boucles. Dans cet exercice, effectuez les saisies avec des `integerbox` et les affichages avec des `msgbox`, tous deux appartenant au module `easygui`.
    Initialisez une liste avec 5 entiers de votre choix puis saisissez un entier.
    Dans une boucle for, parcourez la liste. Si l'entier saisie appartient à la liste, sauvez-le et interrompez la boucle (puisque vous l'avez trouvé). Si la boucle s'est bien terminée, utilisez une clause `else` pour afficher un message l'annonçant.
    Entrez maintenant un autre entier, cette fois-ci positif.
    Écrivez une boucle `while` pour déterminer si cet entier est premier. S'il ne l'est pas, la boucle devra afficher le premier diviseur trouvé et s'interrompre. S'il est premier, l'afficher dans une clause `else`.

Cours n° 3 : « Les fonctions »

1. Écrire une procédure `table` avec quatre paramètres : `base`, `debut`, `fin` et `inc`.
   Cette procédure doit afficher la table des `base`, de `debut` à `fin`, de `inc` en `inc`.
   Tester la procédure par un appel dans le programme principal.

2. Écrire une fonction `cube` qui retourne le cube de son argument.

Écrire une fonction `volumeSphere` qui calcule le volume d'une sphère de rayon `r` fourni en argument et qui utilise la fonction `cube`.
Tester la fonction `volumeSphere` par un appel dans le programme principal.

3. Écrire une fonction `maFonction` qui retourne $f(x) = 2x^3 + x - 5$. ◁
   Écrire une procédure `tabuler` avec quatre paramètres : `fonction`, `borneInf`, `borneSup` et `nbPas`. Cette procédure affiche les valeurs de `fonction`, de `borneInf` à `borneSup`, tous les `nbPas`. Elle doit respecter $borneInf < borneSup$.
   Tester cette procédure par un appel dans le programme principal après avoir saisi les deux bornes dans une `floatbox` et le nombre de pas dans une `integerbox` (utilisez le module `easyguiB`).

4. Écrire une fonction `volMasseEllipsoide` qui retourne le volume et la masse d'un ellipsoïde grâce à un tuple. Les paramètres sont les trois demi-axes et la masse volumique. On donnera à ces quatre paramètres des valeurs par défaut.
   On donne : $v = \frac{4}{3}\pi abc$
   Tester cette fonction par des appels avec différents nombres d'arguments.

5. Écrire une fonction `somme` avec un argument « tuple de longueur variable » qui calcule la somme des nombres contenus dans le tuple.
   Tester cette fonction par des appels avec différents tuples d'entiers ou de flottants.

6. Écrire une autre fonction `somme` avec trois arguments, et qui renvoie leur somme.
   Dans le programme principal, définir un tuple de trois nombres, puis utilisez la syntaxe d'appel à la fonction qui *décompresse* le tuple. Affichez le résultat.

7. Écrire une fonction `unDictionnaire` avec un argument « dictionnaire de longueur variable », et qui affiche son argument.
   Dans le programme principal, définir un dictionnaire, puis utilisez la syntaxe d'appel à la fonction qui *décompresse* le dictionnaire. Affichez le résultat.

Cours n° 4 : « Structures de données Python »

1. définir la liste : `liste =[17, 38, 10, 25, 72]`, puis effectuez les actions suivantes :
   - triez et affichez la liste ;
   - ajoutez l'élément 12 à la liste et affichez la liste ;
   - renversez et affichez la liste ;
   - affichez l'indice de l'élément 17 ;
   - enlevez l'élément 38 et affichez la liste ;
   - affichez la sous-liste du 2^{e}au 3^{e}élément ;
   - affichez la sous-liste du début au 2^{e}élément ;
   - affichez la sous-liste du 3^{e}élément à la fin de la liste ;
   - affichez la sous-liste complète de la liste ;

– affichez le dernier élément en utilisant un indiçage négatif.

Bien remarquer que certaines méthodes de liste ne retournent rien.

2. Initialisez `truc` comme une liste vide, et `machin` comme une liste de cinq flottants nuls. Affichez ces listes.
   Utilisez la fonction `range()` pour afficher :
   – les entiers de 0 à 3 ;
   – les entiers de 4 à 7 ;
   – les entiers de 2 à 8 par pas de 2.

   Définir `chose` comme une liste des entiers de 0 à 5 et testez l'appartenance des éléments 3 et 6 à `chose`.

3. Utilisez une liste en compréhension pour ajouter 3 à chaque élément d'une liste d'entiers de 0 à 5.

4. Utilisez une liste en compréhension pour ajouter 3 à chaque élément d'une liste d'entiers de 0 à 5, mais seulement si l'élément est supérieur ou égal à 2.

▷ 5. Utilisez une liste en compréhension pour obtenir la liste `['ad', 'ae', 'bd', 'be', 'cd', 'ce']` à partir des chaînes `"abc"` et `"de"`.
   Indication : utilisez deux boucles `for` imbriquées.

6. Utilisez une liste en compréhension pour calculer la somme d'une liste d'entiers de 0 à 9.

7. Définir deux ensembles (*sets*) : $X = \{a, b, c, d\}$ et $Y = \{s, b, d\}$, puis affichez les résultats suivants :
   – les ensembles initiaux ;
   – le test d'appartenance de l'élément `'c'` à X ;
   – le test d'appartenance de l'élément `'a'` à Y ;
   – les ensembles $X - Y$ et $Y - X$;
   – l'ensemble $X \cup Y$ (union) ;
   – l'ensemble $X \cap Y$ (intersection).

▷▷ 8. Écrire une fonction `compterMots` ayant un argument (une chaîne de caractères) er qui renvoie un *dictionnaire* qui contient la fréquence de tous les mots de la chaîne entrée.

9. Le type dictionnaire (ou tableau associatif) permet de représenter des tableaux structurés. En effet, à chaque *clé* un dictionnaire associe une *valeur*, et cette valeur peut elle-même être une structure de donnée (liste, tuple ou un dictionnaire…).

Soit le tableau suivant représentant des informations physico-chimiques sur des éléments simples (température d'ébullition (T_e) et de fusion (T_f), numéro (Z) et masse (M) atomique :

Au	T_e / T_f	2970	1063
	Z / A	79	196.967
Ga	T_e / T_f	2237	29.8
	Z / A	31	69.72

Affectez les données de ce tableau à un dictionnaire `dico` python de façon à pouvoir écrire par exemple :

```
>>> print dico["Au"]["Z/A"][0] # affiche : 79
```

10. Implémentez une pile LIFO avec une liste.

    Pour cela, définir trois fonctions :

    `pile:` qui retourne une pile à partir d'une liste variable d'éléments passés en paramètre ;

    `empile:` empile un élément en « haut » de la pile ;

    `depile:` dépile un élément du « haut » de la pile.

11. De la même manière, implémentez une queue FIFO avec une liste. Essayez d'ajouter un menu de manipulation de la queue. ◁

    Conseil : N'utilisez que des procédures sans argument et une liste en variable globale.

Cours nº 5 : Interlude : nombres parfaits et nombres chanceux

Définitions :

- On appelle *nombre premier* tout entier naturel supérieur à 1 qui possède exactement deux diviseurs, lui-même et l'unité ;
- On appelle *diviseur propre* de `n`, un diviseur quelconque de `n`, `n` exclu ;
- un entier naturel est dit *parfait* s'il est égal à la somme de tous ses diviseurs propres ;
- les nombres `a` tels que : $(a+n+n^2)$ est premier pour tout `n` tel que $0 \leqslant n < (a-1)$, sont appelés *nombres chanceux*.

Écrire un module (`parfait_chanceux_m.py`) définissant quatre fonctions : `somDiv`, `estParfait`, `estPremier`, `estChanceux` et un auto-test :

- la fonction `somDiv` retourne la somme des diviseurs propres de son argument ;
- les trois autres fonctions vérifient la proprriété donnée par leur définition et retourne un booléen. Plus précisément, si par exemple la fonction `estPremier` vérifie que son argument est premier, elle retourne `True`, sinon elle retourne `False`.

La partie de test doit comporter quatre appels à la fonction `verif` permettant de tester `somDiv(12)`, `estParfait(6)`, `estPremier(31)` et `estChanceux(11)`.

Puis écrire le programme principal (`parfait_chanceux.py`) qui comporte :
- l'initialisation de deux listes : `parfaits` et `chanceux` ;
- une boucle de parcours de l'intervalle [2,1000] incluant les tests nécessaires pour remplir ces listes ;
- enfin l'affichage de ces listes dans des boîtes de message du module `easygui`.

Cours n° 6 : « Modules et fichiers »

1. Écrire un module de calcul des racines du trinôme réel : $ax^2 + bx + c$.
   Le module définit une fonction `trinome` avec les trois paramètres du trinôme, a, b et c. La fonction doit retourner un tuple dont le premier élément est le nombre de racines du trinôme (0, 1 ou 2), et les autres éléments sont les racines éventuelles.

   Testez votre fonction avec les trois jeux de valeurs suivantes : $1,-3,2$, $1,-2,1$ et $1,1,1$.

2. Écrire un programme principal utilisant le module précédent.
   Les trois paramètres seront saisis dans une `flotbox` du module `easyguiB` et les résultats seront affichés dans une `msgbox`.

Cours n° 7 : « Programmation Orientée Objet »

1. Définir une classe `MaClasse` possédant les attributs suivants :

   données : deux attributs de classes : `x = 23` et `y = x + 5`.

   méthode : une méthode `affiche` contenant un attribut d'instance `z = 42` et les affichages de `y` et de `z`.

   Dans le programme principal, instanciez un objet de la classe `MaClasse` et invoquez la méthode `affiche`.

2. Définir une classe `Vecteur2D` avec un constructeur fournissant les coordonnées par défaut d'un vecteur du plan (par exemple : `x = 0` et `y = 0`).

   Dans le programme principal, instanciez un `Vecteur2D` sans paramètre, un `Vecteur2D` avec ses deux paramètres, et affichez-les.

3. Enrichissez la classe `Vecteur2D` précédente en lui ajoutant une méthode d'affichage et une méthode de surcharge d'addition de deux vecteurs du plan.

   Dans le programme principal, instanciez deux `Vecteur2D`, affichez-les et affichez leur somme.

Énoncés

Cours nº 8 : « Notions de COO et d'encapsulation »

1. Définir une classe `Rectangle` avec un constructeur donnant des valeurs (`longueur` et `largeur`) par défaut et un attribut `nom = "rectangle"`, une méthode d'affichage et une méthode `surface` renvoyant la surface d'une instance.

   Définir une classe `Carre` héritant de `Rectangle` et qui surcharge l'attribut d'instance : `nom = "carré"`.
   Dans le programme principal, instanciez un `Rectangle` et un `Carre` et affichez-les.

2. Définir une classe `Point` avec un constructeur fournissant les coordonnées par défaut d'un point du plan (par exemple : `x = 0.0` et `y = 0.0`).

   Définir une classe `Segment` dont le constructeur possède quatre paramètres : deux pour l'origine et deux pour l'extrémité. Ce constructeur définit deux attributs : `orig` et `extrem`, instances de la classe `Point`. De cette manière, vous concevez une classe *composite* : La classe `Segment` est composée de deux instances de la classe `Point`.
   Ajouter une méthode d'affichage.

   Enfin écrire un auto-test qui affiche une instance de Segment initialisée par les valeurs 1, 2, 3 et 4.

3. Définir une fonction *fabrique* `creer_plus` renvoyant une fonction *fermeture* `plus`. ◁
   `cree_plus` a un argument `ajout`. Son code ne renferme que la fonction `plus` qui, elle aussi, possède un argument `increment` et dont le code se contente de renvoyer la somme : `(ajout + increment)`.
   Dans le programme principal, créez deux fonctions, par exemple `p = creer_plus(23)` et `p = creer_plus(42)`, puis affichez les valeurs données par `p(100)` et `q(100)`.

4. Écriture d'une fonction *fabrique* renvoyant une instance de classe. ◁
   Définir une classe `CasNormal` contenant une méthode `uneMethode` qui affiche `"normal"`.
   Définir une classe `CasSpecial` contenant une méthode `uneMethode` qui affiche `"spécial"`.
   Enfin définir la fonction fabrique `casQuiConvient` avec un paramètre `estNormal` initialisé par défaut à `True`. Si le paramètre est vérifié, le corps de la fonction renvoie une instance de la classe `CasNormal`, sinon il renvoie une instance de la classe `CasSpecial`.

   Dans le programme principal, créez l'instance que vous désirez grâce à la fabrique, puis vérifiez son type en appelant dessus la méthode `uneMethode`.

Scripts supplémentaires

1. Écrire un programme qui, à partir de la saisie d'un rayon et d'une hauteur, calcule le volume d'un cône droit.

2. Une boucle while : entrez un prix HT (entrez 0 pour terminer) et affichez sa valeur TTC.

3. Une autre boucle while : calculez la somme d'une suite de nombres positifs ou nuls. Comptez combien il y avait de données et combien étaient supérieures à 100.
   Un nombre inférieur ou égal à 0 indique la fin de la suite.

4. L'utilisateur donne un entier positif n et le programme affiche `PAIR` s'il est divisible par 2 et `IMPAIR` sinon.

5. L'utilisateur donne un entier positif et le programme annonce combien de fois de suite cet entier est divisible par 2.

▷ 6. L'utilisateur donne un entier supérieur à 1 et le programme affiche, s'il y en a, tous ses diviseurs propres *sans répétition* ainsi que leur nombre. S'il n'y en a pas, il indique qu'il est premier. Par exemple :

```
Entrez un entier strictement positif : 12
Diviseurs propres sans répétition de 12 : 2 3 4 6 (soit 4 diviseurs propres)

Entrez un entier strictement positif : 13 Diviseurs propres sans
répétition de 13 : aucun ! Il est premier
```

7. Écrire un programme qui estime la valeur de la constante mathématique e en utilisant la formule :

$$e = \sum_{i=0}^{n} \frac{1}{i!}$$

   Pour cela, définissez la fonction factorielle et, dans votre programme principal, saisissez l'ordre n et affichez l'approximation correspondante de e.

8. Un gardien de phare va aux toilettes cinq fois par jour or les WC sont au rez-de-chaussée…
   Écrire une procédure (donc sans retour) `hauteurparcourue` qui reçoit deux paramètres le nombre de marches du phare et la hauteur de chaque marche (en cm), et qui affiche :

```
Pour x marches de y cm, il parcourt z.zz m par semaine.
```

   On n'oubliera pas :
   – qu'une semaine comporte 7 jours ;
   – qu'une fois en bas, le gardien doit remonter ;
   – que le résultat est à exprimer en m.

9. Un permis de chasse à points remplace désormais le permis de chasse traditionnel. Chaque chasseur possède au départ un capital de 100 points. S'il tue une poule il perd 1 point, 3 points pour 1 chien, 5 points pour une vache et 10 points pour un ami. Le permis coûte 200 euros.
   Écrire une fonction `amende` qui reçoit le nombre de victimes du chasseur et qui renvoie la somme due.
   Utilisez cette fonction dans un programme principal qui saisit le nombre de victimes et qui affiche la somme que le chasseur doit débourser.

10. Je suis ligoté sur les rails en gare d'Arras. Écrire un programme qui affiche un tableau me permettant de connaître l'heure à laquelle je serai déchiqueté par le train parti de la gare du Nord à 9h (il y a 170 km entre la gare du Nord et Arras).
    Le tableau prédira les différentes heures possibles pour toutes les vitesses de 100 km/h à 300 km/h, par pas de 10 km/h, les résultats étant arrondis à la minute inférieure.
    – Écrire une procédure `tchacatchac` qui reçoit la vitesse du train et qui affiche l'heure du drame ;
    – écrire le programme principal qui affiche le tableau demandé.

11. Un programme principal saisit une chaîne d'ADN valide et une séquence d'ADN valide (« valide » signifie qu'elles ne sont pas vides et sont formées exclusivement d'une combinaison arbitraire de `"a"`, `"t"`, `"g"` ou `"c"`). ◁
    Écrire une fonction `valide` qui renvoie vrai si la saisie est valide, faux sinon.
    Écrire une fonction `saisie` qui effectue une saisie valide et renvoie la valeur saisie sous forme d'une chaîne de caractères.
    Écrire une fonction `proportion` qui reçoit deux arguments, la chaîne et la séquence et qui retourne la proportion de séquence dans la chaîne (c'est-à-dire son nombre d'occurrences).
    Le programme principal appelle la fonction `saisie` pour la chaîne et pour la séquence et affiche le résultat.
    Exemple d'affichage :
    ```
    Il y a 13.33 % de "ca" dans votre chaîne.
    ```

12. Il s'agit d'écrire, d'une part, un programme principal, et d'autre part, une fonction utilisée dans le programme principal.
    L'utilisateur remplit un tableau de $N = 100$ entiers avec des entiers aléatoires en utilisant une fonction `randint(`a, b`)` qui retourne un entier entre a et $b-1$. Une fonction nommée `indiceDuMin()` reçoit ce tableau et retourne l'indice de la case qui contient le minimum.
    Écrire la fonction `indiceDuMin()`.
    Écrire le programme qui échange le premier élément du tableau avec le minimum de ce tableau.

13. Un tableau tab comporte $N = 100$ variables flottantes dont les n premières ($n < 100$) sont utilisées.

Écrire une fonction `indiceDuMax()` qui retourne l'indice du plus grand flottant parmi ces n, et une autre `indiceDuMin()` qui retourne l'indice du plus petit.

Écrire ensuite un programme principal effectuant les actions suivantes :
- saisie *filtrée* de n (vous devez faire en sorte que n ne puisse pas être saisi hors de ses limites) ;
- remplissage aléatoire des n premières valeurs de tab (on utilisera le module `random()`, sans argument, qui retourne un flottant au hasard entre 0.0 et +1.0) ;
- affichage de *l'amplitude* du tableau (écart entre sa plus grande et sa plus petite valeur) ;
- affichage de la *moyenne* des n premières valeurs de tab.

14. Écrire une fonction `conv()` qui reçoit deux paramètres, une température et un entier `n`, et qui retourne la conversion Celsius → Fahrenheit (`n` = 1), ou Fahrenheit → Celsius (`n` = 2).
Rappel : $T_F = 32 + 1.8 \times T_C$

15. Fonction renvoyant plusieurs valeurs sous forme d'un *tuple*.
Écrire une fonction `minMaxMoy` qui reçoit une liste d'entiers et qui renvoie le minimum, le maximum et la moyenne de cette liste. Le programme principal appellera cette fonction avec la liste : `[10, 18, 14, 20, 12, 16]`.

16. Saisir un entier entre 1 et 3999 (pourquoi cette limitation ?). L'afficher en nombre romain.

▷ 17. Améliorer le script précédent en utilisant la fonction `zip()`.

▷ 18. Un tableau contient n entiers ($2 < n < 100$), tous compris entre 0 et 500. Vérifier qu'ils sont tous différents.

19. L'utilisateur donne un entier n entre 2 et 12, le programme donne le nombre de façons de faire n en lançant deux dés.

20. Même problème que le précédent mais avec n entre 3 et 18 et trois dés.

▷▷ 21. Généralisation des deux questions précédentes. L'utilisateur saisit deux entrées, d'une part le nombre de dés, nbd (que l'on limitera pratiquement à 10), et d'autre part la somme, s, comprise entre nbd et $6.nbd$. Le programme calcule et affiche le nombre de façons de faire s avec les nbd dés.

▷▷ 22. Même problème que le précédent mais codé récursivement.

23. Pour faire des calculs sur les matrices carrées, on peut utiliser le type « liste de listes » et indexer un élément de la 3è ligne et 4è colonne par $m[2][3]$ (compte-tenu que les indices commencent à 0).

    On déclare trois matrices carrées de dimension N : $m1$, $m2$ et $m3$ contenant des entiers.

    On affecte$m1$, ligne à ligne, par les N^2 premiers entiers pairs commençant à 2 ; $m2$ est la matrice unité, c'est-à-dire qu'elle contient des 1 sur la diagonale principle (NW-SE) et des 0 partout ailleurs.

    Pratiquement, on se limitera à $Nmax = 10$. Écrire l'algorithme du calcul de :

$$m3 = m1 - m2$$

    .

Solutions des exercices

Les scripts du cours

Cours nº 1

```
# -*- coding: UTF-8 -*-
"""La fonction print()."""
# fichier : cours1_05.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
## affichage simple :
temps = 6.892
distance = 19.7
vitesse = distance/temps
print("vitesse =", vitesse)

## affichage formate :
print("{}".format("-"*23))
print("\n vitesse = {:.2f} m/s".format(vitesse)) # arrondi a 2 chiffres
```

```
# -*- coding: UTF-8 -*-
"""Les instructions input."""
# fichier : cours1_10.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
## instruction input :
nom = input("ton nom ? ")
age = input("age ? ")
age = float(age)
print("\t Nom :", nom, "\t Age :", age)

## bonnes pratiques :
nom = input("ton nom ? ")    # pour une chaine
age = float(input("age ? ")) # sinon : transtyper explicitement

print("{}\n\t Nom : {}\t Age : {}".format("-"*40, nom, age))
```

Cours nº 2

```
# -*- coding: UTF-8 -*-
"""Les instructions de choix."""
# fichier : cours2_05.py
# auteur : Bob Cordeau

# import
from math import sqrt

# programme principal ------------------------------------------------
x = float(input("x ? "))

if x >= 0:
    y = sqrt(x)
    print("La racine de {:.2f} est {:.3f}".format(x, y))
else:
    print("On ne peut pas prendre la racine d'un reel negatf !")

print("\nAu revoir")
```

```
ï»¿# -*- coding: UTF-8 -*-
"""Les instructions de choix (2)
   Trouver le minimum de deux nombres"""
# fichier : cours2_10.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
print("Donnez deux valeurs entieres :")
x = int(input("n1 = "))
y = int(input("n2 = "))

# ecriture classique :
if x < y:
    plus_petit = x
else:
    plus_petit = y

# ecriture compacte :
plus_petit = x if x < y else y

print("\nLa plus petite des deux est", plus_petit)

print("\nAu revoir")
```

```
ï»¿# -*- coding: UTF-8 -*-
"""Les instructions de choix (3)
   Instructions conditionnelles imbriquees"""
# fichier : cours2_15.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
p_seuil, v_seuil = 2.3, 7.41
print("Seuil pression :", p_seuil, "\tSeuil volume ;", v_seuil, "\n")
```

```
pression = float(input("Pression courante = "))
volume = float(input("Volume courant = "))

if (pression > p_seuil) and (volume > v_seuil):
    print("\t pression ET volume eleves. Stoppez !")
elif pression > p_seuil:
    print("\t Il faut augmenter le volume")
elif volume > v_seuil:
    print("\t Vous pouvez diminuer le volume")
else:
    print("\t Tout va bien !")
```

```
# -*- coding: UTF-8 -*-
"""Instruction 'while'."""
# fichier : cours2_20.py
# auteur : Bob Cordeau

# programme principal -------------------------------------------------
a, b = 0, 10

while a < b:
    print(a, end=" ")
    a = a + 1

print("\n\nAutre exemple :\n")

while b:   # signifie : tant que b est vrai (i-e b non nul)
    b = b - 1
    if b % 2 != 0:
        print(b, end=" ")
print()
```

```
# -*- coding: UTF-8 -*-
"""Une saisie filtree."""
# fichier : cours2_25.py
# auteur : Bob Cordeau

# programme principal -------------------------------------------------
n = int(input("Entrez un entier [1 .. 10] : "))
while not(1 <= n <= 10):
    n = int(input("Entrez un entier [1 .. 10], S.V.P. : "))

print("\nValeur saisie :", n)
```

```
# -*- coding: UTF-8 -*-
"""Instruction 'for' (1)."""
# fichier : cours2_30.py
# auteur : Bob Cordeau

# programme principal -------------------------------------------------
print(" Exemple 1 ".center(40, '-'))

for lettre in "ciao":
```

```
    print(lettre)
print()

print(" Exemple 2 ".center(40, '-'), "\n")

for i in [7, 'm', 2.718]:
    print(i, end=" ")

print("\n\n{:-^40}".format(" idem avec format "))
```

```
# -*- coding: UTF-8 -*-
"""Instruction 'for' (2)."""
# fichier : cours2_35.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
print(" Instruction <range> ".center(40, '-'), "\n")
print(range(5))
print(range(3, 10))
print(range(0, 10, 2))

print("\n"+" <range> dans un <for> ".center(40, '-'), "\n")
for i in range(0, 15, 3):
    print(i, end=" ")
print()
```

```
# -*- coding: UTF-8 -*-
"""Instruction 'break'."""
# fichier : cours2_40.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
for i in range(1, 11):
    if i == 5:
        break
    print(i, end=" ")
print()
```

```
# -*- coding: UTF-8 -*-
"""Instruction 'continue'."""
# fichier : cours2_45.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
for i in range(1, 11):
    if i == 5:
        continue
    print(i, end=" ")
print()
```

```
# -*- coding: UTF-8 -*-
"""Instruction 'try .. except'."""
# fichier : cours2_50.py
# auteur : Bob Cordeau
```

Solutions

```
# import
from math import sin

# programme principal -------------------------------------------------
for x in range(-3, 4): # -3 -2 -1 0 1 2 3
    try:
        print("{:.3f}".format(sin(x)/x), end=" ")
    except:
        print("{:.3f}".format(float(1)), end=" ")
print()
```

```
# -*- coding: UTF-8 -*-
"""Le 'else' de boucle."""
# fichier : cours2_55.py
# auteur : Bob Cordeau

# import
from easygui import integerbox, msgbox

# programme principal -------------------------------------------------
## Boucle for-else
uneSequence = [2, 5, 9, 7, 11]
cible = integerbox("Entrez un entier :","")

for i in uneSequence:
    if i == cible:
        sauve = i
        break
else:
    msgbox("L'element cherche n'est pas dans la sequence.")
    sauve = None

msgbox("Boucle for-else. On obtient : sauve = {}".format(sauve))

## Boucle while-else
y = integerbox("Entrez un entier positif :", "", 5, 1)

x = y // 2
while x > 1:
    if y % x == 0:
        msgbox("{:d} a pour facteur {:d}".format(y, x))
        break
    x = x - 1
else:
    msgbox("{:d} est premier".format(y))
```

Cours nº 3

```
# -*- coding: UTF-8 -*-
"""Procedure."""
# fichier : cours3_05.py
# auteur : Bob Cordeau

# procedure
def table(base, debut, fin, inc):
    """Affiche la table des <base>, de <debut> a <fin>, de <inc> en <inc>."""
    n = debut
    while n <= fin:
        print(n, 'x', base, '=', n*base)
        n = n + inc

# programme principal ------------------------------------------------
table(7, 2, 13, 2)
```

```
# -*- coding: UTF-8 -*-
"""Fonction avec 'return'."""
# fichier : cours3_10.py
# auteur : Bob Cordeau

# import
from math import pi

# fonctions
def cube(x):
    """Calcule le cube de l'argument."""
    return x**3

def volumeSphere(r):
    """Calcule le volume d'une sphere de rayon <r>."""
    return 4 * pi * cube(r) / 3

# programme principal ------------------------------------------------
rayon = float(input("Rayon : "))
print("\nVolume de la sphere de rayon {:.1f} : {:.3f}"
                                .format(rayon, volumeSphere(rayon)))
```

```
# -*- coding: UTF-8 -*-
"""Procedure avec une fonction en parametre."""
# fichier : cours3_15.py
# auteur : Bob Cordeau

# import
from easygui import floatbox, integerbox

# fonctions
def maFonction(x):
    """Definition d'une fonction particuliere."""
    return 2*x**3 + x - 5

def tabuler(fonction, borneInf, borneSup, nbPas):
```

```
    """Affiche les valeurs de <fonction>.

       On doit avoir : (borneInf < borneSup) ET (nnPas > 0)
    """
    h = (borneSup - borneInf) / float(nbPas)
    x = borneInf
    while x <= borneSup:
        y = fonction(x)
        print("f({:.2f}) = {:.3f}".format(x, y))
        x += h

# programme principal -------------------------------------------------
a = floatbox("Borne inferieure :", "", 0.0, -100.0, 100.0)
b = floatbox("Borne superieure :", "", a+.1, a)
n = integerbox("Nombre de pas :", "", 10, 0)
tabuler(maFonction, a, b, n)
```

```
# -*- coding: UTF-8 -*-
"""Parametre avec valeur par defaut."""
# fichier : cours3_20.py
# auteur : Bob Cordeau

# imports
from math import pi
from easygui import floatbox

# fonction
def masseEllipsoide(a=2, b=3, c=4, rho=11.2):
    """Retourne le volume et la masse d'un ellipsoide de demi-axes <a>, <b>, <c>
       et de densite <rho>.
    """
    vol = 4*pi*a*b*c/3
    mas = vol * rho
    print(type(vol))
    print(type(mas))
    return (vol, mas)

# programme principal -------------------------------------------------
aa = floatbox("demi-axe a :", "" , 2.5)
bb = floatbox("demi-axe b :", "" , 3.05)
cc = floatbox("demi-axe c :", "" , 3.8)
dens = floatbox("densite :", "" , 1.5)

a, b = masseEllipsoide(aa, bb, cc, dens)
print("volume = {:.2f} \t masse = {:.2f}".format(a, b))
print('-'*40)
a, b = masseEllipsoide()
print("volume = {:.2f} \t masse = {:.2f}".format(a, b))
print('-'*40)
a, b = masseEllipsoide(rho=dens, b=bb)
print("volume = {:.2f} \t masse = {:.2f}".format(a, b))
print('-'*40)
```

```
# -*- coding: UTF-8 -*-
```

```
"""Fonction : passage d'un tuple."""
# fichier : cours3_25.py
# auteur : Bob Cordeau

# fonction
def somme(*args):
    resultat = 0
    for nombre in args:
        resultat += nombre
    return resultat

# programme principal ------------------------------------------------
print("-"*40)
print(somme(23))
print("\n", "-"*40)
print(somme(-10, 13))
print("\n", "-"*40)
print(somme(23, 42, 13))
print("\n", "-"*40)
print(somme(-10.0, 12))
```

```
# -*- coding: UTF-8 -*-
"""Decompression d'un tuple."""
# fichier : cours3_30.py
# auteur : Bob Cordeau

# fonction
def somme(a, b, c):
    return a+b+c

# programme principal ------------------------------------------------
sequence = (2, 4, 6)
print(somme(*sequence))
```

```
# -*- coding: UTF-8 -*-
"""Fonction : passage d'un dictionnaire."""
# fichier : cours3_35.py
# auteur : Bob Cordeau

# fonction
def unDictionnaire(**kargs):
    return kargs

# programme principal ------------------------------------------------
print(" appel avec des parametres nommes ".center(60, '-'))
print(unDictionnaire(a=23, b=42))

print(" appel avec un dictionnaire decompresse ".center(60, '-'))
mots = {'d':85, 'e':14, 'f':9}
print(unDictionnaire(**mots))
```

Cours nº 4

```
# -*- coding: UTF-8 -*-
"""Listes : methodes et indicage."""
# fichier : cours4_05.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
nombres = [17, 38, 10, 25, 72]
print(" Liste initiale ".center(50, '-'))
print(nombres, '\n')

rien = input('"Entree"')

print(" Tri ".center(50, '-'))
nombres.sort()
print(nombres, '\n')

rien = input('"Entree"')

print(" Ajout d'un element ".center(50, '-'))
nombres.append(12)
print(nombres, '\n')

rien = input('"Entree"')

print(" Retournement ".center(50, '-'))
nombres.reverse()
print(nombres, '\n')

rien = input('"Entree"')

print(" Indice d'un element ".center(50, '-'))
print(nombres.index(17), '\n')

rien = input('"Entree"')

print(" Retrait d'un element ".center(50, '-'))
nombres.remove(38)
print(nombres, '\n')

rien = input('"Entree"')

print(" Indicage ".center(50, '-'))
print("nombres[1:3] =", nombres[1:3])
print("nombres[:2] =", nombres[:2])
print("nombres[2:] =", nombres[2:])
print("nombres[:] =", nombres[:])
print("nombres[-1] =", nombres[-1])
```

```
# -*- coding: UTF-8 -*-
"""Initialisation des listes."""
# fichier : cours4_10.py
# auteur : Bob Cordeau
```

```
# programme principal -------------------------------------------------
truc = []
machin = [0.0]*5

print("truc =", truc)
print("machin =", machin, "\n")

rien = input('"Entree"')

print("range(4) =", range(4))
print("range(4, 8) =", range(4, 8))
print("range(2, 9, 2) =", range(2, 9, 2), "\n")

rien = input('"Entree"')

chose = range(6)
print("chose =", chose)

rien = input('"Entree"')

print("Test d'appartenance de l'element 3 :", 3 in chose)
print("Test d'appartenance de l'element 6 :", 6 in chose)
```

```
# -*- coding: UTF-8 -*-
"""Liste en intension.
   Forme 1.
"""
# fichier : cours4_15.py
# auteur : Bob Cordeau

# programme principal -------------------------------------------------
result1 = []
for i in range(6):
    result1.append(i+3)

print(" boucle for ".center(50, '-'))
print(result1, '\n')

rien = input('"Entree"')

result2 = [i+3 for i in range(6)]

print(" forme 1 ".center(50, '-'))
print(result2)
```

```
# -*- coding: UTF-8 -*-
"""Liste en intension.
   Forme 2.
"""
# fichier : cours4_20.py
# auteur : Bob Cordeau

# programme principal -------------------------------------------------
```

```
result3 = []
for i in range(6):
    if i >= 2:
        result3.append(i+3)

print(" boucle for ".center(50, '-'))
print(result3, '\n')

rien = input('"Entree"')

result4 = [i+3 for i in range(6) if i >= 2]

print(" forme 2 ".center(50, '-'))
print(result4)
```

```
# -*- coding: UTF-8 -*-
"""Liste en intension.
   Forme 3.
"""
# fichier : cours4_25.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
result5 = []
for i in "abc":
    for j in "de":
        result5.append(i+j)

print(" boucle for ".center(50, '-'))
print(result5, '\n')

rien = input('"Entree"')

result6 = [i+j for i in "abc" for j in "de"]

print(" forme 3 ".center(50, '-'))
print(result6)
```

```
# -*- coding: UTF-8 -*-
"""Liste en intension.
   Calcul d'une somme.
"""
# fichier : cours4_30.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
s1 = 0
for i in range(10):
    s1 = s1 + i

print(" somme (boucle for) ".center(50, '-'))
print("somme =", s1, '\n')

rien = input('"Entree"')
```

```
s2 = sum([i for i in range(10)])

print(" somme (liste en intension) ".center(50, '-'))
print("somme =", s2)
```

```
# -*- coding: UTF-8 -*-
"""Les ensembles en Python.
"""
# fichier : cours4_35.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
X = set('abcd')
Y = set('sbds')

print(" ensembles de depart ".center(50, '-'))
print("X =", X)
print("Y =", Y)

rien = input('"Entree"')

print(" appartenance ".center(50, '-'))
print("'c' appartient a X ?", 'c' in X)
print("'a' appartient a Y ?", 'a' in Y)

rien = input('"Entree"')

print(" difference ".center(50, '-'))
print("X - Y :", X - Y)
print("Y - X :", Y - X)

rien = input('"Entree"')

print(" union ".center(50, '-'))
print("X | Y :", X | Y)

rien = input('"Entree"')

print(" intersection ".center(50, '-'))
print("X & Y :", X & Y)
```

```
# -*- coding: UTF-8 -*-
"""Les dictionnaires en Python.

   Extraire les mots d'un texte, et leur frequence.
"""
# fichier : cours4_40.py
# auteur : Bob Cordeau

# fonction
def compterMots(texte):
    dict = {}
    listeMots = texte.split()
```

```
    for mot in listeMots:
        if mot in dict:
            dict[mot] = dict[mot] + 1
        else:
            dict[mot] = 1
    return dict

# programme principal -------------------------------------------------
res = compterMots("Ala Met Asn Glu Met Cys Asn Glu Hou Ala Met Gli Asn Asn")

for c in res.keys():
    print(c, "-->", res[c])
```

```
# -*- coding: UTF-8 -*-
"""Les dictionnaires en Python (2/2)."""
# fichier : cours4_42.py
# auteur : Bob Cordeau

dico = {"Au":{"Te/Tf":(2970, 1063),
              "N/M atomique":(79, 196.967)
             },
        "Ga":{"Te/Tf":(2237, 29.8),
              "N/M atomique":(31, 69.72)
             }
       }

# programme principal -------------------------------------------------
print(dico["Au"], "\n")
print(dico["Ga"]["Te/Tf"], "\n")
print("Numero atomique de l'or :", dico["Au"]["N/M atomique"][0], "\n")
print("Masse atomique de l'or :", dico["Au"]["N/M atomique"][1])
```

```
# -*- coding: UTF-8 -*-
"""Implementation d'une pile LIFO avec une liste."""
# fichier : cours4_45.py
# auteur : Bob Cordeau

# fonctions
def pile(*args):
    p = []
    if not args:
        return p
    for elem in args:
        p.append(elem)
    return list(p)

def empile(p, a):
    p.append(a)

def depile(p):
    try:
        return p.pop()
    except:
        print("La pile est vide !")
```

```
# programme principal ------------------------------------------------
print(" Pile initiale ".center(50, '-'))
lifo = pile(5, 8, 9)
print("lifo :", lifo, '\n')

rien = input('"Entree"')

print(" Empilage ".center(50, '-'))
empile(lifo, 11)
print("lifo :", lifo, '\n')

rien = input('Entree"')

print(" Depilages ".center(50, '-'))
for i in range(5):
    depile(lifo)
    print("lifo :", lifo)
```

```
# -*- coding: UTF-8 -*-
"""Implementation d'une queue FIFO avec une liste.

   Un menu (utilisant un dictionnaire) appelle des procedures sans argument.
"""
# fichier : cours4_50.py
# auteur : Bob Cordeau

# initialisation
queue = []

# fonctions
def enQueue():
    queue.append(int(input("Entrez un entier : ")))

def deQueue():
    if len(queue) == 0:
        print("\nImpossible : la queue est vide !")
    else:
        print("\nElement '{:d}' supprime".format(queue.pop(0)))

def afficheQueue():
    print("\nqueue :", queue)

# programme principal ------------------------------------------------
CMDs = {'a':enQueue, 'v':afficheQueue, 's':deQueue}

menu = """
(A)jouter
(V)oir
(S)upprimer
(Q)uitter

   Votre choix : """
```

```
while True:
    while True:
        try:
            choix = input(menu).strip()[0].lower()
        except:
            choix = 'q'

        if choix not in 'avsq':
            print("Option invalide ! Reessayez")
        else:
            break

    if choix == 'q':
        print("\nAu revoir !")
        break
    CMDs[choix]()
```

Cours nº 5

```python
# -*- coding: UTF-8 -*-
"""Module de fonctions pour les nombres parfaits et chanceux."""
# fichier : parfait_chanceux_m.py
# auteur : Bob Cordeau

# imports
from verif_m import verif

# fonctions
def somDiv(n):
    """Retourne la somme des diviseurs propres de <n>."""
    somme = 1
    for i in range(2, n//2+1):
        if n % i == 0:
            somme += i
    return somme

def estParfait(n):
    """Teste si <n> est parfait."""
    return True if somDiv(n) == n else False

def estPremier(n):
    """Teste si <n> est premier."""
    return True if somDiv(n) == 1 else False

def estChanceux(n):
    """Teste si <n> est chanceux."""
    for i in range(0, n-1):
        if not estPremier(n + i + i**2):
            return False
    return True

# Auto-test ------------------------------------------------------------
if __name__=='__main__':
    verif(somDiv(12), 16, comment="Somme des diviseurs propres de 12 : ")
    verif(estParfait(6), True, comment="6 est-il parfait : ")
    verif(estPremier(31), True, comment="31 est-il premier : ")
    verif(estChanceux(11), True, comment="11 est-il chanceux : ")
```

```python
# -*- coding: UTF-8 -*-
"""Liste des nombres parfaits et chanceux dans [2 .. 1000]."""
# fichier : parfait_chanceux.py
# auteur : Bob Cordeau

# imports
from parfait_chanceux_m import estParfait, estChanceux

# programme principal ------------------------------------------------
parfaits, chanceux = [], []
intervalle = range(2, 1001)

for i in intervalle:
```

```
    if estParfait(i):
        parfaits.append(i)
    if estChanceux(i):
        chanceux.append(i)

msg = " Nombres remarquables dans [2 .. 1000] ".center(70, '-')
msg += "\n\nParfaits :\t{}\n\nChanceux :\t{}".format(parfaits, chanceux)
print(msg)
```

Cours nº 6

```
# -*- coding: UTF-8 -*-
"""Module de calcul des racines reelles d'un trineme."""
# fichier : cours6_05.py
# auteur : Bob Cordeau

# import
from math import sqrt

# fonction
def trinome(a, b, c):
    delta = b**2 - 4*a*c

    if delta > 0.0:
        racine_delta = sqrt(delta)
        return (2, (-b-racine_delta)/(2*a), (-b+racine_delta)/(2*a))
    elif delta < 0.0:
        return (0,)
    else:
        return (1, (-b/(2*a)))

if __name__ == "__main__":
    print(trinome(1.0, -3.0, 2.0))
    print(trinome(1.0, -2.0, 1.0))
    print(trinome(1.0, 1.0, 1.0))
```

```
# -*- coding: UTF-8 -*-
"""Calcul des racines reelles d'un trinome."""
# fichier : cours6_10.py
# auteur : Bob Cordeau

# import
from cours6_05 import trinome
from easygui import floatbox, msgbox

# programme principal ------------------------------------------------
titre = "Resolution du trinome a*x**2 + b*x + c = 0"
a = floatbox("a : ", titre, 1.0, -100.0, 100.0)
b = floatbox("b : ", titre, 2.0, -100.0, 100.0)
c = floatbox("c : ", titre, -1.0, -100.0, 100.0)

solutions = trinome(a, b, c)

if solutions[0] == 2:
    msg = "x1 = {:g} et x2 = {:g}".format(solutions[1], solutions[2])
elif solutions[0] == 1:
    msg = "x1 = x2 = {:g}".format(solutions[1])
else:
    msg = "Pas de racine reelle."

msgbox(msg, "Racines du trinome")
```

Cours nº 7

```
# -*- coding: UTF-8 -*-
"""Syntaxe objet."""
# fichier : cours7_05.py
# auteur : Bob Cordeau

# classe
class MaClasse:
    """Definition d'une classe."""
    x = 23
    y = x + 5  # x et y : attributs de classe

    def affiche(self):
        """Definition d'une methode."""
        self.z = 42   # attribut d'instance (i.e. de l'objet self)
        print(MaClasse.y, end=" ") # dans une methode, on qualifie un attribut de
            classe...
        print(self.z)  # ...mais pas un attribut d'instance

# programme principal -------------------------------------------
obj = MaClasse()
obj.affiche() # a l'appel, self = obj. Idem : C.affiche(obj)
```

```
# -*- coding: UTF-8 -*-
"""Classe avec constructeur."""
# fichier : cours7_10.py
# auteur : Bob Cordeau

# classe
class Vecteur2D:
    """Les Vecteurs plans."""

    def __init__(self, x0=0, y0=0):
        """Constructeur avec parametres par defaut."""
        self.x = x0 # initialisation de x et y, attributs d'instance
        self.y = y0

# programme principal -------------------------------------------
print(" une instance par defaut ".center(50, '-'))
v1 = Vecteur2D()
print("x = %g, y = %g" % (v1.x, v1.y))
print()

print(" une instance initialisee ".center(50, '-'))
v2 = Vecteur2D(-5.2, 4.1)
print("x = %g, y = %g" % (v2.x, v2.y))
```

```
# -*- coding: UTF-8 -*-
"""Surcharge des operateurs."""
# fichier : cours7_15.py
# auteur : Bob Cordeau

# classe
```

Solutions

```
class Vecteur2D:
    """Definition d'une classe."""

    def __init__(self, x0=0, y0=0):
        """Constructeur avec parametres par defaut."""
        self.x = x0 # initialisation de x et y, attributs d'instance
        self.y = y0

    def __add__(self, autre):
        """Addition vectorielle."""
        return Vecteur2D(self.x+autre.x, self.y+autre.y)

    def __str__(self):
        """Affichage d'un Vecteur2D."""
        return "Vecteur({:g}, {:g})".format(self.x, self.y)

# programme principal ------------------------------------------------
v1, v2 = Vecteur2D(1.2, 2.3), Vecteur2D(3.4, 4.5)

print(v1)
print(v2)
print(v1 + v2)
```

Cours nº 8

```
# -*- coding: UTF-8 -*-
"""module d'exemple de polymorphisme."""
# fichier : cours8_05.py
# auteur : Bob Cordeau

# classes
class Rectangle:
    """classe des rectangles."""
    def __init__(self, longueur=30, largeur=15):
        "Initialisation avec valeurs par defaut"
        self.lon = longueur
        self.lar = largeur
        self.nom = "rectangle"

    def surface(self):
        "Retourne la surface d'un rectangle."
        return self.lon*self.lar

    def __str__(self):
        "Affichage des caracteristiques d'un rectangle."
        return ("\nLe {} de cÃ´tes {} et {} a une surface de {}"
                    .format(self.nom, self.lon, self.lar, self.surface()))

class Carre(Rectangle):
    """classe des carres (herite de Rectangle)."""
    def __init__(self, cote=10):
        "Constructeur avec valeur par defaut"
        Rectangle.__init__(self, cote, cote)
        self.nom = "carre" # surchage d'attribut d'instance

# Auto-test ------------------------------------------------------------
if __name__ == '__main__':
    r = Rectangle(12, 8)
    print(r)

    c = Carre()
    print(c)
```

```
# -*- coding: UTF-8 -*-
"""Module d'exemple de composition."""
# fichier : cours8_10.py
# auteur : Bob Cordeau

# classes
class Point:
    """classe des points du plan."""

    def __init__(self, x=0.0, y=0.0):
        "Initialisation avec valeurs par defaut"
        self.px = float(x)
        self.py = float(y)
```

```
class Segment:
    """classe composite utilisant la classe Point."""

    def __init__(self, x1, y1, x2, y2):
        "L'initialisation utilise deux objets Point"
        self.orig = Point(x1, y1)
        self.extrem = Point(x2, y2)

    def __str__(self):
        """Representation d'un objet segment."""
        return ("Segment : [({:g}, {:g}), ({:g}, {:g})]"
            .format(self.orig.px, self.orig.py, self.extrem.px, self.extrem.py))

# Auto-test ------------------------------------------------------------
if __name__ == '__main__':
    s = Segment(1, 2, 3, 4)
    print(s)
```

```
# -*- coding: UTF-8 -*-
"""Idiome de la fonction fabrique renvoyant une fermeture."""
# fichier : cours8_30.py
# auteur : Bob Cordeau

# fonctions
def creer_plus(ajout):
    def plus(increment):
        """
        Fonction 'fermeture' : utilise des noms locaux a la fonction enveloppante.
        """
        return increment + ajout
    return plus

# Programme principal --------------------------------------------
print(" creation de deux fabriques distinctes ". center(50, '-'))
p = creer_plus(23)
q = creer_plus(42)
print(p(100), q(100))
```

```
# -*- coding: UTF-8 -*-
"""Fonction fabrique renvoyant une classe."""
# fichier : cours8_40.py
# auteur : Bob Cordeau

# classes
class CasNormal:
    def uneMethode(self):
        print("normal")

class CasSpecial:
    def uneMethode(self):
        print("special")

# Fonction fabrique renvoyant une classe
def casQuiConvient(estNormal=True):
```

```
    return CasNormal() if estNormal else CasSpecial()

# Programme principal -------------------------------------------------
uneInstance = casQuiConvient(estNormal=False)
uneInstance.uneMethode()
```

Scripts supplémentaires

```
# -*- coding: UTF-8 -*-
"""Volume d'un cone."""
# fichier : exo_005.py
# auteur : Bob Cordeau

# imports
from math import pi

# programme principal -------------------------------------------------
rayon = float(input("Rayon du cone (m) : "))
hauteur = float(input("Hauteur du cone (m) : "))

volume = (pi*rayon*rayon*hauteur)/3.0
print("Volume du cone = {} m3".format(volume))
```

```
# -*- coding: UTF-8 -*-
"""Calcul d'un prix TTC."""
# fichier : exo_010.py
# auteur : Bob Cordeau

# programme principal -------------------------------------------------
prixHT = float(input("Prix HT (0 pour terminer)? "))

while prixHT > 0:
    print("Prix TTC : {:.2f}\n".format(prixHT * 1.196))
    prixHT = float(input("Prix HT (0 pour terminer)? "))

print("Au revoir !")
```

```
# -*- coding: UTF-8 -*-
"""Somme d'entiers et nombre d'entiers superieur a 100."""
# fichier : exo_015.py
# auteur : Bob Cordeau

# programme principal -------------------------------------------------
somme, nombre_total, nombre_grands = 0, 0, 0

x = int(input("x (0 pour terminer) ? "))
while x > 0:
    somme = somme + x
    nombre_total = nombre_total + 1
    if x > 100:
        nombre_grands = nombre_grands + 1
    x = int(input("x (0 pour terminer) ? "))

print("\nSomme :", somme)
print("{} valeur(s) en tout, dont {} superieure(s) Ã  100".format(nombre_total,
                                                                 nombre_grands))
```

```
# -*- coding: UTF-8 -*-
"""Parite."""
# fichier : exo_020.py
```

```
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
n = int(input("Entrez un entier strictement positif : "))
while n < 1:
    n = int(input("Entrez un entier STRICTEMENT POSITIF, s.v.p. : "))

if n%2 == 0:
    print(n, "est pair.")
else:
    print(n, "est impair.")
```

```
# -*- coding: UTF-8 -*-
"""Nombre de fois qu'un entier est divisible par 2."""
# fichier : exo_025.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
n = int(input("Entrez un entier strictement positif : "))
while n < 1:
    n = int(input("Entrez un entier STRICTEMENT POSITIF, s.v.p. : "))
save = n

cpt = 0
while n%2 == 0:
    n /= 2
    cpt += 1

print("{} est {} fois divisible par 2.".format(save, cpt))
```

```
# -*- coding: UTF-8 -*-
"""Diviseurs propres d'un entier."""
# fichier : exo_030.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
n = int(input("Entrez un entier strictement positif : "))
while n < 1:
    n = int(input("Entrez un entier STRICTEMENT POSITIF, s.v.p. : "))

i = 2      # plus petit diviseur possible de n
cpt = 0    # initialise le compteur de divisions
p = n/2    # calcule une fois dans la boucle

print("Diviseurs propres sans repetition de {} : ".format(n), end=" ")
while i <= p:
    if n%i == 0:
        cpt += 1
        print(i, end=" ")
    i += 1

if not cpt:
    print("aucun : il est premier.")
else:
```

```
    print("(soit {} diviseurs propres)".format(cpt))
```

```
# -*- coding: UTF-8 -*-
"""Approximation de 'e'."""
# fichier : exo_035.py
# auteur : Bob Cordeau

# fonctions
def fact(n):
    r = 1
    for i in range(1, n+1):
        r *= i
    return r

# programme principal ------------------------------------------
n = int(input("n ? "))
exp = 0.0
for i in range(n):
    exp = exp + 1.0/fact(i)

print("Approximation de 'e' : {:.3f}".format(exp))
```

```
# -*- coding: UTF-8 -*-
"""Gardien de phare."""
# fichier : exo_040.py
# auteur : Bob Cordeau

# fonctions
def hauteurParcourue(nb, h):
    print("{} marches de {} cm. Il parcourt {:.2f} m par semaine.".format(nb,
                                                    h, nb*h*2*5*7/100.0))

# programme principal ------------------------------------------
nb_marches = int(input("Combien de marches ? "))
hauteur_marche = int(input("Hauteur d'une marche (cm) ? "))

hauteurParcourue(nb_marches, hauteur_marche)
```

```
# -*- coding: UTF-8 -*-
"""Permis de chasse."""
# fichier : exo_045.py
# auteur : Bob Cordeau

# fonctions
def permisSup(p, c, v, a):
    points_perdus = p + 3*c + 5*v + 10*a
    nbre_permis = points_perdus/100.0
    return 200*nbre_permis

# programme principal ------------------------------------------
poules = int(input("Combien de poules ? "))
chiens = int(input("Combien de chiens ? "))
vaches = int(input("Combien de vaches ? "))
amis = int(input("Combien d'amis ? "))
```

```
payer = permisSup(poules, chiens, vaches, amis)

print("\nA payer :", end=" ")
print("rien a payer") if payer == 0 else print(payer, "euros")
```

```
# -*- coding: UTF-8 -*-
"""Histoire de train."""
# fichier : exo_050.py
# auteur : Bob Cordeau

# fonctions
def tchacatchac(v):
    """Affiche l'heure du drame."""
    heure = 9 + int(170/v)
    minute = (60 * 170 / v) % 60
    print("A {} km/h, je me fais dechiqueter aÂ  {} h {:.2f} min.".format(v,
                                                        heure, minute))

# programme principal ------------------------------------------------
i = 100
while i <= 300:
    tchacatchac(i)
    i += 10
```

```
# -*- coding: UTF-8 -*-
"""Proportion d'une sÃ©quence dans une chaine d'ADN."""
# fichier : exo_055.py
# auteur : Bob Cordeau

# fonctions
def valide(seq):
    """Retourne VRAI si la sequence est valide, FAUX sinon."""
    ret = len(seq) != 0
    for c in seq:
        ret = (ret and True) \
              if (c == 'a') or (c == 't') or (c == 'g') or (c == 'c') \
              else False
    return ret

def proportion(a, s):
    """Retourne la proportion de la sequence <s> dans la chaine <a>."""
    n = len(a)
    k = a.count(s)
    return 100.0*k/n

def saisie(c):
    s = input("{} : ".format(c))
    while not valide(s):
        print('"{}" ne peut contenir que les chainons {}'.format(c,
                                                "'a', 't', 'g' ou 'c'"))
        s = input("{} : ".format(c))
    return s
```

```
# programme principal -------------------------------------------------
adn = saisie("chaine")
seq = saisie("sequence")

print('Il y a {:.1f} % de "{}" dans "{}".'.format(proportion(adn, seq),
                                                  seq, adn))
```

```
# -*- coding: UTF-8 -*-
"""Echanges."""
# fichier : exo_060.py
# auteur : Bob Cordeau

# imports
from random import seed, randint

# fonctions
def listAleaInt(n, a, b):
    """Retourne une liste de <n> entiers aleatoires dans [<a> .. <b>]."""
    return [randint(a, b) for i in range(n)]

# programme principal -------------------------------------------------
seed() # initialise le generateur de nombres aleatoires
t = listAleaInt(100, 2, 125) # construction de la liste

## calcul de l'indice du minimum de la liste
iMin = t.index(min(t))

## Affichages
print("{} t[0] = {},\tt[iMin] = {}".format("Avant echange :", t[0], t[iMin]))
t[0], t[iMin] = t[iMin], t[0]
print("{} t[0] = {},\tt[iMin] = {}".format("AprÃ¨s echange :", t[0], t[iMin]))
```

```
# -*- coding: UTF-8 -*-
"""Amplitude et moyenne d'une liste de flottants."""
# fichier : exo_065.py
# auteur : Bob Cordeau

# imports
from random import seed, random

# fonctions
def listAleaFloat(n):
    """Retourne une liste de <n> flottants aleatoires."""
    return [random() for i in range(n)]

# programme principal -------------------------------------------------
n = int(input("Entrez un entier [2 .. 100] : "))
while not (2 <= n <= 100): # saisie filtree
    n = int(input("Entrez un entier [2 .. 100], s.v.p. : "))

seed() # initialise le generateur de nombres aleatoires
t = listAleaFloat(n)  # construction de la liste

print("Liste :", t)
```

```
print("Amplitude : {:.2f}".format(max(t) - min(t)))
print("Moyenne : {:.2f}".format(sum(t)/n))
```

```
# -*- coding: UTF-8 -*-
"""Conversions de temperatures."""
# fichier : exo_070.py
# auteur : Bob Cordeau

# fonctions
def conv(t, n):
    """Renvoie la conversion Celsius -> Fahrenheit ou inversement suivant <n>."""
    if n == 1:     # Celsius -> Fahrenheit
        return (32.0 + 1.8*t)
    elif n == 2:   # Fahrenheit -> Celsius
        return ((t-32.0)/1.8)

# programme principal -------------------------------------------------
t = float(input("Temperature ? "))
u = float(input("Unite de depart (1 = Celsius, 2 = Fahrenheit) ? "))
while not (1 <= u <= 2):
    u = float(input("Unite de depart (1 = Celsius, 2 = Fahrenheit), SVP ? "))

unit = {1: 'Â°C', 2: 'Â°F'} # dictionnaire
print("\n{:.2f} {} = {:.2f} {}".format(t, unit[u], conv(t, u), unit[(u%2)+1]))
```

```
# -*- coding: UTF-8 -*-
"""Min, max et moyenne d'une liste d'entiers."""
# fichier : exo_075.py
# auteur : Bob Cordeau

# fonctions
def minMaxMoy(liste):
    """Renvoie le min, le max et la moyenne de la liste."""
    n = len(liste)
    if n == 0:
        return None
    min = max = som = liste[0]
    for i in liste[1:]:
        if i < min:
            min = i
        if i > max:
            max = i
        som = som + i
    return (min, max, som/float(n))

# programme principal -------------------------------------------------
lp = [10, 18, 14, 20, 12, 16]

print("Liste =", lp)
min, max, moy = minMaxMoy(lp)
print("min : {}, max : {}, moyenne : {:.2f}".format(min, max, moy))
```

```
# -*- coding: UTF-8 -*-
"""Nombres romains (version 1)."""
```

```
# fichier : exo_080.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
n = int(input('Entrez un entier [1 .. 4000[ : '))
while not(1 <= n <= 3999):
    n = int(input('Entrez un entier [1 .. 4000[, s.v.p. : '))

s = "" # Chaine resultante

while n >= 1000:
    s += "M"
    n -= 1000

if n >= 900:
    s += "CM"
    n -= 900

if n >= 500:
    s += "D"
    n -= 500

if n >= 400:
    s += "CD"
    n -= 400

while n >= 100:
    s += "C"
    n -= 100

if n >= 90:
    s += "XC"
    n -= 90

if n >= 50:
    s += "L"
    n -= 50

if n >= 40:
    s += "XL"
    n -= 40

while n >= 10:
    s += "X"
    n -= 10

if n >= 9:
    s += "IX"
    n -= 9

if n >= 5:
    s += "V"
    n -= 5
```

```
if n >= 4:
    s += "IV"
    n -= 4

while n >= 1:
    s += "I"
    n -= 1

print("En romain :", s)
```

```
# -*- coding: UTF-8 -*-
"""Nombres romains (version 2)."""
# fichier : exo_085.py
# auteur : Bob Cordeau

# globales
code = zip(
    [1000,900 ,500,400 ,100,90 ,50 ,40 ,10 ,9 ,5 ,4   ,1],
    ["M" ,"CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"]
)

# fonctions
def decToRoman(num):
    res = []
    for d, r in code:
        while num >= d:
            res.append(r)
            num -= d
    return ''.join(res)

# programme principal -------------------------------------------------
for i in range(1, 4000):
    print(i, decToRoman(i))
```

```
# -*- coding: UTF-8 -*-
"""Liste d'entiers differents."""
# fichier : exo_090.py
# auteur : Bob Cordeau

# imports
from random import seed, randint

# fonctions
def listAleaInt(n, a, b):
    """Retourne une liste de <n> entiers aleatoires entre <a> et <b>."""
    return [randint(a, b) for i in range(n)]

# programme principal -------------------------------------------------
N = 100
n = int(input("Entrez un entier [1 .. 100] : "))
while not(1 <= n <= N):
    n = int(input("Entrez un entier [1 .. 100], s.v.p. : "))
```

```
# construction de la liste
seed() # initialise le generateur de nombres aleatoires
t = listAleaInt(n, 0, 500)

# Sont-ils differents ?
tousDiff = True
i = 0
while tousDiff and i<(n-1):
    j = i + 1
    while tousDiff and j<n:
        if t[i] == t[j]:
            tousDiff = False
        else:
            j += 1
    i += 1

print("Liste :", t)
print("==> Tous les elements sont distincts.") if tousDiff else print("==> Au moins
    une valeur est repetee.")
```

```
# -*- coding: UTF-8 -*-
"""Jeu de des (1)."""
# fichier : exo_095.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
n = int(input("Entrez un entier [2 .. 12] : "))
while not(2 <= n <= 12):
    n = int(input("Entrez un entier [2 .. 12], s.v.p. : "))

s = 0
for i in range(1, 7):
    for j in range(1, 7):
        if i+j == n:
            s += 1

print("Il y a {} facon(s) de faire {} avec deux des.".format(s, n))
```

```
# -*- coding: UTF-8 -*-
"""Jeu de des (2)."""
# fichier : exo_100.py
# auteur : Bob Cordeau

# programme principal ------------------------------------------------
n = int(input("Entrez un entier [3 .. 18] : "))
while not(3 <= n <= 18):
    n = int(input("Entrez un entier [3 .. 18], s.v.p. : "))

s = 0
for i in range(1, 7):
    for j in range(1, 7):
        for k in range(1, 7):
            if i+j+k == n:
                s += 1
```

```
print("Il y a {} facon(s) de faire {} avec trois des.".format(s, n))
```

```
# -*- coding: UTF-8 -*-
"""Jeu de des (3)."""
# fichier : exo_105.py
# auteur : Bob Cordeau

# globales
MAX = 8

# programme principal -------------------------------------------------
nbd = int(input("Nombre de des entre 2 et {} : ".format(MAX)))
while not(2 <= nbd <= MAX):
    nbd = int(input("Nombre de des entre 2 et {}, s.v.p. : ".format(MAX)))

s = int(input("Entrez un entier entre {} et {} : ".format(nbd, 6*nbd)))
while not(nbd <= s <= 6*nbd):
    s = int(input("Entrez un entier entre {} et {}, s.v.p. : ".format(nbd, 6*nbd)))

if s == nbd or s == 6*nbd:
    cpt = 1 # 1 seule solution
else:
    I = [1]*nbd # initialise une liste de <nbd> des
    cpt, j = 0, 0
    while j < nbd:
        som = sum([I[k] for k in range(nbd)])

        if som == s:
            cpt += 1 # compteur de bonnes solutions
        if som == 6*nbd:
            break

        j = 0
        if I[j] < 6:
            I[j] += 1
        else:
            while I[j] == 6:
                I[j] = 1
                j += 1
            I[j] += 1

print("Il y a {} facons de faire {} avec {} des.".format(cpt, s, nbd))
```

```
# -*- coding: UTF-8 -*-
"""Jeu de des (recursif)."""
# fichier : exo_110.py
# auteur : Bob Cordeau

# globales
MAX = 8

# fonctions
def calcul(d, n):
```

```
    """Calcul recursif du nombre de facons de faire <n> avec <d> des."""
    resultat, debut = 0, 1
    if d == 1 or n == d or n == 6*d: # conditions terminales
        return 1
    else:                # sinon appels recursifs
        if n > 6*(d-1):  # optimisation importante
            debut = n - 6*(d-1)

        for i in range(debut, 7):
            if n == i:
                break
            resultat += calcul(d-1, n-i)
    return resultat

# programme principal -------------------------------------------------
nbd = int(input("Nombre de des entre 2 et {} : ".format(MAX)))
while not(2 <= nbd <= MAX):
    nbd = int(input("Nombre de des entre 2 et {}, s.v.p. : ".format(MAX)))

s = int(input("Entrez un entier entre {} et {} : ".format(nbd, 6*nbd)))
while not(nbd <= s <= 6*nbd):
    s = int(input("Entrez un entier entre {} et {}, s.v.p. : ".format(nbd, 6*nbd)))
print("Il y a {} facon(s) de faire {} avec {} des.".format(calcul(d, n), n, d))
```

```
# -*- coding: UTF-8 -*-
"""Calculs de matrices."""
# fichier : exo_115.py
# auteur : Bob Cordeau

# Procedure
def affiche(m):
    for i in range(n):
        print("\t", m[i])

# programme principal -------------------------------------------------
n = int(input("Dimension des matrices carrees [2 .. 10] : "))
while not(2 <= n <= 10):
    n = int(input("Dimension des matrices carrees [2 .. 10], SVP : "))

# initialisation des 3 matrices
m1, m2, m3 = [0]*n, [0]*n, [0]*n
for i in range(n):
    m1[i], m2[i], m3[i] = [0]*n, [0]*n, [0]*n

# calcul des matrices
k = 2
for i in range(n):
    for j in range(n):
        m1[i][j] = k   # matrice d'elements pairs
        k += 2

        if i == j:
            m2[i][j] = 1   # matrice unite
```

```
        m3[i][j] = m1[i][j] - m2[i][j]

# Affichages
print("m1 :")
affiche(m1)
print("\nm2 :")
affiche(m2)
print("\nm3 = m1 - m2 :")
affiche(m3)
```

Colophon

Ces exercices ont été composés grâce au logiciel LaTeX sous l'éditeur TeXnicCenter. Le corps du texte est en police **Utopia**, les exemples de code en police `Typewriter`.

Ce document est disponible à l'adresse :

`www.iut-orsay.fr/dptmphy/Pedagogie/Welcome.html`