Exercices

2.1 Décrivez le plus clairement et le plus complètement possible ce qui se passe a chacune des trois lignes de l'exemple ci-dessous :

```
>>> largeur = 20
>>> hauteur = 5 * 9.3
>>> largeur * hauteur
930
```

2.2 Assignez les valeurs respectives 3, 5, 7 a trois variables a, b, c.
Effectuez l'opération *a - b//c*. Interprétez le résultat obtenu.

2.3 Testez les lignes d'instructions suivantes. Décrivez dans votre cahier ce qui se passe :

```
>>> r, pi = 12, 3.14159
>>> s = pi * r**2
>>> print(s)
>>> print(type(r), type(pi), type(s))
```

Quelle est, à votre avis, l'utilité de la *fonction* **type()** ?
(Note : les *fonctions* seront décrites en détail aux chapitres 6 et 7.)

Exercices

4.1 Ecrivez les lignes d'instructions nécessaires pour obtenir ce résultat.
A la suite de l'exercice propose ci-dessus, vous aurez certainement trouve une méthode, et un Professeur vous demanderait certainement de la commenter en classe. Comme il s'agit d'une opération courante, les langages de programmation proposent souvent des raccourcis pour l'effectuer (par exemple des instructions spécialisées, telle l'instruction SWAP du langage *Basic*). Sous Python, *l'affectation parallèle* permet de programmer l'échange d'une manière particulièrement élégante :

```
>>> a, b = b, a
```

(On pourrait bien entendu échanger d'autres variables en même temps, dans la même instruction.)

4.2 Ecrivez un programme qui affiche les 20 premiers termes de la table de multiplication par 7.

4.3 Ecrivez un programme qui affiche une table de conversion de sommes d'argent exprimées en euros, en dollars canadiens. La progression des sommes de la table sera ≪ géométrique≫, comme dans l'exemple ci-dessous :

```
1 euro(s) = 1.65 dollar(s)
2 euro(s) = 3.30 dollar(s)
4 euro(s) = 6.60 dollar(s)
8 euro(s) = 13.20 dollar(s)
```

etc. (S'arrêter à 16384 euros.)

4.4 Ecrivez un programme qui affiche une suite de 12 nombres dont chaque terme soit égal au triple du terme précèdent.

4.5 Ecrivez un programme qui calcule le volume d'un parallélépipède rectangle dont sont fournis au départ la largeur, la hauteur et la profondeur.

4.6 Ecrivez un programme qui convertit un nombre entier de secondes fourni au départ en un nombre d'années, de mois, de jours, de minutes et de secondes (utilisez l'opérateur modulo : **%**).

4.7 Ecrivez un programme qui affiche les 20 premiers termes de la table de multiplication par 7, en signalant au passage (à l'aide d'une astérisque) ceux qui sont des multiples de 3.

Exemple : 7 14 21 * 28 35 42 * 49 ...

4.8 Ecrivez un programme qui calcule les 50 premiers termes de la table de multiplication par 13, mais n'affiche que ceux qui sont des multiples de 7.

4.9 Ecrivez un programme qui affiche la suite de symboles suivante :

```
*
**
***
****
*****
******
*******
```

Exercices

5.1 Ecrivez un programme qui convertisse en radians un angle fourni au départ en degrés, minutes, secondes.

5.2 Ecrivez un programme qui convertisse en degrés, minutes, secondes un angle fourni au départ en radians.

5.3 Ecrivez un programme qui convertisse en degrés Celsius une température exprimée au départ en degrés Fahrenheit, ou l'inverse. La formule de conversion est : $T F=TC \times 1,8+32$.

5.4 Ecrivez un programme qui calcule les intérêts accumules chaque année pendant 20 ans, par capitalisation d'une somme de 100 euros placée en banque au taux fixe de 4,3 %.

5.5 Une légende de l'Inde ancienne raconte que le jeu d'échecs a été inventé par un vieux sage, que son roi voulut remercier en lui affirmant qu'il lui accorderait n'importe quel cadeau en récompense. Le vieux sage demanda qu'on lui fournisse simplement

un peu de riz pour ses vieux jours, et plus précisément un nombre de grains de riz suffisant pour que l'on puisse en déposer 1 seul sur la première case du jeu qu'il venait d'inventer, deux sur la suivante, quatre sur la troisième, et ainsi de suite jusqu'à la 64e case.
Ecrivez un programme Python qui affiche le nombre de grains à déposer sur chacune des 64 cases du jeu. Calculez ce nombre de deux manières :
• le nombre exact de grains (nombre entier) ;
• le nombre de grains en notation scientifique (nombre réel).

5.6 Ecrivez un script qui détermine si une chaine contient ou non le caractère ≪ e ≫.

5.7 Ecrivez un script qui compte le nombre d'occurrences du caractère ≪ e ≫ dans une chaine.
5.8 Ecrivez un script qui recopie une chaine (dans une nouvelle variable), en insérant des astérisques entre les caractères.
Ainsi par exemple, ≪ **gaston** ≫ devra devenir ≪ **g*a*s*t*o*n** ≫
5.9 Ecrivez un script qui recopie une chaine (dans une nouvelle variable) en l'inversant.
Ainsi par exemple, ≪ **zorglub** ≫ deviendra ≪ **bulgroz** ≫.

5.10 En partant de l'exercice précèdent, écrivez un script qui détermine si une chaine de caractères donnée est un *palindrome* (c'est-à-dire une chaine qui peut se lire indifféremment dans les deux sens), comme par exemple ≪ radar ≫ ou ≪ s.o.s. ≫.

5.11 Soient les listes suivantes :

t1 = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
t2 = ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin',
'Juillet', 'Aout', 'Septembre', 'Octobre', 'Novembre', 'Décembre']
Ecrivez un petit programme qui crée une nouvelle liste **t3**. Celle-ci devra contenir tous les éléments des deux listes en les alternant, de telle manière que chaque nom de mois soit suivi du nombre de jours correspondant :
['Janvier',31,'Fevrier',28,'Mars',31, etc...].

5.12 Ecrivez un programme qui affiche ≪ proprement ≫ tous les éléments d'une liste.
Si on l'appliquait par exemple a la liste **t2** de l'exercice ci-dessus, on devrait obtenir :
Janvier Février Mars Avril Mai Juin Juillet Aout Septembre Octobre
Novembre Décembre
5.13 Ecrivez un programme qui recherche le plus grand élément présent dans une liste donnée. Par exemple, si on l'appliquait a la liste **[32, 5, 12, 8, 3, 75, 2, 15]**, ce programme devrait afficher :
le plus grand élément de cette liste a la valeur 75.

5.14 Ecrivez un programme qui analyse un par un tous les éléments d'une liste de nombres (par exemple celle de l'exercice précèdent) pour générer deux nouvelles listes. L'une contiendra seulement les nombres *pairs* de la liste initiale, et l'autre les nombres *impairs*.
Par exemple, si la liste initiale est celle de l'exercice précèdent, le programme devra Construire une liste **pairs** qui contiendra **[32, 12, 8, 2]**, et une liste **impairs** qui contiendra **[5, 3, 75, 15]**. Astuce : pensez à utiliser l'opérateur **modulo** (%) déjà cité précédemment.
5.15 Ecrivez un programme qui analyse un par un tous les éléments d'une liste de mots (par exemple : **['Jean', 'Maximilien', 'Brigitte', 'Sonia', 'Jean-Pierre', 'Sandra']**) pour générer deux nouvelles listes. L'une contiendra les mots comportant moins de 6 caractères, l'autre les mots comportant 6 caractères ou davantage.

Exercices
Note : *dans tous ces exercices, utilisez la fonction* ***input()*** *pour l'entrée des données.*
6.1 Ecrivez un programme qui convertisse en mètres par seconde et en km/h une vitesse fournie par l'utilisateur en miles/heure. (*Rappel : 1 mile = 1609 mètres*).
6.2 Ecrivez un programme qui calcule le périmètre et l'aire d'un triangle quelconque dont l'utilisateur fournit les 3 cotes.
(Rappel : l'aire d'un triangle quelconque se calcule à l'aide de la formule :

$$S=\sqrt{d\cdot(d-a)\cdot(d-b)\cdot(d-c)}$$

dans laquelle d désigne la longueur du demi-périmètre, et a, b, c celles des trois côtés.)
6.3 Ecrivez un programme qui calcule la période d'un pendule simple de longueur donnée. La formule qui permet de calculer la période d'un pendule simple est

$$T=2\pi\sqrt{\frac{l}{g}}$$

l représentant la longueur du pendule et ***g*** la valeur de l'accélération de la pesanteur au lieu d'expérience.
6.4 Ecrivez un programme qui permette d'encoder des valeurs dans une liste. Ce programme devrait fonctionner en boucle, l'utilisateur étant invite à entrer sans cesse de nouvelles valeurs, jusqu'à ce qu'il décide de terminer en frappant <*Enter*> en guise d'entrée. Le programme se terminerait alors par l'affichage de la liste.
Exemple de fonctionnement:
Veuillez entrer une valeur : 25
Veuillez entrer une valeur : 18
Veuillez entrer une valeur : 6284
Veuillez entrer une valeur :

[25, 18, 6284]

6.5 Que fait le programme ci-dessous, dans les quatre cas où l'on aurait défini au préalable que la variable **a** vaut 1, 2, 3 ou 15 ?

```
if a !=2:
print('perdu')
elif a ==3:
print('un instant, s.v.p.')
else :
print('gagne')
```

6.6 Que font ces programmes ?

a) a = 5

```
b = 2
if (a==5) & (b<2):
print('"&" signifie "et"; on peut aussi utiliser\
le mot "and"')
```

b) a, b = 2, 4

```
if (a==4) or (b!=4):
print('gagne')
elif (a==4) or (b==4):
print('presque gagne')
```

c) a = 1

```
if not a:
print('gagne')
elif a:
print('perdu')
```

6.7 Reprendre le programme c) avec a = 0 au lieu de a = 1. Que se passe-t-il ? Conclure !

6.8 Ecrire un programme qui, étant données deux bornes entièrs a et b, additionne les nombres multiples de 3 et de 5 compris entre ces bornes. Prendre par exemple a= 0, b = 32 ; le résultat devrait être alors 0 + 15 + 30 = 45.

Modifier légèrement ce programme pour qu'il additionne les nombres multiples de 3 ou de 5 compris entre les bornes a et b. Avec les bornes 0 et 32, le résultat devrait donc être : 0 + 3 + 5 + 6 + 9 + 10 + 12 + 15 + 18 + 20 + 21 + 24 + 25 + 27 + 30 = 225.

6.9 Déterminer si une année (dont le millésime est introduit par l'utilisateur) est bissextile ou non. Une année A est bissextile si A est divisible par 4. Elle ne l'est cependant pas si A est un multiple de 100, à moins que A ne soit multiple de 400.

6.10 Demander à l'utilisateur son nom et son sexe (M ou F). En fonction de ces données, afficher ≪ Cher Monsieur ≫ ou ≪ Chère Mademoiselle ≫ suivi du nom de la personne.
6.11 Demander à l'utilisateur d'entrer trois longueurs a, b, c. A l'aide de ces trois longueurs, déterminer s'il est possible de construire un triangle. Déterminer ensuite si ce triangle est rectangle, isocèle, équilatéral ou quelconque. Attention : un triangle rectangle peut être isocèle.
6.12 Demander à l'utilisateur qu'il entre un nombre. Afficher ensuite : soit la racine carrée de ce nombre, soit un message indiquant que la racine carrée de ce nombre ne peut être calculée.
6.13 Convertir une note scolaire N quelconque, entrée par l'utilisateur sous forme de points (par exemple 27 sur 85), en une note standardisée suivant le code ci-dessous :

Note Appreciation
N >= 80 % A
80 % > N >= 60 % B
60 % > N >= 50 % C
50 % > N >= 40 % D
N < 40 % E

6.14 Soit la liste suivante :

['Jean-Michel', 'Marc', 'Vanessa', 'Anne', 'Maximilien',
'Alexandre-Benoit', 'Louise']

Ecrivez un script qui affiche chacun de ces noms avec le nombre de caractères correspondant.
6.15 Ecrire une boucle de programme qui demande à l'utilisateur d'entrer des notes d'élevés. La boucle se terminera seulement si l'utilisateur entre une valeur négative.
Avec les notes ainsi entrées, construire progressivement une liste. Après chaque entrée d'une nouvelle note (et donc à chaque itération de la boucle), afficher le nombre de notes entrées, la note la plus élevée, la note la plus basse, la moyenne de toutes les notes.
6.16 Ecrivez un script qui affiche la valeur de la force de gravitation s'exerçant entre deux masses de 10 000 kg, pour des distances qui augmentent suivant une progression géométrique de raison 2, à partir de 5 cm (0,05 mètre).

La force de gravitation est régie par la formule F=6,67 .10^{-11}·(m·m')/d^2

Exemple d'affichage :

d = .05 m : la force vaut 2.668 N
d = .1 m : la force vaut 0.667 N

d = .2 m : la force vaut 0.167 N
d = .4 m : la force vaut 0.0417 N
etc.

Exercices

7.1 Importez le module ***turtle*** pour pouvoir effectuer des dessins simples.
Vous allez dessiner une série de triangles équilatéraux de différentes couleurs.
Pour ce faire, définissez d'abord une fonction **triangle()** capable de dessiner un triangle d'une couleur bien déterminée (ce qui signifie donc que la définition de votre fonction doit comporter un paramètre pour recevoir le nom de cette couleur). Utilisez ensuite cette fonction pour reproduire ce même triangle en différents endroits, en changeant de couleur à chaque fois.

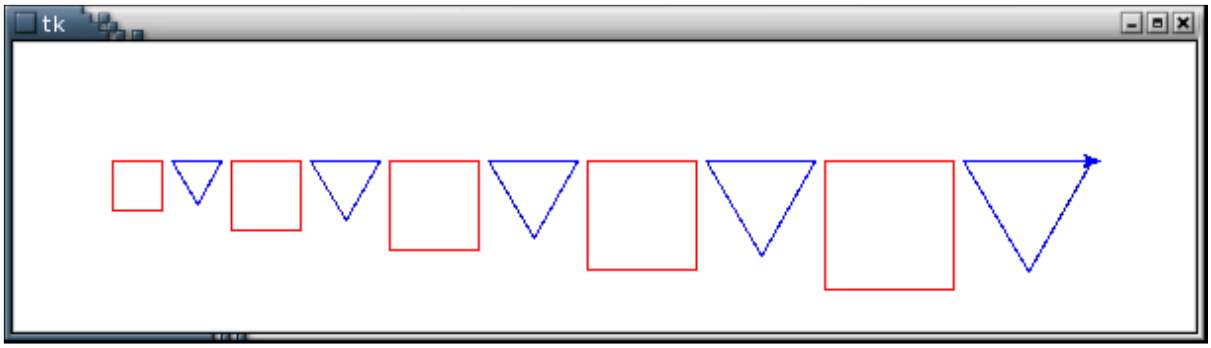
7.2 Définissez une fonction **ligneCar(n, ca)** qui renvoie une chaine de **n** caractères **ca**.

7.3 Définissez une fonction **surfCercle(R)**. Cette fonction doit renvoyer la surface (l'aire) d'un cercle dont on lui a fourni le rayon **R** en argument. Par exemple, l'exécution de l'instruction :
print(surfCercle(2.5)) doit donner le résultat : **19.63495...**

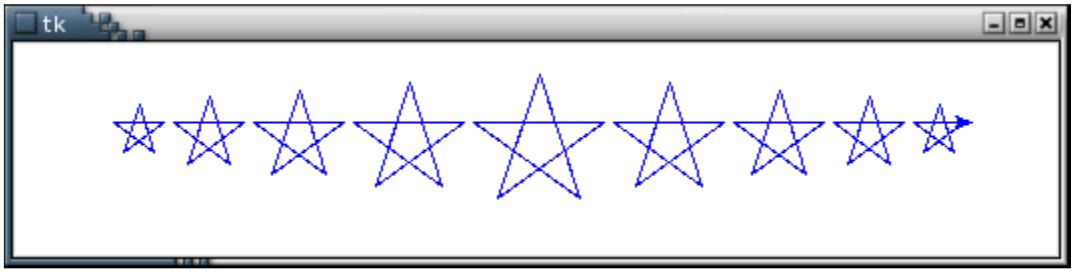
7.4 Définissez une fonction **volBoite(x1,x2,x3)** qui renvoie le volume d'une boite parallélépipédique dont on fournit les trois dimensions **x1, x2, x3** en arguments.
Par exemple, l'exécution de l'instruction :
print(volBoite(5.2, 7.7, 3.3)) doit donner le résultat : **132.132.**

7.5 Définissez une fonction **maximum(n1,n2,n3)** qui renvoie le plus grand de 3 nombres **n1, n2, n3** fournis en arguments. Par exemple, l'exécution de l'instruction :
print(maximum(2,5,4)) doit donner le résultat : **5.**

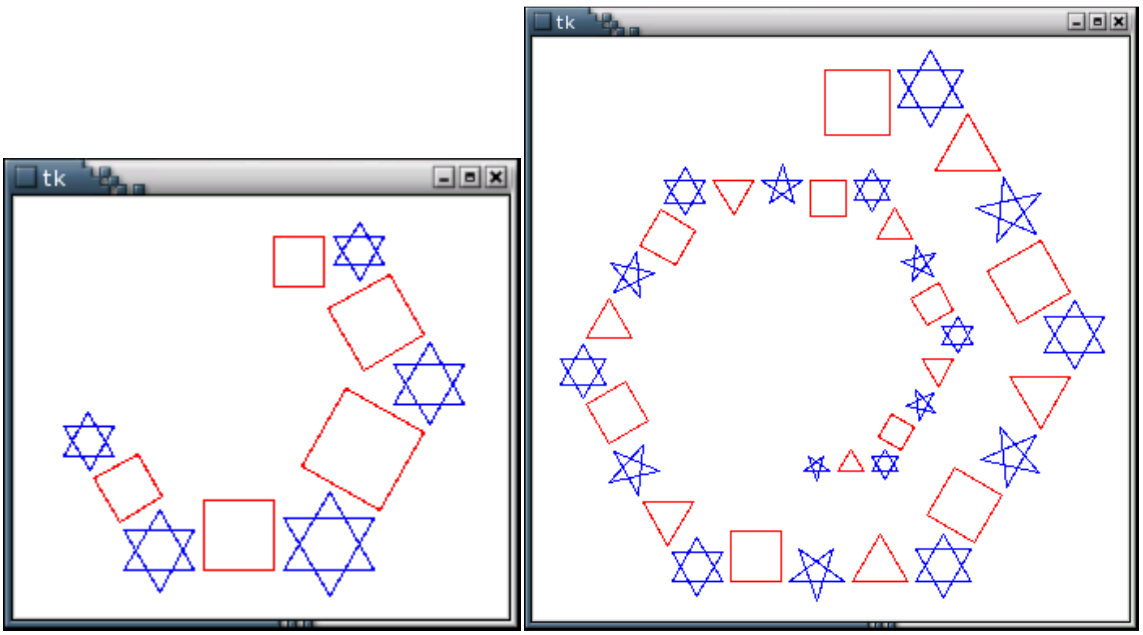
7.6 Complétez le module de fonctions graphiques **dessins_tortue.py** décrit à la page 72. Commencez par ajouter un paramètre **angle** a la fonction **carre()**, de manière a ce que les carres puissent être traces dans différentes orientations. Définissez ensuite une fonction **triangle(taille, couleur, angle)** capable de dessiner un triangle équilatéral d'une taille, d'une couleur et d'une orientation bien déterminées.
Testez votre module à l'aide d'un programme qui fera appel à ces fonctions à plusieurs reprises, avec des arguments varies pour dessiner une série de carres et de triangles :



7.7 Ajoutez au module de l'exercice précèdent une fonction **etoile5()** spécialisée dans le dessin d'étoiles à 5 branches. Dans votre programme principal, insérez une boucle qui dessine une rangée horizontale de de 9 petites étoiles de tailles variées :



7.8 Ajoutez au module de l'exercice précèdent une fonction **etoile6()** capable de dessiner une étoile a 6 branches, elle-même constituée de deux triangles équilatéraux imbriques. Cette nouvelle fonction devra faire appel à la fonction **triangle()** définie précédemment. Votre programme principal dessinera également une série de ces étoiles :



7.9 définissez une fonction **compteCar(ca,ch)** qui renvoie le nombre de fois que l'on rencontre le caractère **ca** dans la chaine de caractères **ch**. Par exemple, l'exécution de l'instruction :
print(compteCar('e', 'Cette phrase est un exemple')) doit donner le résultat : **7**

7.10 Définissez une fonction **indexMax(liste)** qui renvoie l'index de l'élément ayant la valeur la plus élevée dans la liste transmise en argument. Exemple d'utilisation :
serie = [5, 8, 2, 1, 9, 3, 6, 7]
print(indexMax(serie))
4
7.11 Définissez une fonction **nomMois(n)** qui renvoie le nom du n-ieme mois de l'année.
Par exemple, l'exécution de l'instruction :
print(nomMois(4)) doit donner le résultat : **Avril.**
7.12 Définissez une fonction **inverse(ch)** qui permette d'inverser les l'ordre des caracteres d'une chaine quelconque. La chaine inversée sera renvoyée au programme appelant.
7.13 Définissez une fonction **compteMots(ph)** qui renvoie le nombre de mots contenus dans la phrase **ph**. On considère comme mots les ensembles de caractères inclus entre des espaces.
7.14 Modifiez la fonction **volBoite(x1,x2,x3)** que vous avez définie dans un exercice précèdent, de manière a ce qu'elle puisse être appelée avec trois, deux, un seul, ou même aucun argument. Utilisez pour ceux-ci des valeurs par défaut égales a 10.
Par exemple :
print(volBoite()) doit donner le resultat : **1000**
print(volBoite(5.2)) doit donner le resultat : **520.0**
print(volBoite(5.2, 3)) doit donner le resultat : **156.0**
7.15 Modifiez la fonction **volBoite(x1,x2,x3)** ci-dessus de manière a ce qu'elle puisse être appelée avec un, deux, ou trois arguments. Si un seul est utilisé, la boite est considérée comme cubique (l'argument étant l'arête de ce cube). Si deux sont utilisés, la boite est considérée comme un prisme a base carrée (auquel cas le premier argument est le côté du carrée, et le second la hauteur du prisme). Si trois arguments sont utilisés, la boite est considérée comme un parallélépipède. Par exemple :
print(volBoite()) doit donner le resultat : **-1** (indication d'une erreur)
print(volBoite(5.2)) doit donner le resultat : **140.608**
print(volBoite(5.2, 3)) doit donner le resultat : **81.12**
print(volBoite(5.2, 3, 7.4)) doit donner le resultat : **115.44**

7.16 Définissez une fonction **changeCar(ch,ca1,ca2,debut,fin)** qui remplace tous les caractères **ca1** par des caractères **ca2** dans la chaine de caracteres **ch**, à partir de l'indice **debut** et jusqu'à l'indice **fin**, ces deux derniers arguments pouvant être omis (et dans ce cas la chaine est traitée d'une extrémité à l'autre). Exemples de la fonctionnalité attendue :

```
>>> phrase = 'Ceci est une toute petite phrase.'
>>> print(changeCar(phrase, ' ', '*'))
Ceci*est*une*toute*petite*phrase.
>>> print(changeCar(phrase, ' ', '*', 8, 12))
Ceci est*une*toute petite phrase.
>>> print(changeCar(phrase, ' ', '*', 12))
Ceci est une*toute*petite*phrase.
>>> print(changeCar(phrase, ' ', '*', fin = 12))
Ceci*est*une*toute petite phrase.
```

7.17 Définissez une fonction **eleMax(liste,debut,fin)** qui renvoie l'élément ayant la plus grande valeur dans la liste transmise. Les deux arguments **début** et **fin** indiqueront les indices entre lesquels doit s'exercer la recherche, et chacun d'eux pourra être omis (comme dans l'exercice précèdent). Exemples de la fonctionnalité attendue :

```
>>> serie = [9, 3, 6, 1, 7, 5, 4, 8, 2]
>>> print(eleMax(serie))
9
>>> print(eleMax(serie, 2, 5))
7
>>> print(eleMax(serie, 2))
8
>>> print(eleMax(serie, fin =3, debut =1))
6
```

Exercices

8.1 Comment faut-il modifier le programme pour ne plus avoir que des lignes de couleur *cyan, maroon* et *green* ?

8.2 Comment modifier le programme pour que toutes les lignes tracées soient horizontales et parallèles ?

8.3 Agrandissez le canevas de manière a lui donner une largeur de 500 unités et une hauteur de 650. Modifiez également la taille des lignes, afin que leurs extrémités se confondent avec les bords du canevas.

8.4 Ajoutez une fonction **drawline2** qui tracera deux lignes rouges en croix au centre du canevas : l'une horizontale et l'autre verticale. Ajoutez également un bouton

portant l'indication ≪ viseur ≫. Un clic sur ce bouton devra provoquer l'affichage de la croix.

8.5 Reprenez le programme initial. Remplacez la méthode **create_line** par **create_rectangle**.

Que se passe-t-il ?

De la même façon, essayez aussi **create_arc**, **create_oval**, et **create_polygon**.

Pour chacune de ces méthodes, notez ce qu'indiquent les coordonnées fournies en paramètres.

(Remarque : pour le polygone, il est nécessaire de modifier légèrement le programme !)

8.6 - Supprimez la ligne **global x1, y1, x2, y2** dans la fonction **drawline** du programme original. Que se passe-t-il ? Pourquoi ?

- Si vous placez plutôt ≪ x1, y1, x2, y2 ≫ entre les parenthèses, dans la ligne de définition de la fonction **drawline**, de manière à transmettre ces variables à la fonction en tant que paramètres, le programme fonctionne-t-il encore ? N'oubliez pas de modifier aussi la ligne du programme qui fait appel à cette fonction !

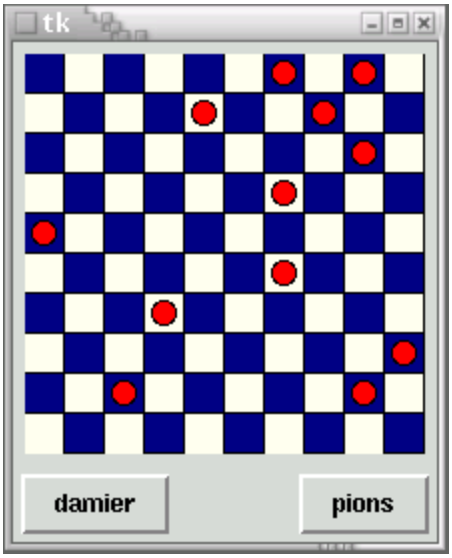
- Si vous définissez **x1, y1, x2, y2 = 10, 390, 390, 10** a la place de **global x1, y1, ...**, que se passe-t-il ? Pourquoi ? Quelle conclusion pouvez-vous tirer de tout cela ?

8.7 a) Créez un court programme qui dessinera les 5 anneaux olympiques dans un rectangle de fond blanc (white). Un bouton ≪ Quitter ≫ doit permettre de fermer la fenêtre.

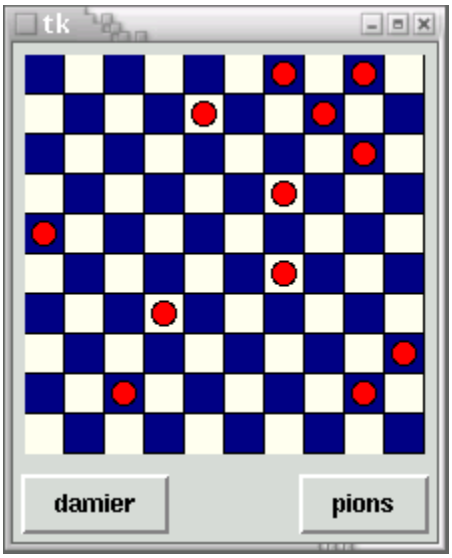
b) Modifiez le programme ci-dessus en y ajoutant 5 boutons. Chacun de ces boutons provoquera le trace de chacun des 5 anneaux

8.8 Dans votre cahier de notes, établissez un tableau a deux colonnes. Vous y noterez à gauche les définitions des classes d'objets déjà rencontrées (avec leur liste de paramètres), et à droite les méthodes associées à ces classes (également avec leurs paramètres). Laissez de la place pour compléter ultérieurement.

8.9 Inspirez-vous du script précèdent pour écrire une petite application qui fait apparaitre un damier (dessin de cases noires sur fond blanc) lorsque l'on clique sur un bouton :



8.10 A l'application de l'exercice précédent, ajoutez un bouton qui fera apparaitre des pions au hasard sur le damier (chaque pression sur le bouton fera apparaitre un nouveau pion).



8.11 Modifiez le script ci-dessus de manière à faire apparaitre un petit cercle rouge à l'endroit où l'utilisateur a effectué son clic (vous devrez d'abord remplacer le widget **Frame** par un widget Canvas).

8.12 Ecrivez un programme qui fait apparaitre une fenêtre avec un canevas. Dans ce canevas on verra deux cercles (de tailles et de couleurs différentes), qui sont censés représenter deux astres. Des boutons doivent permettre de les déplacer à volonté tous les deux dans toutes les directions. Sous le canevas, le programme doit afficher en permanence : a) la distance séparant les deux astres ; b) la force gravitationnelle qu'ils exercent l'un sur l'autre (penser à afficher en haut de fenêtre les masses choisies pour chacun d'eux, ainsi que l'échelle des distances). Dans cet exercice, vous utiliserez

évidemment la loi de la gravitation universelle de Newton (cf. exercice 6.16, page 59, et un manuel de Physique générale).

8.13 En vous inspirant du programme qui détecte les clics de souris dans un canevas, modifiez le programme ci-dessus pour y réduire le nombre de boutons : pour déplacer un astre, il suffira de le choisir avec un bouton, et ensuite de cliquer sur le canevas pour que cet astre se positionne à l'endroit où l'on a clique.

8.14 Extension du programme ci-dessus. Faire apparaitre un troisième astre, et afficher en permanence la force résultante agissant sur chacun des trois (en effet : chacun subit en permanence l'attraction gravitationnelle exercée par les deux autres !).

8.15 Même exercice avec des charges électriques (loi de Coulomb). Donner cette fois une possibilité de choisir le signe des charges.

8.16 Ecrivez un petit programme qui fait apparaitre une fenêtre avec deux champs : l'un indique une température en degrés *Celsius*, et l'autre la même température exprimée en degrés *Fahrenheit*. Chaque fois que l'on change une quelconque des deux températures, l'autre est corrigée en conséquence. Pour convertir les degrés *Fahrenheit* en *Celsius* et vice-versa, on utilise la formule $T\,F=TC\times 1{,}80+32$. Revoyez aussi le petit programme concernant la calculatrice simplifiée (page 94).

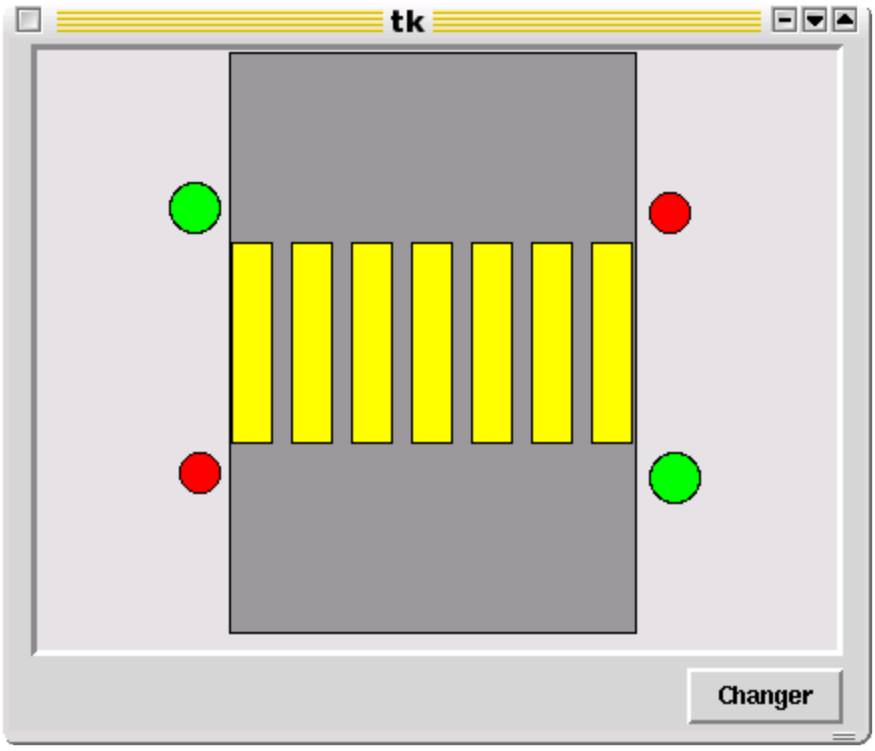
8.17 Ecrivez un programme qui fait apparaitre une fenêtre avec un canevas. Dans ce canevas, placez un petit cercle censé représenter une balle. Sous le canevas, placez un bouton. Chaque fois que l'on clique sur le bouton, la balle doit avancer d'une petite distance vers la droite, jusqu'à ce qu'elle atteigne l'extrémité du canevas. Si l'on continue à cliquer, la balle doit alors revenir en arrière jusqu'à l'autre extrémité, et ainsi de suite.

8.18 Améliorez le programme ci-dessus pour que la balle décrive cette fois une trajectoire circulaire ou elliptique dans le canevas (lorsque l'on clique continuellement). Note : pour arriver au résultat escompte, vous devrez nécessairement définir une variable qui représentera l'angle décrit, et utiliser les fonctions *sinus* et *cosinus* pour positionner la balle en fonction de cet angle.

8.19 Modifiez le programme ci-dessus de telle manière que la balle, en se déplaçant, laisse derrière elle une trace de la trajectoire décrite.

8.20 Modifiez le programme ci-dessus de manière à tracer d'autres figures. Consultez votre professeur pour des suggestions (courbes de *Lissajous*).

8.21 Ecrivez un programme qui fait apparaitre une fenêtre avec un canevas
et un bouton. Dans le canevas, tracez un rectangle gris fonce, lequel représentera une route, et par-dessus, une série de rectangles jaunes censés représenter un passage pour piétons. Ajoutez quatre cercles colores pour figurer les feux
de circulation concernant les piétons et les véhicules. Chaque utilisation du bouton devra provoquer le changement de couleur des feux :



8.22 Ecrivez un programme qui montre un canevas dans lequel est dessine un circuit électrique simple (générateur + interrupteur + résistance). La fenêtre doit être pourvue de champs d'entrée qui permettront de paramétrer chaque élément (c'est-à-dire choisir les valeurs des résistances et tensions). L'interrupteur doit être fonctionnel (prévoyez un bouton ≪ Marche/arrêt ≫ pour cela). Des ≪ étiquettes ≫ doivent afficher en permanence les tensions et intensités résultant des choix effectues par l'utilisateur.

8.23 Dans la fonction **start_it()**, supprimez l'instruction **if flag == 0:** (et l'indentation des deux lignes suivantes). Que se passe-t-il ? (Cliquez plusieurs fois sur le bouton ≪ Demarrer ≫.)

Tachez d'exprimer le plus clairement possible votre explication des faits observes.

8.24 Modifiez le programme de telle façon que la balle change de couleur a chaque ≪ virage ≫.

8.25 Modifiez le programme de telle façon que la balle effectue des mouvements obliques comme une bille de billard qui rebondit sur les bandes (≪ en zig-zag ≫).

8.26 Modifiez le programme de manière à obtenir d'autres mouvements. Tachez par exemple d'obtenir un mouvement circulaire (comme dans les exercices de la page 107).

8.27 Modifiez ce programme, ou bien écrivez-en un autre similaire, de manière à simuler le mouvement d'une balle qui tombe (sous l'effet de la pesanteur), et rebondit sur le sol. Attention : il s'agit cette fois de mouvements accélérés !

8.28 A partir des scripts précédents, vous pouvez à présent écrire un programme de jeu fonctionnant de la manière suivante : une balle se déplace au hasard sur un canevas, a vitesse faible. Le joueur doit essayer de cliquer sur cette balle à l'aide de la souris. S'il y arrive, il gagne un point, mais la balle se déplace désormais un peu plus vite, et ainsi de suite. Arrêter le jeu après un certain nombre de clics et afficher le score atteint.
8.29 Variante du jeu précèdent : chaque fois que le joueur parvient à ≪ l'attraper ≫, la balle devient plus petite (elle peut également changer de couleur).
8.30 Ecrivez un programme dans lequel évoluent plusieurs balles de couleurs différentes, qui rebondissent les unes sur les autres ainsi que sur les parois.
8.31 Perfectionnez le jeu des précédents exercices en y intégrant l'algorithme ci-dessus. Il s'agit à présent pour le joueur de cliquer seulement sur la balle rouge. Un clic erroné (sur une balle d'une autre couleur) lui fait perdre des points.
8.32 Ecrivez un programme qui simule le mouvement de deux planètes tournant autour du soleil sur des orbites circulaires différentes (ou deux électrons tournant autour d'un noyau d'atome...).
8.33 Ecrivez un programme pour le jeu du serpent : un ≪ serpent ≫ (constitue en fait d'une courte ligne de carres) se déplace sur le canevas dans l'une des 4 directions : droite, gauche, haut, bas. Le joueur peut à tout moment changer la direction suivie par le serpent à l'aide des touches fléchées du clavier. Sur le canevas se trouvent également des
≪ proies ≫ (des petits cercles fixes disposes au hasard). Il faut diriger le serpent de manière à ce qu'il ≪ mange ≫ les proies sans arriver en contact avec les bords du canevas.
A chaque fois qu'une proie est mangée, le serpent s'allonge d'un carrée, le joueur gagne un point, et une nouvelle proie apparait ailleurs. La partie s'arrête lorsque le serpent touche l'une des parois, ou lorsqu'il a atteint une certaine taille.
8.34 Perfectionnement du jeu précédent : la partie s'arrête également si le serpent ≪ se recoupe ≫.

Exercices

9.1 Ecrivez un script qui permette de créer et de relire aisément un fichier texte. Votre programme demandera d'abord à l'utilisateur d'entrer le nom du fichier. Ensuite il lui proposera le choix, soit d'enregistrer de nouvelles lignes de texte, soit d'afficher le contenu du fichier. L'utilisateur devra pouvoir entrer ses lignes de texte successives en utilisant simplement la touche <Enter> pour les séparer les unes des autres. Pour terminer les entrées, il lui suffira d'entrer une ligne vide (c'est-a-dire utiliser la touche <Enter> seule).

L'affichage du contenu devra montrer les lignes du fichier séparées les unes des autres de la manière la plus naturelle (les codes de fin de ligne ne doivent pas apparaitre).

9.2 Considérons que vous avez à votre disposition un fichier texte contenant des phrases de différentes longueurs. Ecrivez un script qui recherche et affiche la phrase la plus longue.

9.3 Ecrivez un script qui génère automatiquement un fichier texte contenant les tables de multiplication de 2 à 30 (chacune d'entre elles incluant 20 termes seulement).

9.4 Ecrivez un script qui recopie un fichier texte en triplant tous les espaces entre les mots.

9.5 Vous avez à votre disposition un fichier texte dont chaque ligne est la représentation d'une valeur numérique de type réel (mais sans exposants). Par exemple :

14.896

7894.6

123.278

etc.

Ecrivez un script qui recopie ces valeurs dans un autre fichier en les arrondissant en nombres entiers (l'arrondi doit être correct).

9.6 Ecrivez un script qui compare les contenus de deux fichiers et signale la première différence rencontrée.

9.7 A partir de deux fichiers préexistants A et B, construisez un fichier C qui contienne alternativement

un élément de A, un élément de B, un élément de A... et ainsi de suite

Jusqu'à atteindre la fin de l'un des deux fichiers originaux. Complétez ensuite C avec les éléments restant sur l'autre.

9.8 Ecrivez un script qui permette d'encoder un fichier texte dont les lignes contiendront chacune les noms, prénom, adresse, code postal et no de téléphone de différentes personnes (considérez par exemple qu'il s'agit des membres d'un club).

9.9 Ecrivez un script qui recopie le fichier utilise dans l'exercice précédent, en y ajoutant la date de naissance et le sexe des personnes (l'ordinateur devra afficher les lignes une par une et demander à l'utilisateur d'entrer pour chacune les données complémentaires).

9.10 Considérons que vous avez fait les exercices précédents et que vous disposez à présent d'un fichier contenant les coordonnées d'un certain nombre de personnes. Ecrivez un script qui permette d'extraire de ce fichier les lignes qui correspondent à un code postal bien détermine.

9.11 Modifiez le script de l'exercice précédent, de manière à retrouver les lignes correspondant

a des prénoms dont la première lettre est située entre F et M (inclus) dans l'alphabet.
9.12 Ecrivez des fonctions qui effectuent le même travail que celles du module **pickle** (voir page 121). Ces fonctions doivent permettre l'enregistrement de variables diverses dans un fichier texte, en les accompagnant systématiquement d'informations concernant leur format exact.

Exercices

10.1 Déterminez vous-même ce qui se passe, dans la technique de *slicing*, lorsque l'un ou l'autre des indices de découpage est erroné, et décrivez cela le mieux possible. (Si le second indice est plus petit que le premier, par exemple, ou bien si le second indice est plus grand que la taille de la chaine).

10.2 Découpez une grande chaine en fragments de 5 caractères chacun. Rassemblez ces morceaux dans l'ordre inverse. La chaine doit pouvoir contenir des caractères accentues.

10.3 Tachez d'écrire une petite fonction **trouve()** qui fera exactement le contraire de ce que fait l'opérateur d'indexage (c'est-a-dire les crochets **[ ]**). Au lieu de partir d'un index donne pour retrouver le caractère correspondant, cette fonction devra retrouver l'index correspondant à un caractère donne.

En d'autres termes, il s'agit d'écrire une fonction qui attend deux arguments : le nom de la chaine a traiter et le caractère à trouver. La fonction doit fournir en retour l'index du premier caractère de ce type dans la chaine. Ainsi par exemple, l'instruction :

print(trouve("Juliette & Romeo", "&")) devra afficher : **9**

Attention : il faut penser à tous les cas possibles. Il faut notamment veiller à ce que la fonction renvoie une valeur particulière (par exemple la valeur -1) si le caractère recherche n'existe pas dans la chaine traitée. Les caractères accentues doivent être acceptes.

10.4 Améliorez la fonction de l'exercice précédent en lui ajoutant un troisième paramètre : l'index a partir duquel la recherche doit s'effectuer dans la chaine.

Ainsi par exemple, l'instruction :

print(trouve ("Cesar & Cleopatre", "r", 5)) devra afficher : **15** (et non 4 !).

10.5 Ecrivez une fonction **compteCar()** qui compte le nombre d'occurrences d'un caractère donne dans une chaine. Ainsi :

10.6 Dans un conte américain, huit petits canetons s'appellent respectivement : *Jack, Kack, Lack, Mack, Nack, Oack, Pack et Qack*. Ecrivez un petit script qui génère tous ces noms à partir des deux chaines suivantes :

prefixes = 'JKLMNOP' et **suffixe = 'ack'**

Si vous utilisez une instruction **for ... in ...**, votre script ne devrait comporter que deux lignes.

10.7 Dans un script, écrivez une fonction qui recherche le nombre de mots contenus dans une phrase donnée.
10.8 Ecrivez un script qui recherche le nombre de caractères "e", "e", "e", "e", "e" contenus dans une phrase donnée.
10.9 Ecrivez une fonction **estUnChiffre()** qui renvoie ≪ vrai ≫, si l'argument transmis est un chiffre, et ≪ faux ≫ sinon. Tester ainsi tous les caractères d'une chaine en parcourant celle-ci a l'aide d'une boucle **for**.
10.10 Ecrivez une fonction **estUneMaj()** qui renvoie ≪ vrai ≫ si l'argument transmis est une majuscule. Tachez de tenir compte des majuscules accentuées !
10.11 Ecrivez une fonction **chaineListe()** qui convertisse une phrase en une liste de mots.
10.12Utilisez les fonctions définies dans les exercices précédents pour écrire un script qui puisse extraire d'un texte tous les mots qui commencent par une majuscule.
10.13Utilisez les fonctions définies dans les exercices précédents pour écrire une fonction
qui renvoie le nombre de caractères majuscules contenus dans une phrase donnée en argument.
10.14 Ecrivez un petit script qui affiche une table des codes *ASCII*. Le programme doit afficher tous les caractères en regard des codes correspondants. A partir de cette table, établissez la relation numérique simple reliant chaque caractère majuscule au caractère minuscule correspondant.
10.15Modifiez le script précédent pour explorer les codes situes entre 128 et 256, où vous retrouverez nos caractères accentues (parmi de nombreux autres). La relation numérique trouvée dans l'exercice précédent reste-t-elle valable aussi pour les caractères accentues du Français ?
10.16 A partir de cette relation, écrivez une fonction qui convertit tous les caractères minuscules en majuscules, et vice-versa (dans une phrase fournie en argument).
10.17 Ecrivez un script qui recopie un fichier texte en remplaçant tous ses espaces par le groupe de trois caractères **-*-.** Le fichier à copier sera fourni encode à la norme Latin-1, et le fichier destinataire devra être encode en Utf-8. Les noms des 2 fichiers devront être demandes en début de script.
10.18 Ecrivez une fonction **voyelle(car)**, qui renvoie ≪ vrai ≫ si le caractère fourni en argument est une voyelle.
10.19 Ecrivez une fonction **compteVoyelles(phrase)**, qui renvoie le nombre de voyelles contenues dans une phrase donnée.
10.20 Explorez la gamme des caractères *Unicode* disponibles sur votre ordinateur, à l'aide de boucles de programmes similaires a celle que nous avons nous-même utilisée

pour afficher l'alphabet grec. Trouvez ainsi les codes correspondant à l'alphabet cyrillique, et écrivez un script qui affiche celui-ci en majuscules et en minuscules.

10.21 Ecrivez un script qui recopie en *Utf-8* un fichier texte encode a l'origine en *Latin-1*, en veillant en outre à ce que chaque mot commence par une majuscule.
Le programme demandera les noms des fichiers a l'utilisateur. Les opérations de lecture et d'écriture des fichiers auront lieu en mode texte ordinaire.
10.22 Variante de l'exercice précédent : effectuez les opérations de lecture et d'écriture des fichiers en mode binaire, et les opérations de décodage/encodage sur les séquences d'octets. Au passage, vous traiterez les lignes de manière à remplacer tous les espaces par le groupe de 3 caractères ≪ -*- ≫.
10.23 Ecrivez un script qui compte le nombre de mots contenus dans un fichier texte.
10.24 Ecrivez un script qui recopie un fichier texte en fusionnant (avec la précédente) les lignes qui ne commencent pas par une majuscule.
10.25 Vous disposez d'un fichier contenant des valeurs numériques. Considérez que ces valeurs sont les diamètres d'une série de sphères. Ecrivez un script qui utilise les données de ce fichier pour en créer un autre, organise en lignes de texte qui exprimeront ≪ en clair ≫ les autres caractéristiques de ces sphères (surface de section, surface extérieure et volume), dans des phrases telles que :
Diam. 46.20 cm Section 1676.39 cm2 Surf. 6705.54 cm2 Vol. 51632.67 cm3
Diam. 120.00 cm Section 11309.73 cm2 Surf. 45238.93 cm2 Vol. 904778.68 cm3
Diam. 0.03 cm Section 0.00 cm2 Surf. 0.00 cm2 Vol. 0.00 cm3
Diam. 13.90 cm Section 151.75 cm2 Surf. 606.99 cm2 Vol. 1406.19 cm3
Diam. 88.80 cm Section 6193.21 cm2 Surf. 24772.84 cm2 Vol. 366638.04 cm3
etc.
10.26 Vous avez à votre disposition un fichier texte dont les lignes représentent des valeurs numériques de type réel, sans exposant (et encodées sous forme de chaines de caractères).
Ecrivez un script qui recopie ces valeurs dans un autre fichier, en les arrondissant de telle sorte que leur partie décimale ne comporte plus qu'un seul chiffre après la virgule, ce chiffre ne pouvant être que 0 ou 5 (l'arrondi doit être correct).
10.27 Ecrivez un script qui génère la liste des carrées et des cubes des nombres de 20 a 40.
10.28 Ecrivez un script qui crée automatiquement la liste des *sinus* des angles de 0° à 90°, par pas de 5°. Attention : la fonction **sin()** du module **math** considère que les angles sont fournis en *radians* (360° = 2 π radians).

10.29 Ecrivez un script qui permette d'obtenir a l'écran les 15 premiers termes des tables de multiplication par 2, 3, 5, 7, 11, 13, 17, 19 (ces nombres seront places au départ dans une liste) sous la forme d'une table similaire a la suivante :

2 4 6 8 10 12 14 16 18 20 22 24 26 28 30

3 6 9 12 15 18 21 24 27 30 33 36 39 42 45

5 10 15 20 25 30 35 40 45 50 55 60 65 70 75

etc.

10.30 Soit la liste suivante : **['Jean-Michel', 'Marc', 'Vanessa', 'Anne', 'Maximilien', 'Alexandre-Benoit', 'Louise']**

Ecrivez un script qui affiche chacun de ces noms avec le nombre de caractères correspondant.

10.31 Vous disposez d'une liste de nombres entiers quelconques, certains d'entre eux étant présents en plusieurs exemplaires. Ecrivez un script qui recopie cette liste dans une autre, *en omettant les doublons*. La liste finale devra être *triée*.

10.32 Ecrivez un script qui recherche le mot le plus long dans une phrase donnée (l'utilisateur du programme doit pouvoir entrer une phrase de son choix).

10.33 Ecrivez un script capable d'afficher la liste de tous les jours d'une année imaginaire, laquelle commencerait un jeudi. Votre script utilisera lui-même trois listes : une liste des noms de jours de la semaine, une liste des noms des mois, et une liste des nombres de jours que comportent chacun des mois (ne pas tenir compte des années bissextiles).

Exemple de sortie :

jeudi 1 janvier vendredi 2 janvier samedi 3 janvier dimanche 4 janvier

... et ainsi de suite, jusqu'au jeudi 31 décembre.

10.34 Vous avez a votre disposition un fichier texte qui contient un certain nombre de noms d'élèves. Ecrivez un script qui effectue une copie *triée* de ce fichier.

10.35 Ecrivez une fonction permettant de trier une liste. Cette fonction ne pourra pas utiliser la méthode intégrée **sort()** de Python : vous devez donc *définir vous-même l'algorithme de tri*.

10.36 Soient les listes suivantes :

t1 = [31,28,31,30,31,30,31,31,30,31,30,31]

t2 = ['Janvier','Février','Mars','Avril','Mai','Juin', 'Juillet','Aout','Septembre','Octobre','Novembre','Décembre']

Ecrivez un petit programme qui insère dans la seconde liste tous les éléments de la première, de telle sorte que chaque nom de mois soit suivi du nombre de jours correspondant

: **['Janvier',31,'Fevrier',28,'Mars',31, etc.]**.

10.37 Créez une liste **A** contenant quelques éléments. Effectuez une *vraie copie* de cette liste dans une nouvelle variable **B**. Suggestion : créez d'abord une liste **B** de même taille que **A** mais ne contenant que des zéros. Remplacez ensuite tous ces zéros par les éléments tires de **A**.

10.38Meme question, mais autre suggestion : créez d'abord une liste **B** vide. Remplissez-la ensuite à l'aide des éléments de **A** ajoutes l'un après l'autre.

10.39Meme question, autre suggestion encore : pour créer la liste **B**, découpez dans la liste **A** une tranche incluant tous les éléments (a l'aide de l'operateur **[:]**).

10.40 *Un nombre premier est un nombre qui n'est divisible que par un et par lui-même*. Ecrivez un programme qui établit la liste de tous les nombres premiers compris entre 1 et 1000, en utilisant la méthode du *crible d'Eratosthène* :

• Créez une liste de 1000 éléments, chacun initialise a la valeur 1.

• Parcourez cette liste a partir de l'élément d'indice 2 : si l'élément analyse possède la valeur 1, mettez a zéro tous les autres éléments de la liste, dont les indices sont des multiples entiers de l'indice auquel vous êtes arrive.

Lorsque vous aurez parcouru ainsi toute la liste, les indices des éléments qui seront restes a 1 seront les nombres premiers recherches. En effet : A partir de l'indice 2, vous annulez tous les éléments d'indices pairs : 4, 6, 8, 10, etc. Avec l'indice 3, vous annulez les éléments d'indices 6, 9, 12, 15, etc., et ainsi de suite. Seuls resteront a 1 les éléments dont les indices sont effectivement des nombres premiers.

10.41 Réécrivez la fonction **list_aleat()** ci-dessus, en utilisant la méthode **append()** pour construire la liste petit à petit à partir d'une liste vide (au lieu de remplacer les zéros d'une liste préexistante comme nous l'avons fait).

10.42 Ecrivez une fonction **imprime_liste()** qui permette d'afficher ligne par ligne tous les éléments contenus dans une liste de taille quelconque. Le nom de la liste sera fourni en argument. Utilisez cette fonction pour imprimer la liste de nombres aléatoires génères par la fonction **list_aleat()**. Ainsi par exemple, l'instruction **imprime_liste(liste_aleat(8))** devra afficher une colonne de 8 nombres réels aléatoires.

10.43 Vous allez écrire un programme destine à vérifier le fonctionnement du générateur de nombres aléatoires de Python en appliquant la théorie exposée ci-dessus. Votre programme devra donc :

• Demander à l'utilisateur le nombre de valeurs a tirer au hasard a l'aide de la fonction **random()**. Il serait intéressant que le programme propose un nombre par défaut (1000 par exemple).

• Demander à l'utilisateur en combien de fractions il souhaite partager l'intervalle des Valeurs possibles (c'est-a-dire l'intervalle de 0 à 1). Ici aussi, il faudrait proposer un nombre de fractions par défaut (5 par exemple). Vous pouvez également limiter le

choix de l'utilisateur a un nombre compris entre 2 et le 1/10e du nombre de valeurs tirées au hasard.

• Construire une liste de N compteurs (N étant le nombre de fractions souhaitées). Chacun d'eux sera évidemment initialise à zéro.

• Tirer au hasard toutes les valeurs demandées, à l'aide de la fonction **random()** , et mémoriser ces valeurs dans une liste.

• Mettre en œuvre un parcours de la liste des valeurs tirées au hasard (boucle), et effectuer un test sur chacune d'elles pour déterminer dans quelle fraction de l'intervalle 0-1 elle se situe. Incrémenter de une unité le compteur correspondant.

• Lorsque c'est terminé, afficher l'état de chacun des compteurs.

10.44 Ecrivez un script qui tire au hasard des cartes à jouer. Le nom de la carte tirée doit être correctement présente, ≪ en clair ≫. Le programme affichera par exemple :

Frappez <Enter> pour tirer une carte :
Dix de Trefle
Frappez <Enter> pour tirer une carte :
As de Carreau
Frappez <Enter> pour tirer une carte :
Huit de Pique
Frappez <Enter> pour tirer une carte :
etc.

10.45 Ecrivez un script qui crée un mini-système de base de données fonctionnant a l'aide
d'un dictionnaire, dans lequel vous mémoriserez les noms d'une série de copains, leur âge et leur taille. Votre script devra comporter deux fonctions : la première pour le remplissage du dictionnaire, et la seconde pour sa consultation. Dans la fonction de remplissage, utilisez une boucle pour accepter les données entrées par l'utilisateur. Dans le dictionnaire, le nom de l'élève servira de clé d'accès, et les valeurs seront constituées de tuples (âge, taille), dans lesquels l'âge sera exprimé en années (donnée de type entier), et la taille en mètres (donnée de type réel). La fonction de consultation comportera elle aussi une boucle, dans laquelle l'utilisateur pourra fournir un nom quelconque pour obtenir en retour le couple ≪ âge, taille ≫
correspondant. Le résultat de la requête devra être une ligne de texte bien formatée, telle par exemple : ≪ Nom : Jean Dhoute - âge : 15 ans - taille : 1.74 m ≫. Pour obtenir ce résultat, servez-vous du formatage des chaines de caractères décrit a la page 144.

10.46 Ecrivez une fonction qui échange les clés et les valeurs d'un dictionnaire (ce qui permettra par exemple de transformer un dictionnaire anglais/français en un dictionnaire français/anglais). On suppose que le dictionnaire ne contient pas plusieurs valeurs identiques.

10.47 Vous avez à votre disposition un fichier texte quelconque (pas trop gros). Ecrivez un script qui compte les occurrences de chacune des lettres de l'alphabet dans ce texte (on simplifiera le problème en ne tenant pas compte des lettres accentuées).

10.48Modifiez le script ci-dessus afin qu'il établisse une table des occurrences de chaque *mot* dans le texte. Conseil : dans un texte quelconque, les mots ne sont pas seulement sépares par des espaces, mais également par divers signes de ponctuation. Pour simplifier le problème, vous pouvez commencer par remplacer tous les caractères non-alphabétiques par des espaces, et convertir la chaine résultante en une liste de mots a l'aide de
la méthode **split()**.

10.49 Vous avez à votre disposition un fichier texte quelconque (pas trop gros). Ecrivez un script qui analyse ce texte, et mémorise dans un dictionnaire l'emplacement exact de chacun des mots (compte en nombre de caractères a partir du début). Lorsqu'un même mot apparait plusieurs fois, tous ses emplacements doivent être mémorises : chaque valeur de votre dictionnaire doit donc être une liste d'emplacements.

10.50 Complétez l'exercice 10.46 (mini-système de base de données) en lui ajoutant deux fonctions : l'une pour enregistrer le dictionnaire résultant dans un fichier texte, et l'autre pour reconstituer ce dictionnaire à partir du fichier correspondant.
Chaque ligne de votre fichier texte correspondra à un élément du dictionnaire. Elle sera formatée de manière a bien séparer :
- la clé et la valeur (c'est-à-dire le nom de la personne, d'une part, et l'ensemble : ≪ âge + taille ≫, d'autre part ;
- dans l'ensemble ≪ âge + taille ≫, ces deux données numériques.

Vous utiliserez donc deux caractères séparateurs différents, par exemple ≪ @ ≫ pour séparer la clé et la valeur, et ≪ # ≫ pour séparer les données constituant cette valeur :

Juliette@18#1.67
Jean-Pierre@17#1.78
Delphine@19#1.71
Anne-Marie@17#1.63 etc.

10.51 Améliorez encore le script de l'exercice précédent, en utilisant un dictionnaire pour diriger le flux d'exécutions du programme au niveau du menu principal.
Votre programme affichera par exemple :

Choisissez :
(R)ecuperer un dictionnaire préexistant sauvegarde dans un fichier
(A)jouter des données au dictionnaire courant
(C)onsulter le dictionnaire courant
(S)auvegarder le dictionnaire courant dans un fichier

(T)erminer :
Suivant le choix opère par l'utilisateur, vous effectuerez alors l'appel de la fonction correspondante en la sélectionnant dans un dictionnaire de fonctions.

Exercices
11.1 Ecrivez une fonction **distance()** qui permette de calculer la distance entre deux points. (Il faudra vous rappeler le théorème de Pythagore !) Cette fonction attendra évidemment deux objets **Point()** comme arguments.

Exercices
12.1 Définissez une classe **Domino()** qui permette d'instancier des objets simulant les pièces d'un jeu de dominos. Le constructeur de cette classe initialisera les valeurs des points présents sur les deux faces A et B du domino (valeurs par défaut = 0).
Deux autres méthodes seront définies :
• une méthode **affiche_points()** qui affiche les points présents sur les deux faces ;
• une méthode **valeur()** qui renvoie la somme des points présents sur les 2 faces.
Exemples d'utilisation de cette classe :

```
>>> d1 = Domino(2,6)
>>> d2 = Domino(4,3)
>>> d1.affiche_points()
face A : 2 face B : 6
>>> d2.affiche_points()
face A : 4 face B : 3
>>> print("total des points :", d1.valeur() + d2.valeur())
15
>>> liste_dominos = []
>>> for i in range(7):
... liste_dominos.append(Domino(6, i))
>>> print(liste_dominos[3])
<__main__.Domino object at 0xb758b92c>
```

etc.
12.2 Definissez une classe **CompteBancaire()**, qui permette d'instancier des objets tels que **compte1**, **compte2**, etc. Le constructeur de cette classe initialisera deux attributs d'instance **nom** et **solde**, avec les valeurs par défaut 'Dupont' et 1000.
Trois autres méthodes seront définies :
• **dépôt(somme)** permettra d'ajouter une certaine somme au solde ;
• **retrait(somme)** permettra de retirer une certaine somme du solde ;

• **affiche()** permettra d'afficher le nom du titulaire et le solde de son compte.
Exemples d'utilisation de cette classe :

```
>>> compte1 = CompteBancaire('Duchmol', 800)
>>> compte1.depot(350)
>>> compte1.retrait(200)
>>> compte1.affiche()
Le solde du compte bancaire de Duchmol est de 950 euros.
>>> compte2 = CompteBancaire()
>>> compte2.depot(25)
>>> compte2.affiche()
Le solde du compte bancaire de Dupont est de 1025 euros.
```

12.3 Definissez une classe **Voiture()** qui permette d'instancier des objets reproduisant le comportement de voitures automobiles. Le constructeur de cette classe initialisera les attributs d'instance suivants, avec les valeurs par défaut indiquées :
marque = 'Ford', couleur = 'rouge', pilote = 'personne', vitesse = 0.
Lorsque l'on instanciera un nouvel objet **Voiture()**, on pourra choisir sa marque et sa couleur, mais pas sa vitesse, ni le nom de son conducteur. Les méthodes suivantes seront définies :
• **choix_conducteur(nom)** permettra de designer (ou changer) le nom du conducteur.
• **accélérer(taux, durée)** permettra de faire varier la vitesse de la voiture. La variation de vitesse obtenue sera égale au produit : **taux** × **durée**. Par exemple, si la voiture accélère au taux de 1,3 m/s pendant 20 secondes, son gain de vitesse doit être égal a 26 m/s. Des taux négatifs seront acceptés (ce qui permettra de décélérer). La variation de vitesse ne sera pas autorisée si le conducteur est 'personne'.
• **affiche_tout()** permettra de faire apparaitre les propriétés présentes de la voiture, c'est-à-dire sa marque, sa couleur, le nom de son conducteur, sa vitesse.
Exemples d'utilisation de cette classe :

```
>>> a1 = Voiture('Peugeot', 'bleue')
>>> a2 = Voiture(couleur = 'verte')
>>> a3 = Voiture('Mercedes')
>>> a1.choix_conducteur('Romeo')
>>> a2.choix_conducteur('Juliette')
>>> a2.accelerer(1.8, 12)
>>> a3.accelerer(1.9, 11)
Cette voiture n'a pas de conducteur !
>>> a2.affiche_tout()
Ford verte pilotee par Juliette, vitesse = 21.6 m/s.
>>> a3.affiche_tout()
```

Mercedes rouge pilotee par personne, vitesse = 0 m/s.

12.4 Definissez une classe **Satellite()** qui permette d'instancier des objets simulant des satellites artificiels lances dans l'espace, autour de la terre. Le constructeur de cette classe initialisera les attributs d'instance suivants, avec les valeurs par défaut indiquées:

masse = 100, vitesse = 0.

Lorsque l'on instanciera un nouvel objet **Satellite()**, on pourra choisir son nom, sa masse et sa vitesse. Les méthodes suivantes seront définies :

• **impulsion(force, durée)** permettra de faire varier la vitesse du satellite. Pour savoir comment, rappelez-vous votre cours de physique : la variation de vitesse **Δv** subie par un objet de masse **m** soumis a l'action d'une force **F** pendant un temps **t** vaut **Δv** $=F\times t/m$. Par exemple : un satellite de 300 kg qui subit une force de 600 Newtons pendant 10 secondes voit sa vitesse augmenter (ou diminuer) de 20 m/s.

• **affiche_vitesse()** affichera le nom du satellite et sa vitesse courante.

• **énergie()** renverra au programme appelant la valeur de l'énergie cinétique du satellite.

Rappel : l'énergie cinétique se calcule à l'aide de la formule $Ec=(m\times v^2)/2$

Exemples d'utilisation de cette classe :

```
>>> s1 = Satellite('Zoe', masse =250, vitesse =10)
>>> s1.impulsion(500, 15)
>>> s1.affiche_vitesse()
vitesse du satellite Zoe = 40 m/s.
>>> print(s1.energie())
200000
>>> s1.impulsion(500, 15)
>>> s1.affiche_vitesse()
vitesse du satellite Zoe = 70 m/s.
>>> print(s1.energie())
612500
```

12.5 Definissez une classe **Cercle()**. Les objets construits a partir de cette classe seront des cercles de tailles variées. En plus de la méthode constructeur (qui utilisera donc un paramètre **rayon**), vous définirez une méthode **surface()**, qui devra renvoyer la surface du cercle.

Definissez ensuite une classe **Cylindre()** dérivée de la précédente. Le constructeur de cette nouvelle classe comportera les deux paramètres **rayon** et **hauteur**. Vous y ajouterez une méthode **volume()** qui devra renvoyer le volume du cylindre (rappel : volume d'un cylindre = surface de section × hauteur).

Exemple d'utilisation de cette classe :

>>> cyl = Cylindre(5, 7)
>>> print(cyl.surface())
78.54
>>> print(cyl.volume())
549.78

12.6 Complétez l'exercice précédent en lui ajoutant encore une classe **Cone()**, qui devra dériver cette fois de la classe **Cylindre()**, et dont le constructeur comportera lui aussi les deux paramètres **rayon** et **hauteur**. Cette nouvelle classe possèdera sa propre méthode **volume()**, laquelle devra renvoyer le volume du cône (rappel : volume d'un cône= volume du cylindre correspondant divise par 3).
Exemple d'utilisation de cette classe :

>>> co = Cone(5,7)
>>> print(co.volume())
183.26

12.7 Definissez une classe **JeuDeCartes()** permettant d'instancier des objets dont le comportement soit similaire à celui d'un vrai jeu de cartes. La classe devra comporter au moins les quatre méthodes suivantes :

• méthode constructeur : création et remplissage d'une liste de 52 éléments, qui sont eux-mêmes des tuples de 2 entiers. Cette liste de tuples contiendra les caractéristiques de chacune des 52 cartes. Pour chacune d'elles, il faut en effet mémoriser séparément un entier indiquant la valeur (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, les 4 dernières valeurs étant celles des valet, dame, roi et as), et un autre entier indiquant la couleur de la carte (c'est-a-dire 3,2,1,0 pour Cœur, Carreau, Trefle et Pique).
Dans une telle liste, l'élément (11,2) désigne donc le valet de Trefle, et la liste terminée doit être du type :

[(2, 0), (3,0), (3,0), (4,0), ... (12,3), (13,3), (14,3)]

• méthode **nom_carte()** : cette méthode doit renvoyer, sous la forme d'une chaine, l'identité d'une carte quelconque dont on lui a fourni le tuple descripteur en argument. Par exemple, l'instruction : **print(jeu.nom_carte((14, 3)))** doit provoquer l'affichage de : **As de pique**

• méthode **battre()** : comme chacun sait, battre les cartes consiste à les mélanger. Cette méthode sert donc à mélanger les éléments de la liste contenant les cartes, quel qu'en soit le nombre.

• méthode **tirer()** : lorsque cette méthode est invoquée, une carte est retirée du jeu. Le tuple contenant sa valeur et sa couleur est renvoyé au programme appelant. On retire toujours la première carte de la liste. Si cette méthode est invoquée alors qu'il ne reste plus aucune carte dans la liste, il faut alors renvoyer l'objet spécial **None** au programme appelant. Exemple d'utilisation de la classe **JeuDeCartes()** :

```
jeu = JeuDeCartes() # instanciation d'un objet
jeu.battre() # mélange des cartes
for n in range(53): # tirage des 52 cartes :
c = jeu.tirer()
if c == None: # il ne reste plus aucune carte
print('Termine !') # dans la liste
else:
print(jeu.nom_carte(c)) # valeur et couleur de la carte
```

12.8 Complément de l'exercice précédent : définir deux joueurs A et B. Instancier deux jeux
de cartes (un pour chaque joueur) et les mélanger. Ensuite, a l'aide d'une boucle, tirer 52 fois une carte de chacun des deux jeux et comparer leurs valeurs. Si c'est la première des deux qui a la valeur la plus élevée, on ajoute un point au joueur A. Si la situation contraire se présente, on ajoute un point au joueur B. Si les deux valeurs sont égales, on passe au tirage suivant. Au terme de la boucle, comparer les comptes de A et B pour déterminer le gagnant.

12.9 Ecrivez un nouveau script qui récupère le code de l'exercice 12.2 (compte bancaire) en l'important comme un module. Definissez-y une nouvelle classe **CompteEpargne()**, dérivant de la classe **CompteBancaire()** importée, qui permette de créer des comptes d'épargne rapportant un certain intérêt au cours du temps. Pour simplifier, nous admettrons que ces intérêts sont calcules tous les mois.

Le constructeur de votre nouvelle classe devra initialiser un taux d'intérêt mensuel par défaut égal a **0,3 %**. Une méthode **changeTaux(valeur)** devra permettre de modifier ce taux a volonté.

Une méthode **capitalisation(nombreMois)** devra :

• afficher le nombre de mois et le taux d'intérêt pris en compte ;

• calculer le solde atteint en capitalisant les intérêts composes, pour le taux et le nombre de mois qui auront été choisis.

Exemple d'utilisation de la nouvelle classe :

```
>>> c1 = CompteEpargne('Duvivier', 600)
>>> c1.depot(350)
>>> c1.affiche()
Le solde du compte bancaire de Duvivier est de 950 euros.
>>> c1.capitalisation(12)
Capitalisation sur 12 mois au taux mensuel de 0.3 %.
>>> c1.affiche()
Le solde du compte bancaire de Duvivier est de 984.769981274 euros.
>>> c1.changeTaux(.5)
```

>>> c1.capitalisation(12)
Capitalisation sur 12 mois au taux mensuel de 0.5 %.
>>> c1.affiche()
Le solde du compte bancaire de Duvivier est de 1045.50843891 euros.

Exercices

13.1 Modifiez le script ci-dessus de telle manière que le fond d'image devienne bleu clair (*light blue*), que le corps de la résistance devienne beige (*beige*), que le fil de cette résistance soit plus fin, et que les bandes colorées indiquant la valeur soient plus larges.

13.2 Modifiez le script ci-dessus de telle manière que l'image dessinée soit deux fois plus grande.

13.3 Modifiez le script ci-dessus de telle manière qu'il devienne possible d'entrer aussi des valeurs de résistances comprises entre 1 et 10 **Ω**. Pour ces valeurs, le premier anneau colore devra rester noir, les deux autres indiqueront la valeur en **Ω** et dixièmes d' **Ω**.

13.4 Modifiez le script ci-dessus de telle façon que le bouton ≪ Montrer ≫ ne soit plus nécessaire. Dans votre script modifie, il suffira de frapper <*Enter*> après avoir entre la valeur de la résistance, pour que l'affichage s'active.

13.5 Modifiez le script ci-dessus de telle manière que les trois bandes colorées redeviennent noires dans les cas où l'utilisateur fournit une entrée inacceptable.

13.6 Perfectionnez le script décrit ci-dessus, en ajoutant un paramètre **couleur** au constructeur de la classe **Wagon()**, lequel déterminera la couleur de la cabine du wagon.

Arrangez-vous également pour que les fenêtres soient noires au départ, et les roues grises (pour réaliser ce dernier objectif, ajoutez aussi un paramètre **couleur** a la fonction **cercle()**). A cette même classe **Wagon()**, ajoutez encore une méthode **allumer()**, qui servira a changer la couleur des 3 fenêtres (initialement noires) en jaune, afin de simuler l'allumage d'un éclairage intérieur. Ajoutez un bouton a la fenêtre principale, qui puisse déclencher cet allumage. Profitez de l'amélioration de la fonction **cercle()** pour teinter le visage des petits personnages en rose (*pink*), leurs yeux et leurs bouches en noir, et instanciez les objets-wagons avec des couleurs différentes.

13.7 Ajoutez des correctifs au programme précédent, afin que l'on puisse utiliser n'importe quel bouton dans le désordre, sans que cela ne déclenche une erreur ou un effet bizarre.

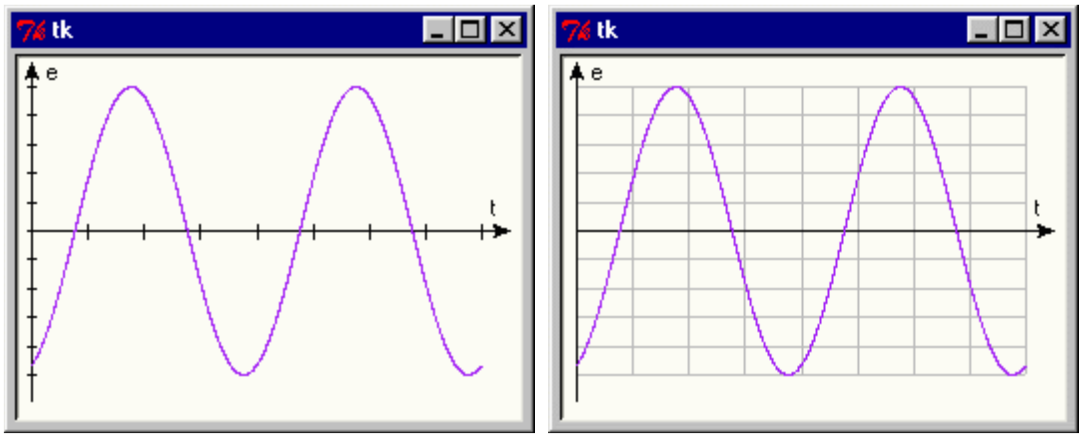
13.8 Créez un quatrième widget, de taille : 400 × 300, couleur de fond : jaune, et faites-y apparaitre plusieurs courbes correspondant à des fréquences et des amplitudes différentes. Il est temps a présent que nous analysions la structure de la classe qui nous

a permis d'instancier tous ces widgets. Nous avons donc enregistré cette classe dans le module **oscillo.py** (voir page 208).

13.9 Modifiez le script de manière a ce que l'axe de référence vertical comporte lui aussi une échelle, avec 5 tirets de part et d'autre de l'origine.

13.10 Comme les widgets de la classe **Canvas()** dont il dérive, votre widget peut intégrer des indications textuelles. Il suffit pour cela d'utiliser la méthode **create_text()**. Cette méthode attend au moins trois arguments : les coordonnées **x** et **y** de l'emplacement ou vous voulez faire apparaitre votre texte et puis le texte lui-même, bien entendu. D'autres arguments peuvent être transmis sous forme d'options, pour préciser par exemple la police de caractères et sa taille. Afin de voir comment cela fonctionne, ajoutez provisoirement la ligne suivante dans le constructeur de la classe **Oscillo- Graphe()**, puis relancez le script :

self.create_text(130, 30, text = "Essai", anchor =CENTER)



Utilisez cette méthode pour ajouter au widget les indications suivantes aux extrémités des axes de référence : **e** (pour ≪ élongation ≫) le long de l'axe vertical, et **t** (pour ≪ temps ≫) le long de l'axe horizontal. Le résultat pourrait ressembler a la figure de gauche page 211.

13.11 Vous pouvez compléter encore votre widget en y faisant apparaitre une grille de référence plutôt que de simples tirets le long des axes. Pour éviter que cette grille ne soit trop visible, vous pouvez colorer ses traits en gris (option **fill = 'grey'**), comme dans la figure de droite de la page 211 .

13.12 Complétez encore votre widget en y faisant apparaitre des repères numériques.

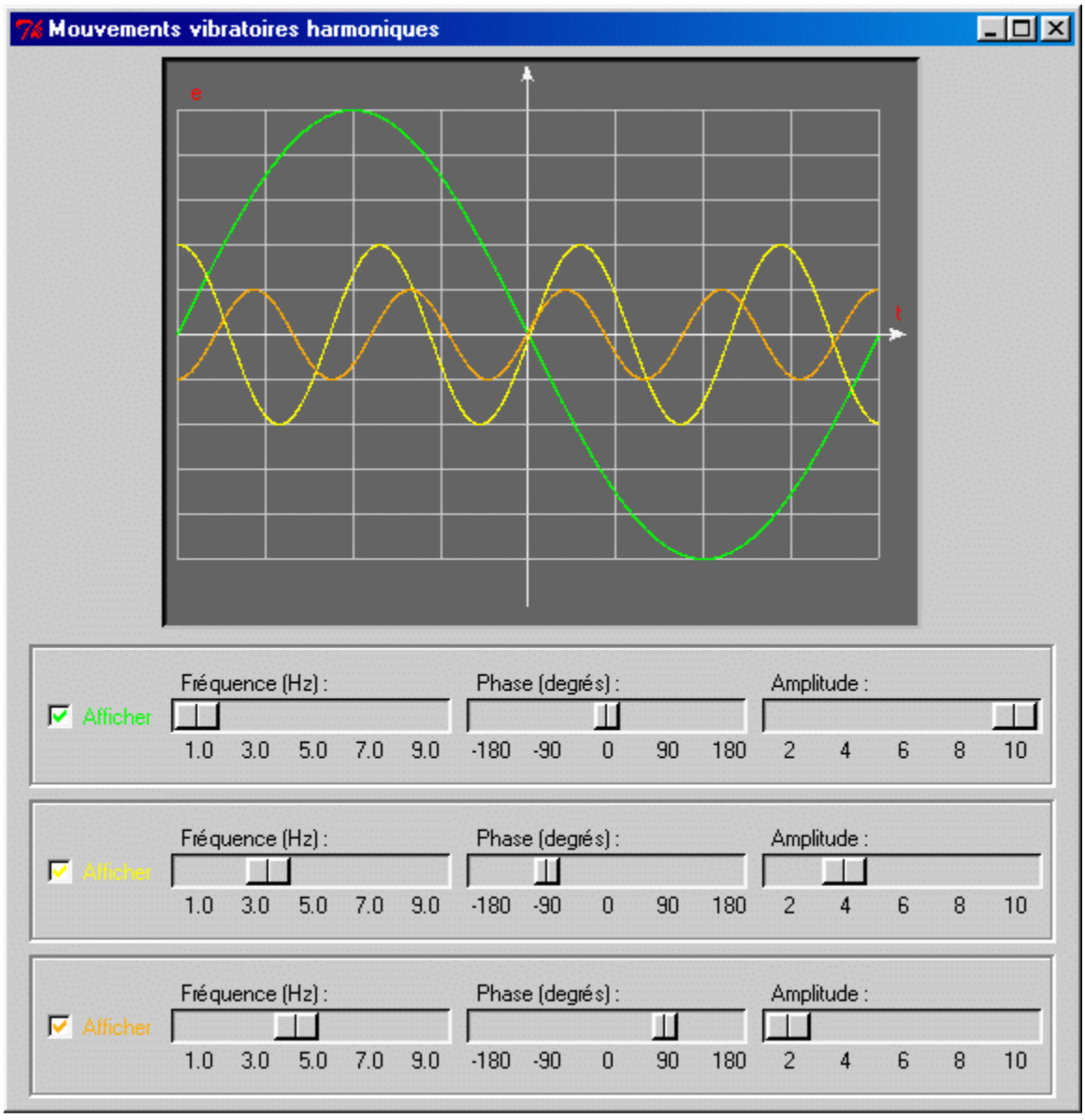
13.13 Votre nouveau widget hérite des propriétés de la classe **Frame()**. Vous pouvez donc modifier son aspect en modifiant les options par défaut de cette classe, a l'aide de la méthode **configure()**. Essayez par exemple de faire en sorte que le panneau de contrôle soit entoure d'une bordure de 4 pixels ayant l'aspect d'un sillon (**bd = 4, relief = GROOVE**). Si vous ne comprenez pas bien ce qu'il faut faire, inspirez-vous du script **oscillo.py** (ligne 10).

13.14 Si l'on assigne la valeur 1 a l'option **showvalue** des widgets **Scale()**, la position précise
du curseur par rapport a l'échelle est affichée en permanence. Activez donc cette fonctionnalité pour le curseur qui contrôle le paramètre ≪ phase ≫.

13.15 L'option **troughcolor** des widgets **Scale()** permet de définir la couleur de leur glissière. Utilisez cette option pour faire en sorte que la couleur des glissières des 3 curseurs soit celle qui est utilisée comme paramètre lors de l'instanciation de votre nouveau widget.

13.16Modifiez le script de telle manière que les widgets curseurs soient écartes davantage les uns des autres (options **padx** et **pady** de la methode **pack()**).

13.17Modifiez le script, de manière à obtenir l'aspect ci-dessous (ecran d'affichage avec grille de référence, panneaux de contrôle entoures d'un sillon) :



13.18Modifiez le script, de manière a faire apparaitre et contrôler 4 graphiques au lieu de trois. Pour la couleur du quatrième graphique, choisissez par exemple : 'blue', 'navy',
'maroon'...

13.19 Aux lignes 33-35, nous récupérons les valeurs des fréquence, phase et amplitude choisies par l'utilisateur sur chacun des trois panneaux de contrôle, en accédant directement aux attributs d'instance correspondants. Python autorise ce raccourci – et c'est bien pratique – mais cette technique est dangereuse. Elle enfreint l'une des recommandations de la théorie générale de la ≪ programmation orientée objet ≫, qui préconise que l'accès aux propriétés des objets soit toujours pris en charge par des méthodes spécifiques. Pour respecter cette recommandation, ajoutez à la classe **ChoixVibra()** une méthode supplémentaire que vous appellerez **valeurs()**, et qui renverra un tuple contenant les valeurs de la fréquence, la phase et l'amplitude choisies. Les lignes 33 à 35 du présent script pourront alors être remplacées par quelque chose comme :

freq, phase, ampl = self.control[i].valeurs()

13.20 Ecrivez une petite application qui fait apparaitre une fenêtre avec un canevas et un widget curseur (**Scale**). Dans le canevas, dessinez un cercle, dont l'utilisateur pourra faire varier la taille à l'aide du curseur.

13.21 Ecrivez un script qui créera deux classes : une classe Application, dérivée de **Frame()**, dont le constructeur instanciera un canevas de 400 × 400 pixels, ainsi que deux boutons. Dans le canevas, vous instancierez un objet de la classe **Visage** décrite ci-après. La classe **Visage** servira à définir des objets graphiques censés représenter des visages humains simplifies. Ces visages seront constitués d'un cercle principal dans lequel trois ovales plus petits représenteront deux yeux et une bouche (ouverte). Une méthode ≪ fermer ≫ permettra de remplacer l'ovale de la bouche par une ligne horizontale. Une méthode ≪ ouvrir ≫ permettra de restituer la bouche de forme ovale. Les deux boutons définis dans la classe **Application** serviront respectivement à fermer et à ouvrir la bouche de l'objet **Visage** installe dans le canevas. Vous pouvez vous inspirer de l'exemple de la page 91 pour composer une partie du code.

13.22 Exercice de synthèse : élaboration d'un dictionnaire de couleurs.

But : réaliser un petit programme utilitaire, qui puisse vous aider a construire facilement et rapidement un nouveau dictionnaire de couleurs, lequel permettrait l'accès technique a une couleur quelconque par l'intermédiaire de son nom usuel en français.

Contexte : En manipulant divers objets colores avec tkinter, vous avez constate que

cette bibliothèque graphique accepte qu'on lui désigne les couleurs les plus fondamentales sous la forme de chaines de caractères contenant leur nom en anglais : *red, blue, yellow,* etc.

Vous savez cependant qu'un ordinateur ne peut traiter que des informations numérisées.

Cela implique que la désignation d'une couleur quelconque doit nécessairement tôt ou tard être encodée sous la forme d'un nombre. Il faut bien entendu adopter pour cela une convention, et celle-ci peut varier d'un système a un autre. L'une de ces conventions, parmi les plus courantes, consiste à représenter une couleur à l'aide de trois octets, qui indiqueront respectivement les intensités des trois composantes rouge, verte et bleue de cette couleur. Cette convention peut être utilisée avec tkinter pour accéder à n'importe quelle nuance colorée. Vous pouvez en effet lui indiquer la couleur d'un élément graphique quelconque, a l'aide d'une chaine de 7 caractères telle que **'#00FA4E'.** Dans cette chaine, le premier caractère (#) signifie que ce qui suit est une valeur hexadécimale. Les six caractères suivants représentent les 3 valeurs hexadécimales des 3 composantes rouge, vert et bleu. Pour visualiser concrètement la correspondance entre une couleur quelconque et son code, vous pouvez explorer les ressources de divers programmes de traitement d'images, tels par exemple les excellents programmes libres ≪ Gimp ≫ et ≪ Inkscape ≫.

Etant donné qu'il n'est pas facile pour les humains que nous sommes de mémoriser de tels codes hexadécimaux, tkinter est également dote d'un dictionnaire de conversion, qui autorise l'utilisation de noms communs pour un certain nombre de couleurs parmi les plus courantes, mais cela ne marche que pour des noms de couleurs exprimes en anglais.

Le but du present exercice est de realiser un logiciel qui facilitera la construction d'un dictionnaire equivalent en francais, lequel pourrait ensuite etre incorpore a l'un ou l'autre de vos propres programmes. Une fois construit, ce dictionnaire serait donc de laforme :

{'vert':'#00FF00', 'bleu':'#0000FF', ... etc ...}.

Cahier des charges :

L'application à réaliser sera une application graphique, construite autour d'une classe. Elle comportera une fenêtre avec un certain nombre de champs d'entrée et de boutons, afin que l'utilisateur puisse aisément encoder de nouvelles couleurs en indiquant à chaque fois son nom français dans un champ, et son code hexadécimal dans un autre. Lorsque le dictionnaire contiendra déjà un certain nombre de données, il devra être possible de le tester, c'est-à-dire d'entrer un nom de couleur en français et de retrouver le code hexadécimal correspondant à l'aide d'un bouton (avec

affichage éventuel d'une zone colorée). Un bouton provoquera l'enregistrement du dictionnaire dans un fichier texte. Un autre permettra de reconstruire le dictionnaire à partir du fichier.

13.23 Le script ci-dessous correspond à une ébauche de projet dessinant des ensembles de dés à jouer disposes à l'écran de plusieurs manières différentes (cette ébauche pourrait être une première étape dans la réalisation d'un logiciel de jeu). L'exercice consistera à analyser ce script et à le compléter. Vous vous placerez ainsi dans la situation d'un programmeur charge de continuer le travail commence par quelqu'un d'autre, ou encore dans celle de l'informaticien prie de participer à un travail d'équipe.

A) Commencez par analyser ce script et ajoutez-y des commentaires, en particulier aux lignes marquées : **#*****, pour montrer que vous comprenez ce que doit faire le programme a ces emplacements :

```
from tkinter import *
class FaceDom(object):
def __init__(self, can, val, pos, taille =70):
self.can =can
# ***
x, y, c = pos[0], pos[1], taille/2
can.create_rectangle(x -c, y-c, x+c, y+c, fill ='ivory', width =2)
d = taille/3
# ***
self.pList =[]
# ***
pDispo = [((0,0),), ((-d,d),(d,-d)), ((-d,-d), (0,0), (d,d))]
disp = pDispo[val -1]
# ***
for p in disp:
self.cercle(x +p[0], y +p[1], 5, 'red')
def cercle(self, x, y, r, coul):
# ***
self.pList.append(self.can.create_oval(x-r, y-r, x+r, y+r, fill=coul))
def effacer(self):
# ***
for p in self.pList:
self.can.delete(p)
class Projet(Frame):
def __init__(self, larg, haut):
```

```
Frame.__init__(self)
self.larg, self.haut = larg, haut
self.can = Canvas(self, bg='dark green', width =larg, height =haut)
self.can.pack(padx =5, pady =5)
# ***
bList = [("A", self.boutA), ("B", self.boutB),
("C", self.boutC), ("D", self.boutD),
("Quitter", self.boutQuit)]
for b in bList:
Button(self, text =b[0], command =b[1]).pack(side =LEFT)
self.pack()
def boutA(self):
self.d3 = FaceDom(self.can, 3, (100,100), 50)
def boutB(self):
self.d2 = FaceDom(self.can, 2, (200,100), 80)
def boutC(self):
self.d1 = FaceDom(self.can, 1, (350,100), 110)
def boutD(self):
# ***
self.d3.effacer()
def boutQuit(self):
self.master.destroy()
Projet(500, 300).mainloop()
```

B) Modifiez ensuite ce script, afin qu'il corresponde au cahier des charges suivant :
Le canevas devra être plus grand : 600 × 600 pixels. Les boutons de commande devront être déplaces à droite et espaces davantage. La taille des points sur une face de dé devra varier proportionnellement à la taille de cette face.

Variante 1 :

Ne conservez que les 2 boutons A et B. Chaque utilisation du bouton A fera apparaitre 3 nouveaux des (de même taille, plutôt petits) disposes sur une colonne (verticale), les valeurs de ces des étant tirées au hasard entre 1 et 6. Chaque nouvelle colonne sera disposée à la droite de la précédente. Si l'un des tirages de 3 des correspond à 4, 2, 1 (dans n'importe quel ordre), un message ≪ gagne ≫ sera affiche dans la fenêtre (ou dans le canevas). Le bouton B provoquera l'effacement complet (pas seulement les points !) de tous les des affiches.

Variante 2 :

Ne conservez que les 2 boutons A et B. Le bouton A fera apparaitre 5 des disposes en

quinconce (c'est-à-dire comme les points d'une face de valeur 5). Les valeurs de ces des seront tirées au hasard entre 1 et 6, mais il ne pourra pas y avoir de doublons. Le bouton B provoquera l'effacement complet (pas seulement les points !) de tous les des affiches.

Variante 3 :

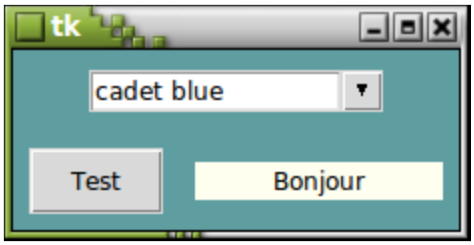
Ne conservez que les 3 boutons A, B et C. Le bouton A fera apparaitre 13 des de même taille disposes en cercle. Chaque utilisation du bouton B provoquera un changement de valeur du premier de, puis du deuxième, du troisième, etc. La nouvelle valeur d'un de sera à chaque fois égale a sa valeur précédente augmentée d'une unité, sauf dans le cas où la valeur précédente était 6 : dans ce cas la nouvelle valeur est 1, et ainsi de suite. Le bouton C provoquera l'effacement complet (pas seulement les points !) de tous les des affiches.

Variante 4 :

Ne conservez que les 3 boutons A, B et C. Le bouton A fera apparaitre 12 des de même taille disposes sur deux lignes de 6. Les valeurs des dés de la première ligne seront dans l'ordre 1, 2, 3, 4, 5, 6. Les valeurs des dés de la seconde ligne seront tirées au hasard entre 1 et 6. Chaque utilisation du bouton B provoquera un changement de valeur aléatoire du premier de de la seconde ligne, tant que cette valeur restera différente de celle du de correspondant dans la première ligne. Lorsque le 1er de de la 2e ligne aura acquis la valeur de son correspondant, c'est la valeur du 2e de de la seconde ligne qui sera changée au hasard, et ainsi de suite, jusqu'a ce que les 6 faces du bas soient identiques à celles du haut. Le bouton C provoquera l'effacement complet (pas seulement les points !) de tous les des affiches.

Exercice

14.1 Perfectionnez le widget ≪ combo box simplifie ≫ décrit à la page 235, de manière à ce que la liste soit cachée au départ, et qu'un petit bouton à droite du champ d'entrée en provoque l'apparition. Vous devrez pour ce faire placer la liste et son ascenseur dans une fenêtre satellite sans bordure (Cf. widget Toplevel, page 246), positionner celle-ci correctement (il vous faudra probablement consulter les sites web traitant de Tkinter pour trouver les informations nécessaires, mais cela fait partie de votre apprentissage !), et vous assurer que cette fenêtre disparaisse après que l'utilisateur ait sélectionne un item dans la liste.



Exercices

16.1 Avant d'aller plus loin, et à titre d'exercice de synthèse, nous allons vous demander de créer entièrement vous-même une base de données ≪ Musique ≫ qui contiendra les deux tables suivantes (cela représente un certain travail, mais il faut que vous puissiez disposer d'un certain nombre de données pour pouvoir expérimenter valablement les fonctions de recherche et de tri prises en charge par le SGBDR) :

Oeuvres
comp (chaîne)
titre (chaîne)
duree (entier)
interpr (chaîne)

Compositeurs
comp (chaîne)
a_naiss (entier)
a_mort (entier)

Commencez à remplir la table **Compositeurs** avec les données qui suivent (et profitez de cette occasion pour faire la preuve des compétences que vous maitrisez déjà, en écrivant un petit script pour vous faciliter l'entrée des informations : une boucle s'impose !)

comp a_naiss a_mort
Mozart 1756 1791
Beethoven 1770 1827
Haendel 1685 1759
Schubert 1797 1828
Vivaldi 1678 1741
Monteverdi 1567 1643
Chopin 1810 1849
Bach 1685 1750
Shostakovich 1906 1975

Dans la table **oeuvres**, entrez les donnees suivantes :

comp titre duree interpr
Vivaldi Les quatre saisons 20 T. Pinnock
Mozart Concerto piano N°12 25 M. Perahia
Brahms Concerto violon N°2 40 A. Grumiaux
Beethoven Sonate "au clair de lune" 14 W. Kempf
Beethoven Sonate "pathetique" 17 W. Kempf
Schubert Quintette "la truite" 39 SE of London
Haydn La creation 109 H. Von Karajan
Chopin Concerto piano N°1 42 M.J. Pires
Bach Toccata & fugue 9 P. Burmester
Beethoven Concerto piano N°4 33 M. Pollini
Mozart Symphonie N°40 29 F. Bruggen
Mozart Concerto piano N°22 35 S. Richter
Beethoven Concerto piano N°3 37 S. Richter

Les champs **a_naiss** et **a_mort** contiennent respectivement l'année de naissance et l'année de la mort des compositeurs. La durée des œuvres est fournie en minutes. Vous pouvez évidemment ajouter autant d'enregistrements d'œuvres et de compositeurs que vous le voulez, mais ceux qui précédent devraient suffire pour la suite de la démonstration. Pour ce qui va suivre, nous supposerons donc que vous avez effectivement encode les données des deux tables décrites ci-dessus. Si vous éprouvez des difficultés à écrire le script nécessaire, veuillez consulter le corrige de l'exercice 16.1, à la page 430. Le petit script ci-dessous est fourni à titre purement indicatif. Il s'agit d'un client SQL rudimentaire, qui vous permet de vous connecter à la base de données ≪ musique ≫ qui devrait à présent exister dans l'un de vos répertoires, d'y ouvrir un curseur et d'utiliser celui-ci pour effectuer des requetés. Notez encore une fois que rien n'est transcrit sur le disque tant que la méthode **commit()** n'a pas été invoquée.

```
# Utilisation d'une petite base de données acceptant les requêtes SQL
import sqlite3
baseDonn = sqlite3.connect("musique.sq3")
cur = baseDonn.cursor()
while 1:
print("Veuillez entrer votre requete SQL (ou <Enter> pour terminer) :")
requete = input()
if requete =="":
break
try:
```

```
cur.execute(requete) # exécution de la requête SQL
except:
print('*** Requete SQL incorrecte ***')
else:
for enreg in cur: # Affichage du résultat
print(enreg)
print()
choix = input("Confirmez-vous l'enregistrement de l'etat actuel (o/n) ? ")
if choix[0] == "o" or choix[0] == "O":
baseDonn.commit()
else:
baseDonn.close()
```

Cette application très simple n'est évidemment qu'un exemple. Il faudrait y ajouter la possibilité de choisir la base de données ainsi que le répertoire de travail. Pour éviter que le script ne se ≪ plante ≫ lorsque l'utilisateur encode une requête incorrecte, nous avons utilisé ici le traitement des *exceptions* déjà décrit à la page 122.

16.2 Modifiez le script décrit dans ces pages de manière à ajouter une table supplémentaire a la base de données. Ce pourrait être par exemple une table ≪ orchestres ≫, dont chaque enregistrement contiendrait le nom de l'orchestre, le nom de son chef, et le nombre total d'instruments.

16.3 Ajoutez d'autres types de champ a l'une des tables (par exemple un champ de type *float* (reel) ou de type *date*), et modifiez le script en conséquence.

Exercices

Le script précédent peut vous servir de banc d'essai pour exercer vos compétences dans un grand nombre de domaines.

17.1 Comme explique précédemment, on peut structurer un site web en le fractionnant en plusieurs classes. Il serait judicieux de séparer les méthodes concernant les ≪ clients ≫ et les ≪ administrateurs ≫ de ce site dans des classes differentes.

17.2 Le script tel qu'il est ne fonctionne à peu près correctement que si l'utilisateur remplit correctement tous les champs qui lui sont proposés. Il serait donc fort utile de lui ajouter une série d'instructions de contrôle des valeurs encodées, avec renvoi de messages d'erreur a l'utilisateur lorsque c'est nécessaire.

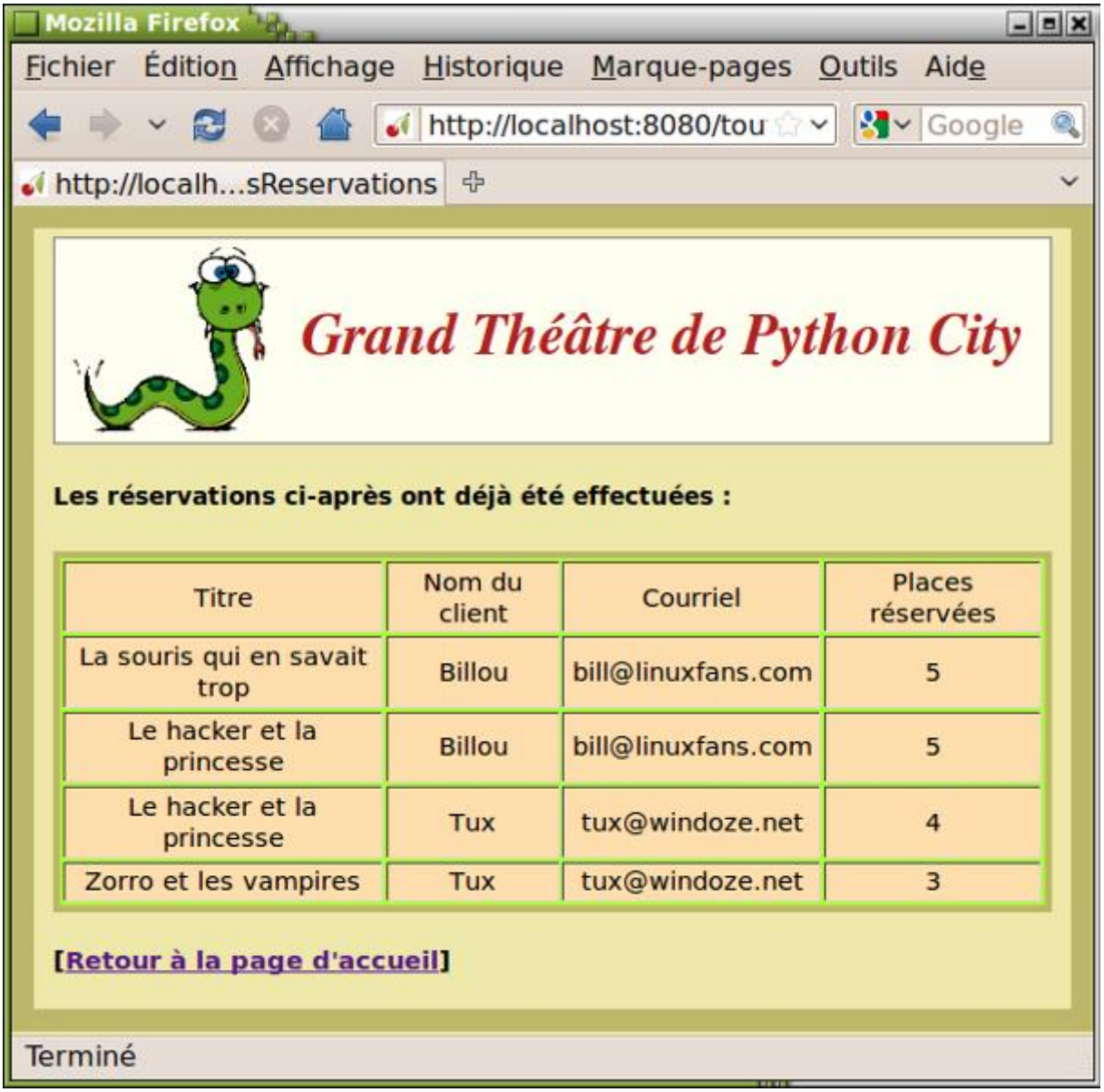
17.3 L'accès ≪ administrateurs ≫ permet seulement d'ajouter de nouveaux spectacles, mais non de modifier ou de supprimer ceux qui sont déjà encodes. Ajoutez donc des méthodes pour implémenter ces fonctions.

17.4 L'accès administrateur est libre. Il serait judicieux d'ajouter au script un mécanisme d'authentification par mot de passe, afin que cet accès soit réservé aux seules personnes possédant le sésame.

17.5 L'utilisateur ≪ client ≫ qui se connecte plusieurs fois de suite, est à chaque fois mémorisé comme un nouveau client, alors qu'il devrait pouvoir ajouter d'autres réservations à son compte existant, éventuellement modifier ses données personnelles, etc. Toutes ces fonctionnalités pourraient être ajoutées.

17.6 Vous aurez probablement remarque que les ***tableaux*** HTML génères dans le script sont produits à partir d'algorithmes très semblables. Il serait donc intéressant d'écrire une fonction généraliste capable de produire un tel tableau, dont on recevrait la description dans un dictionnaire ou une liste.

17.7 La décoration des pages web générées par le script est définie dans une feuille de style annexe (le fichier *spectacles.css*). Libre à vous d'examiner ce qui se passe si vous enlevez le lien activant cette feuille de style (5e ligne du fichier *spectacles.htm*), ou si vous modifiez son contenu, lequel décrit le style à appliquer à chaque balise.



Exercices

18.1 Simplifiez le script correspondant au client de *chat* décrit à la page 345, en supprimant l'un des deux objets threads. Arrangez-vous par exemple pour traiter l'émission de messages au niveau du thread principal.

18.2 Modifiez le jeu des bombardes (version monoposte) du chapitre 15 (voir pages 265 et suivantes), en ne gardant qu'un seul canon et un seul pupitre de pointage. Ajoutez-y une cible mobile, dont le mouvement sera gère par un objet thread

indépendant (de manière a bien séparer les portions de code qui contrôlent l’animation de la cible et celle du boulet).