Introduction à la programmation — Module M02 : Travaux pratiques

IUT de Villetaneuse — Licence Pro ASSUR

Camille Coti
camille.coti@iutv.univ-paris13.fr

1 Prise en main de l'environnement de programmation

Dans votre répertoire utilisateur, créez un répertoire `M02`. Ensuite, pour chaque TP, vous créerez dans ce répertoire un répertoire `TPX`, où X est le numéro du TP. Par exemple, aujourd'hui vous devez créer un répertoire `TP1`. Dans ce répertoire, vous créerez un répertoire par exercice.

Exercice 1.1 : Manipulations dans l'interpréteur interactif

1. Lancez l'interpréteur interactif avec la commande `python`. Vous devriez voir s'afficher quelques informations sur l'interpréteur (version, compilateur utilisé pour le compiler...) et une invite de commandes.
2. Vous pouvez taper des commandes dans cette session interactive. Le fait de rentrer seulement une expression l'évalue et l'affiche. Les affectations ne sont pas affichées. On effectue un affichage avec la fonction `print`. Par exemple :

```
1 Python 2.7.3rc2 (default, Apr 22 2012, 22:30:17)
2 [GCC 4.6.3] on linux2
3 Type "help", "copyright", "credits" or "license" for more
    information.
4 >>> 1
5 1
6 >>> print i
7 5
8 >>>
```

3. Qu'affichent les programmes suivants ?

```
1 print 1
2 print 2
3 print 3
```

```
1 print 2
2 print 1
3 print 3
```

Vous pouvez quitter l'interpréteur interactif avec la combinaison de touches Ctrl+d.

Exercice 1.2 : Écriture de scripts Python

Les programmes Python peuvent être écrits dans des fichiers texte, qui sont interprétés par l'interpréteur Python. Dans cet exercice, nous allons voir différentes façon de les exécuter.

Dans cet exercice, vous éditerez les fichiers contenant votre code avec un éditeur de texte brut, c'est-à-dire qui enregistre sur le disque exactement ce qui a été saisi par l'utilisateur. Attention à utiliser un éditeur qui n'insère pas d'informations de formattage dans les fichiers sauvegardés. Par exemple, vous pourrez utiliser emacs, gedit, vim, ou les environnements de développement intégrés eric ou Eclipse.

1. Créez un fichier dans lequel vous mettez quelques instructions Python, par exemple un affichage. Enregistrez-le sous le nom `script1.py`.
2. La première façon d'exécuter un script Python consiste à le faire lire et exécuter par l'interpréteur. Exécutez votre script en le passant en paramètre de la commande `python` :

```
$ python ./script1.py
```

3. On peut intégrer dans le script le chemin vers l'interpréteur qui va exécuter la série d'instructions composant le programme. Pour cela, on insère au début du fichier un *shebang* qui donne le chemin vers l'interpréteur à utiliser. C'est notamment très utile sur les systèmes où plusieurs version de Python sont installées.
   Avec l'aide de la commande Unix `which`, trouvez le chemin absolu vers l'interpréteur Python appelé par la commande `python`.
4. Créez un nouveau fichier dans lequel vous mettez quelques instructions Python et commençant par le shebang constitué grâce à la question précédente. Enregistrez-le sous le nom `script2.py`.
5. Essayez de l'exécuter directement. Que constatez-vous ?
6. Remédiez à ce problème et exécutez le script
7. Modifiez le shebang pour mettre un chemin ne correspondant à *aucun* interpréteur Python. Essayez d'exécuter le script. Que constatez-vous ?

Exercice 1.3 : Valeurs de variables

1. Recopiez et enregistrez dans le fichier `script3.py` le programme suivant.

```
#!/usr/bin/python

x = 1
y = 2
print x
z = x + y
print z
```

2. Exécutez-le. Quel est l'affichage obtenu ?
3. Expliquez chacune de ses lignes.

Exercice 1.4 : Inversion de valeurs

Écrivez un script Python qui :
— initialise deux variables
— affiche leurs valeurs
— inverse les valeurs de ces deux variables
— affiche leurs valeurs

Exercice 1.5 : Concaténation de chaînes de caractères

Considérons le programme suivant :

```
#!/usr/bin/python

cp = 93
nom = 'Seine Saint-Denis'
phrase = ?????
print phrase
```

Que faut-il mettre à la place des `?????` pour que le programme affiche "Le code postal de la Seine Saint-Denis est 93" ?

Exercice 1.6 : Types et opérations

Donnez la valeur des opérations suivantes :

— 3 + 4
— '3' + '4'
— 3 + '4'
— '3' + 4
— 3 * 4
— '3' * '4'
— 3 * '4'
— '3' * 4
— '3' * 4.0

Exercice 1.7 : Saisie d'une valeur

La fonction `raw_input()` permet de lire une entrée saisie au clavier par l'utilisateur.

1. Écrivez un programme qui demande à l'utilisateur de saisir une valeur et affiche la valeur saisie.
2. La fonction `raw_input()` peut prendre comme paramètre une chaîne de caractères qui sera affichée au début de la ligne où l'utilisateur sera invité à saisir quelque chose. Modifiez votre programme pour qu'il affiche `-->` au début de ligne.
3. De quel type sont les données lues par `raw_input()` ? Modifiez votre programme pour afficher leur type.

Exercice 1.8 : Valeur absolue

Écrivez un programme qui demande à l'utilisateur de saisir un nombre et affiche sa valeur absolue.

Exercice 1.9 : Catégories sportives

Les catégories d'âges dans les compétitions sportives sont les suivantes :

— 12 ans et moins : trop jeune
— 13-14 ans : minime
— 15-16 ans : cadet
— 17-18 ans : junior
— 19-34 ans : senior
— plus de 35 ans : vétéran

Écrivez un programme qui demande à l'utilisateur de saisir son âge et affiche sa catégorie.

Exercice 1.10 : Le temps plus une seconde

Écrivez un programme qui demande à l'utilisateur l'heure, les minutes et les secondes, et affiche l'heure qu'il sera une seconde plus tard.

Par exemple, si l'utilisateur tape 21 puis 32 puis 8, le programme doit afficher : "Dans une seconde, il sera 21 heure(s) 32 minute(s) et 9 secondes".

On commencera par vérifier que l'utilisateur entre une heure valide.

Exercice 1.11 : Factorielle

La factorielle d'un nombre entier, noté $n!$, a pour formule $n! = n \times (n-1) \times (n-2) \times ... \times 2 \times 1 = \prod_{k=1}^{n} k$.

1. Écrivez un programme qui demande à l'utilisateur de saisir un nombre entier et calcule sa factorielle.
2. Afin de ne pas se lancer dans des calculs trop longs, on veut limiter la valeur maximale pouvant être saisie à 100. De plus, la factorielle ne se calcule que pour des entiers strictement positifs. Modifiez votre programme afin de vérifier si ces conditions sur le nombre saisi par l'utilisateur sont bien réalisées, et ne calculer la factorielle que dans ce cas.

2 Tests, boucles, collections

Exercice 2.1 : Affichage des valeurs d'une liste

Écrire un programme qui affecte à une variable `semaine` une liste des noms des jours de la semaine et les affiche tous en parcourant la liste.

Exercice 2.2 : Minimum et maximum

Écrire un programme qui affecte à une variable `tab` une liste d'entiers et affiche le minimum et le maximum contenus dans cette liste.

Exercice 2.3 : Addition de vecteurs

Si A et B sont deux vecteurs de taille n, on les additionne en additionnant leurs termes deux à deux de la façon suivante : $C = A + B \Leftrightarrow \forall i \in [0, n-1] : C_i = A_i + B_i$

Par exemple, $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} 7 \\ 8 \\ 9 \end{pmatrix} = \begin{pmatrix} 8 \\ 10 \\ 12 \end{pmatrix}$

Écrire une fonction qui additionne deux vecteurs passés en paramètres. La première chose faite par cette fonction consiste à vérifier que les deux vecteurs sont bien de même taille. Si ce n'est pas le cas, la fonction retourne un vecteur vide (ne contenant rien). Si tout va bien, la fonction effectue l'addition et retourne le vecteur résultat.

Exercice 2.4 : Recherche dans un tableau

1. Écrire un programme qui demande à l'utilisateur de saisir un mot et indique si ce mot est présent dans le tableau `[ 'chien', 'chat', 'girafe', 'hippopotame', 'chien', 'singe' ]` .
2. Modifier ce programme de façon à afficher le nombre de fois où le mot apparaît dans le tableau.

Exercice 2.5 : Séparer les longs et les courts

Écrire un programme qui parcourt la liste `[ 'Jean', 'Maximilien', 'Brigitte', 'Sonia', 'Jean-Pierre', 'Sandra' ]` et met les mots de moins de 6 lettres dans une liste `courts` et les mots de 6 lettres ou plus dans une liste `longs`.

Votre programme affichera ensuite le nombre d'éléments dans chacune des listes et leurs éléments.

Vous pouvez obtenir la longueur d'une chaîne de caractères et le nombre d'éléments dans une liste avec la fonction `len()`.

Exercice 2.6 : Répertoire téléphonique

On veut écrire un programme qui maintient un répertoire téléphonique pour l'utilisateur. Les entrées dans le répertoire sont stockées sous forme d'un couple associant le nom de la personne à son numéro de téléphone, enregistré sous forme d'un entier. Par exemple, on pourra associer l'entier 0123456789 à la chaîne de caractères "Jean Talu".

1. Quelle structure de données serait la plus adaptée pour représenter les données du répertoire ?
2. Le programme doit proposer un menu à l'utilisateur lui permettant de choisir parmi les possibilités suivantes :
   — Ajouter une entrée dans le répertoire
   — Afficher le numéro de téléphone d'une personne en particulier
   — Afficher le nombre de numéros enregistrés dans le répertoire
   — Afficher le contenu de tout le répertoire
   — Supprimer du répertoire une personne et son numéro
   — Effacer tout le contenu du répertoire
   — Quitter le programme

   Écrire une fonction qui affiche la liste des choix et demande à l'utilisateur de saisir une valeur lui permettant de choisir l'action à effectuer (par exemple, un entier). La fonction doit retourner cet entier.
3. Le programme doit afficher ce menu et proposer à l'utilisateur de choisir une action à effectuer, effectuer l'action demandée et réafficher ce menu tant que l'utilisateur n'a pas demandé à quitter. Écrire pour chacun de ces choix une fonction qui affiche simplement l'action choisie, et les appeler en conséquence du choix de l'utilisateur. Si l'utilisateur entre une valeur autre que celles proposées, le programme affiche une erreur et lui demande à nouveau d'entrer une valeur.
4. Implémenter la fonction qui permet d'ajouter une entrée dans le répertoire. Cette fonction demande à l'utilisateur de saisir un nom et un numéro de téléphone et les insère dans le répertoire.
5. Pour tester si votre fonction d'ajout d'une personne fonctionne correctement, la vérification la plus élémentaire consiste à regarder si le répertoire contient bien une entrée de plus. Implémenter la fonction qui affiche le nombre d'entrées dans le répertoire. Pour tester si cette fonction et la précédente fonctionnent correctement, vous pourrez exécuter votre programme en choisissant dans un permier temps de saisir une nouvelle entrée, puis en choisissant d'afficher le nombre d'entrées dans le répertoire.
6. Implémenter la fonction qui affiche toutes les entrées du répertoire. L'affichage pourra ressembler à :

```
1 Mehdi Doua : 1234554321
2 Alfonse Danlmur : 987678902
3 Jean Talu : 987654321
```

   Vous pouvez maintenant tester le bon fonctionnement de la fonction d'ajout d'une entrée dans le répertoire en affichant tout le contenu du répertoire.
7. Implémenter la fonction qui affiche le numéro d'une personne en particulier. Cette fonction demander à l'utilisateur de saisir un nom, et si une entrée correspondant à ce nom existe dans le répertoire, son numéro est affiché. Sinon, un message d'erreur est affiché.
8. Implémenter la fonction qui supprime une entrée du répertoire. Cette fonction demande à l'utilisateur de saisir le nom de la personne à supprimer du répertoire. Si cette entrée est présente dans le répertoire, elle est supprimée. Sinon, un message d'erreur est affiché.
9. Implémenter la fonction qui efface tout le répertoire. Vous pourrez la tester en vérifiant qu'après effacement, le répertoire ne contient aucune entrée.

Exercice 2.7 : Mots de passe corrects

On souhaite écrire un programme qui vérifie si une liste de mots de passe vérifie un certain nombre de conditions. Chaque mot de passe devra :

— Contenir au moins une lettre minuscule
— Contenir au moins une lettre majuscule
— Contenir au moins un chiffre
— Contenir au moins un caractère spécial parmi [*#+@]
— Avoir une longueur minimale de 4 caractères
— Avoir une longueur maximale de 6 caractères
— Ne pas contenir d'espace

Les mots de passe corrects seront affichés à l'utilisateur.

Par exemple, si l'utilisateur saisit : `Abc@1,a B1#,2w3E*,2We#3345`, alors l'affichage résultant sera `Abc@1,2w3E*`.

1. L'utilisateur saisit la série de mots de passe séparés par des virgules. La première chose à faire est de séparer les mots de passe et de les stocker dans une liste.
2. Ensuite, une fonction est appelée pour chaque mot de passe (donc chaque élément de la liste) afin de vérifier si ce mot de passe remplit les conditions énoncées précédemment. Cette fonction retourne un booléen : `True` si le mot de passe est valable, `False` si le mot de passe n'est pas valable. Les mots de passe valables sont stockés dans une liste.

   Dans un premier temps, vous utiliserez une fonction *bouchon* : cette fonction aura le squelette de la fonction de vérification, mais retournera simplement une valeur possible (`True` ou `False`). L'idée est de mettre en place une fonction et son appel dans le programme, pour ensuite l'écrire ultérieurement.
3. Enfin, une fois tous les mots de passe vérifiés, la liste des mots de passe valables est affichée en les séparant par des virgules.
4. Vous allez maintenant implémenter la fonction qui vérifie si un mot de passe est valable ou non. Vous avez écrit précédemment une fonction bouchon : vous allez maintenant écrire le corps de cette fonction.
5. La fonction de vérification doit commencer par regarder si le mot de passe passé en paramètre contient au moins une lettre minuscule. Écrivez une fonction qui prend un mot en paramètre et retourne `True` si ce mot contient *au moins* une lettre minuscule, et `False` si il n'en contient aucune. Appelez cette fonction depuis la fonction globale de vérification.

   Indication : vous pouvez transformer une chaîne de caractères en tableau de caractères en utilisant la fonction `list()`. Vous pouvez obtenir une chaîne de caractères contenant toutes les lettres minuscules de l'alphabet en utilisant le module `string` et sa constante `string.lowercase`. Pour cela, vous devrez importer le module :

```
1 import string
2 minuscules = list( string.lowercase )
```

6. On doit ensuite vérifier si le mot de passe contient au moins une lettre majuscule. De la même façon, écrivez une fonction qui vérifie si la chaîne de caractères passée contient au moins une lettre majuscule et retourne `True` si elle en contient au moins une, et `False` si il n'en contient aucune.
7. On doit ensuite vérifier si le mot de passe contient au moins un chiffre et au moins un caractères spécial dans la liste fournie. Écrivez ces deux fonctions.
8. On doit ensuite vérifier que le mot de passe est bien d'une longueur acceptable. Écrivez une fonction qui retourne `True` si la longueur est comprise entre 4 et 6 caractères (en incluant ces valeurs), ou `False` sinon.
9. La dernière vérification consiste à s'assurer que le mot de passe ne contient aucun espace. Écrivez une fonction qui retourne `True` si la chaîne de caractères passée ne contient aucun espace, `False` si elle en contient au moins un.
10. Votre fonction de vérification doit effectuer toutes ces vérifications sur la chaîne de caractères qui lui a été passée en paramètre. Appelez les fonctions que vous venez d'implémenter. La

fonction de vérification doit retourner `False` si *au moins* une de ces conditions n'est pas respectée, ou `True` si elles sont *toutes* respectées.

3 Programmation graphique avec la tortue

Le module Turtle permet de dessiner des figures à l'écran. Afin de pouvoir utiliser ce module, il faut ajouter au début du programme l'instruction :

```
1 from turtle import *
```

On dispose alors des fonctions suivantes permettant de dessiner :
— `forward( distance )` : avancer d'une distance donnée
— `left( angle )` : tourner d'un certain angle vers la gauche
— `right( angle )` : tourner d'un certain angle vers la droite
— `up()` : relever le crayon (pour pouvoir avancer sans dessiner)
— `down()` : abaisser le crayon (pour recommencer à dessiner)
— `goto( w, y )` : aller à l'endroit de coordonnées `x` et `y`.
La documentation de ce module se trouve à l'adresse :

`http://docs.python.org/library/turtle.html`

La fenêtre dans laquelle le dessin est tracé se ferme immédiatement après la dernière instruction. Pour attendre quelques secondes, vous pourrez utiliser la fonction `sleep()` du module `time`. Pour cela, ajoutez au début du programme :

```
1 from time import *
```

Et, à l'endroit où vous souhaitez mettre votre programme en pause pendant `nbsec` secondes :

```
1 sleep( nbsec )
```

Exercice 3.1 : Lignes

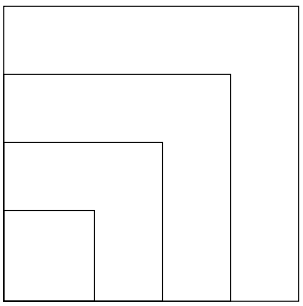
Que dessine le programme suivant ?

```
1  #!/usr/bin/python
2
3  from turtle import *
4
5  forward( 100 )
6  left( 120 )
7  forward( 100 )
8  left( 120 )
9  forward( 100 )
10 left( 60 )
11 forward( 100 )
12 left( 120 )
13 forward( 100 )
```

Exercice 3.2 : Autour du carré

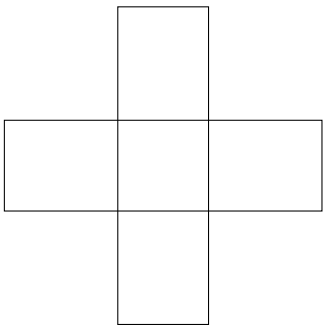
1. Écrivez une fonction Python qui dessine un carré de côté n

2. Modifiez votre fonction afin d'utiliser une boucle `for`
3. Écrivez une fonction qui dessine la figure suivante. Les paramètres de la fonction sont le nombre de carrés, la longueur du petit côté et la différence entre les longueurs de deux carrés consécutifs.



Exercice 3.3 : Autour du rectangle

1. Écrivez une fonction qui dessine un rectangle. Les paramètres de cette fonction sont les longueurs des côtés.
2. Utilisez cette fonction pour dessiner la figure suivante :

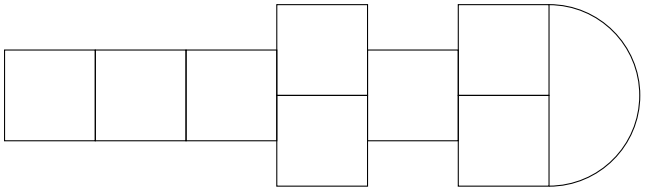


Exercice 3.4 : Cercle

La fonction `circle` permet de tracer un cercle.

1. Avec l'aide de la documentation du module turtle, comprenez son utilisation et écrivez un programme Python qui trace un cercle.
2. Écrivez un programme Python qui trace la figure suivante :

3. Écrivez un programme Python qui dessine une marelle.

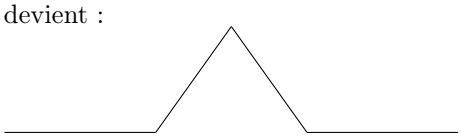


Exercice 3.5 : Fractales

Une fractale est une structure fragmentée qui se répète en elle-même. Par exemple, Si l'on considère un segment, on peut le couper en 3 parties et dessiner la pointe d'un triangle isocèle sur sa partie médiane.

Ainsi, le segment suivant :

devient :



Chacun de ces segments ainsi créés peuvent subir la même transformation :

Et ainsi de suite : chaque segment peut être découpé en segments, qui seront eux-mêmes découpés. Les fractales sont donc des structures intrinsèquement récursives.

En utilisant la tortue, dessiner une fractale telle que celle décrite ci-dessus. Vous arrêterez la récurrence à une profondeur arbitraire, par exemple 5 appels récursifs.

4 Exceptions

Exercice 4.1 : Un quotient sûr

Nous avons vu en cours la fonction `quotient()`, qui peut soulever l'exception `ZeroDivisionError` lorsque le diviseur est égal à zéro.

1. Écrivez en Python une fonction `quotient()` similaire, qui prend en paramètre deux nombres et retourne leur quotient (un réel). Si le diviseur est égal à zéro, cette fonction devra soulever l'exception `ZeroDivisionError`.
2. Écrivez en Python une fonction qui demande à l'utilisateur de saisir deux entiers et appelle votre fonction `quotient()` en lui passant ces deux entiers. Votre programme devra gérer les éventuelles exceptions soulevées par la fonction `quotient()` : afficher un message `"Attention, division par 0"` si l'exception `ZeroDivisionError` est soulevée, ou un message affichant l'exception en question si une autre exception est soulevée. Si aucune exception n'est soulevée, votre programme affiche le résultat.
3. Nous avons vu qu'il existe plusieurs façons de demander à l'utilisateur de saisir une valeur au clavier. Nous avons notamment vu la fonction `raw_input()`, qui retourne toujours une chaîne de caractères, et la fonction `input()`, qui retourne une variable dont le type dépend de ce qui a été saisi. Modifiez votre programme pour demander à l'utilisateur de saisir une valeur avec la fonction `input()`, sans effectuer en conversion de type sur les variables saisies et passées en paramètre de votre fonction `quotient()`.
4. Dans ce cas, la fonction `quotient()` risque de recevoir des paramètres qui ne peuvent pas être utilisés pour une division. La première chose à faire dans cette fonction est donc de vérifier le type des arguments qui lui sont passés. La fonction `type()` retourne le type de l'argument qui lui est passé, qui est une constante définie dans le module `types`. On ne veut accepter par exemple que des entiers courts (de type `IntType`) ou des réels (de type `FloatType`). Modifiez votre fonction `quotient()` pour soulever une exception de type `TypeError` lorsque le dividende ou le diviseur n'est pas d'un type acceptable. La variable incriminée sera remontée par l'exception.
5. Modifiez votre programme afin d'attraper l'exception de type `TypeError` lorsqu'elle est soulevée, et d'afficher un message comme `"Attention, mauvais type d'un des paramètres : "` suivi du paramètre posant problème.

Exercice 4.2 : Recherche interrompue

Les exceptions ne signalement pas forcément une erreur ; elles peuvent également signaler l'arrêt d'un traitement. Par exemple, le parcours d'un grand tableau peut prendre beaucoup de temps. On peut souhaiter l'arrêter dès que possible, lorsque le traitement à effectuer est terminé.

1. Écrivez une fonction Python qui demande à l'utilisateur de saisir la taille du tableau et l'initialise en le remplissant de valeurs aléatoires réelles comprises entre 0 et 100, puis retourne ce tableau (vide en cas de problème).

   On demande à l'utilisateur de saisir une valeur avec la fonction `raw_input()`, qui retourne une chaîne de caractères. La conversion en entier peut soulever une exception. Vous traiterez cette exception pour vous prémunir du cas où l'utilisateur ne saisir pas une valeur pouvant être convertie en entier.

   Pour générer des valeurs pseudo-aléatoires, vous pourrez utiliser le module `random`. Attention à l'initialisation de votre générateur de nombres pseudo-aléatoires avec la fonction `random.seed()`.

2. Écrivez une classe d'exception `ValeurTrouvee` qui sera levée lorsqu'une valeur sera trouvée dans le tableau. Cette exception pourra être levée avec un ou deux arguments : la valeur trouvée seule, ou la valeur trouvée et l'indice auquel elle a été trouvée. Lorsqu'elle sera levée elle affichera une chaîne de caractères donnant ces informations.
3. Écrivez une fonction Python qui recherche dans un tableau la première valeur strictement supérieure à 50. Lorsque cette valeur a été trouvée, une exception de type `ValeurTrouvee` est soulevée en passant la valeur trouvée et l'indice auquel elle a été trouvée. Le fait de soulever cette exception interrompt le parcours du tableau et évite de le lire en entier. Si aucune valeur supérieure à 50 n'est trouvée, votre fonction retourne -1.
4. Écrivez un programme Python qui appelle la fonction initialisant le tableau et appelant votre fonction de recherche dans un tableau. Lorsque cette dernière fonction soulève une exception, votre programme doit l'attraper et afficher le résultat de sa recherche.

Exercice 4.3 : Parcours sûr d'un tableau

On veut écrire des fonctions qui permettent d'accéder en toute sûreté aux éléments d'un tableau. Notamment, on veut signaler une erreur lorsque l'utilisateur essaye d'accéder à un élément situé à un indice ne faisant pas partie du tableau : losque l'incide est négatif ou qu'il dépasse au-delà de la taille du tableau.

1. Définissez une classe d'exception `ErreurNegIndice` qui sera soulevée lorsque l'on essaye d'accéder à une case d'un tableau d'indice négatif. Elle affichera une chaîne de caractères disant que l'indice est négatif en précisant ou non l'indice en question.
2. Définissez une classe d'exception `ErreurIndiceTropGrand` qui sera soulevée lorsque l'on essaye d'accéder à une case d'un tableau située au-delà des limites du tableau, c'est-à-dire d'indice trop grand. Elle affichera une chaîne de caractères disant que l'indice est trop grand en précisant ou non l'indice en question.
3. Écrivez une fonction `lireValeurIndice` qui prend en paramètres un tableau et un indice, et retourne le contenu de ce tableau dans la case d'indice passé en paramètre. Cette fonction commence par vérifier que l'indice est bien de type entier (indice : vous utiliserez la fonction `type()` et le module `types`).
4. On veut que cette fonction soit sure, c'est-à-dire que si l'indice n'est pas dans les limites du tableau, elle soulève une exception au lieu d'essayer d'aller chercher la valeur demandée. Vous modifierez la fonction écrite à la question précédente afin de soulever les exceptions écrites plus tôt dans cet exercice.
5. Écrivez une fonction `main()` pour que votre programme tente de faire des accès erronés à des valeurs d'un tableau.

5 Manipulations de fichiers

Exercice 5.1 : Copie d'un fichier

Écrire un programme Python qui lit un fichier par blocs de 4 octets et recopie son contenu dans un autre fichier. À la fin de la copie, votre programme doit afficher le nombre d'octets copiés. Vous ferez attention à gérer les exceptions qui peuvent être soulevées dans vos appels aux fonctions de manipulation de fichiers.

Exercice 5.2 : Explosion de phrase

La fonction `split()` permet de séparer les éléments d'une chaîne de caractères en les mettant dans une liste. Vous trouverez l'utilisation de cette fonction dans la documentation en ligne de Python.

1. Écrire un programme Python qui lit un texte dans un fichier et écrit son contenu dans un autre fichier à raison d'un mot par ligne.
2. Les lignes du fichier `/etc/passwd` contiennent divers champs séparés par des caractères `:` et relatifs à un utilisateur pour chaque ligne. Écrire un programme Python qui lit le contenu de ce fichier et écrit dans un fichier le login (1er champ) et le nom réel (5eme champ) des utilisateurs, séparés par un espace.

Exercice 5.3 : Le répertoire

Dans le TP2, vous avez programmé un répertoire téléphonique. Cependant, vous avez remarqué que les informations contenues dans ce répertoire étaient perdues lorsque le programme est quitté. Le but de cet exercice est de rendre les données persistantes.

1. Les données du répertoire sont stockées en mémoire dans un dictionnaire. On veut les enregistrer dans un fichier à raison d'une personne par ligne, en écrivant le nom puis le numéro et en utilisant le signe `:` comme séparateur. Écrire une procédure qui prend en paramètre un dictionnaire et une chaîne de caractères et écrit son contenu dans le fichier dont le nom est la chaîne de caractères passée. Le contenu du fichier est écrasé à chaque écriture.
2. Écrire une fonction qui lit le contenu d'un fichier de données défini comme précédemment et dont le nom est passé en paramètre tous forme d'une chaîne de caractères et retourne son contenu sous forme d'un dictionnaire dont les clés sont les noms des personnes et les valeurs sont leurs numéros de téléphone. Attention à retirer le retour à la ligne du numéro de téléphone.
3. Modifier le programme de gestion du répertoire pour appeler les fonctions de lecture et d'écriture du répertoire. La fonction qui sauvegarde le répertoire doit être appelée à chaque modification du répertoire (ajout d'un nom, suppression d'une entrée, effacement du répertoire). Le répertoire doit être chargé depuis un fichier au démarrage du programme. Vous pourrez utiliser une variable globale pour stocker le nom du fichier.

6 Pour aller plus loin : algorithmes

Exercice 6.1 : Chaînes de caractères

Écrivez une procédure en python qui affiche un par un chaque caractère d'une chaîne de caractères.

Exercice 6.2 : Somme de contrôle

La somme de contrôle d'un numéro de carte bleue s'effectue en additionnant tous les chiffres qui le composent. Si le résultat est divisible par 10, alors le numéro est correct.

1. Écrivez une fonction qui prend en argument un numéro de carte bleue sous forme d'une chaine de caractères et retourne True ou False selon si le numéro est correct ou non.
2. Écrivez une fonction qui prend en argument un numéro de carte, et calcule le dernier chiffre nécessaire à le rendre correct.

Exercice 6.3 : Carrés magiques

Un carré magique est un tableau à deux dimensions comportant le même nombre de lignes et de colonnes et dont les sommes des éléments de toutes les lignes, de toutes les colonnes et de toutes les diagonales sont égales.

Par exemple, le carré suivant est magique :

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

Écrivez un programme Python qui affiche si un carré est magique ou non.

Exercice 6.4 : Binôme de Newton et mémoïsation

Le développement d'une expression de type $(x+y)^n$ se calcule avec la formule $\sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$, en calculant les coefficients binomiaux $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

Par exemple, pour $n = 3$:

— $C_3^0 = \frac{3!}{0!(3-0)!} = \frac{3!}{3!} = 1$

— $C_3^1 = \frac{3!}{1!(3-1)!} = \frac{3!}{2!} = 3$

— $C_3^2 = \frac{3!}{2!(3-2)!} = \frac{3!}{2!} = 3$

— $C_3^3 = \frac{3!}{3!(3-3)!} = \frac{3!}{3!} = 1$

On arrive donc à $(x+y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$.

On peut définir la valeur des coefficients binomiaux de façon récursive avec la formule suivante : $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.

1. Écrivez une fonction récursive qui prend en paramètre deux entiers k et n et retourne le calcul du coefficient binomial correspondant $\binom{n}{k}$, calculé de façon récursive.
2. Écrivez un programme qui demande à l'utilisateur de saisir la valeur de n et affiche le polynôme sous forme développée.

3. Vous constaterez que le résultat d'un calcul peut être réutilisé ultérieurement dans le calcul d'un autre coefficient binomial. Pour éviter de refaire ce calcul, on peut garder en mémoire son résultat : c'est ce que l'on appelle la mémoïsation.

   Utilisez une variable globale de type dictionnaire dont les clés sont des tuples `(k,n)` pour stocker les résultats des calculs. Au début de la fonction de calcul du coefficient binomial, regardez si le résultat se trouve dans ce dictionnaire. Si il s'y trouve déjà, retournez directement le résultat. Si il ne s'y trouve pas, calculez-le et enregistrez le résultat dans le dictionnaire.
4. Comparez les temps de calcul avec et sans mémoïsation pour des valeurs de n supérieures à la dizaine.

Exercice 6.5 : Un tri

Dans cet exercice, vous allez implémenter un algorithme qui trie les valeurs contenues dans une liste. Pour cela, vous allez construire une nouvelle liste en y insérant les valeurs de la liste originale de façon triée.

— Initialisez la nouvelle liste à `[]`.
— Pour chaque élément de l'ancienne liste, parcourez la nouvelle liste et insérez l'élément à un endroit où l'élément précédent est plus petit et l'élément suivant est plus grand.

Par exemple, si vous avez la liste `[6, 9, 7, 2, 3, 4, 0, 5, 1, 8]` :

— La nouvelle liste est initialisée à `[]`.
— Élément 6 : on l'insère dans la nouvelle liste, qui vaut maintenant `[6]`.
— Élément 9 : on parcourt la nouvelle liste, 9 est plus grand que 6 et il n'y a plus rien ensuite : on l'insère après 6, la nouvelle liste est maintenant `[6, 9]`.
— Élément 7 : on parcourt la nouvelle liste, 7 est plus grand que 6 et plus petit que 9 : on l'insère après 6, la nouvelle liste est maintenant `[6, 7, 9]`.
— et ainsi de suite...

1. Écrivez une fonction qui prend en argument une liste non triée et retourne une nouvelle liste contenant les mêmes valeurs mais triées.
2. Écrivez un programme qui demande à l'utilisateur de saisir des valeurs non triées et les affiche triées.

7 Pour aller plus loin : des dessins

Exercice 7.1 : Représentation graphique d'une séquence d'ADN

L'ADN est composé de quatre molécules de bases ou nucléotides : adénine, cytosine, guanine, thymine, souvent désignées par leur première lettre A C G T. On peut représenter une séquence d'ADN sous forme graphique en allant dans une direction différente selon le nucléotide rencontré :

— Vers le haut pour l'adénine
— Vers le bas pour la thymine
— Vers la gauche pour la guanine
— Vers la droite pour la cytosine

1. Écrivez une procédure en Python qui prend en paramètre une séquence de nucléotides sous la forme d'une chaîne de caractères de type `ACGTTGCTGA` et de longueur quelconque, et l'affiche de la façon décrite ci-dessus en utilisant la tortue.
2. Écrivez un programme Python qui lit cette chaîne de caractères dans un fichier et l'affiche sous forme graphique en appelant la fonction que vous venez d'écrire.

8 Pour aller plus loin : programmation modulaire

Exercice 8.1 : La tortue en Français

Afin de "protéger la langue française", on souhaite programmer une interface de la Tortue graphique utilisée lors du TP précédent permettant d'appeler ses fonctions avec un nom en français.

On considèrera pour le moment les fonctions suivantes :

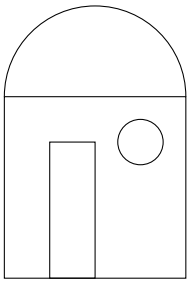
— `forward( distance )`
— `left( angle )`
— `right( angle )`
— `up()`
— `down()`
— `goto( w, y )`
— `circle( rayon, angle, etapes )`
— `color( couleur )`

Qui deviendront respectivement :

— `avant( distance )`
— `gauche( angle )`
— `droite( angle )`
— `lever()`
— `baisser()`
— `aller( w, y )`
— `cercle( rayon, angle, etapes )`
— `couleur( couleur )`

1. Écrivez un module python `turtle_fr.py` qui fournit les fonctions nommées ci-avant en français, qui font appel aux fonctions correspondantes du module `turtle`. Votre module doit également fournir une constante corresponsant à son numéro de version.
2. Afin de tester si les fonctions de votre module fonctionnent correctement, écrivez une fonction `main()` qui ne sera appelée que si le module est exécuté comme un script, mais ne sera pas appelée si votre module est importé par un autre programme Python. Votre fonction `main()` fera quelques appels aux fonctions que vous avez écrites afin de vérifier qu'elles fonctionnent correctement.
3. On veut à présent écrire un module fournissant des fonctions qui tracent des figures géométriques. On peut prendre par exemple les fonctions suivantes :
   — `polygone( x, nb )` : trace un polygone régulier à `nb` côtés de longueur `x`
   — `carre( x )` : trace un carré de côté `x`
   — `triangle( x )` : trace un triangle équilatéral de côté `x`
   — `rectangle( grand, petit )` : trace un rectangle de petit côté `petit` et de grand côté `grand`
   — `demicercle( x )` : trace un demi-cercle de rayon `x`

   Écrivez un module `figuresgeom.py` qui implémente ces fonctions en faisant appel aux fonctions que vous avez écrites dans le module `turtle_fr.py`. Ce module fournira également une constante donnant son numéro de version.
4. Écrivez une fonction `main()` appelée uniquement lorsque le module est exécuté en tant que script, qui effectue quelques appels aux fonctions implémentées dans ce module.
5. Écrivez un programme qui utilise le module `figuresgeom` et le module `turtle_fr` pour tracer le dessin suivant. Votre programme ne devra utiliser aucune fontion du module `turtle` directement, mais uniquement des fonctions des deux modules que vous venez d'écrire.



Exercice 8.2 : Un module de statistiques

Les fonctions statistiques sur des données sont très souvent utilisées. C'est pourquoi il est très intéressant d'écrire un module implémentant les fonctions dont on a besoin, et d'appeler ce module à chaque fois que l'on a besoin de telles fonctions dans un programme.

Dans cet exercice, les données sont fournies sous forme d'une liste.

1. Créez un module `mes_stats.py` qui contiendra les fonctions statistiques que vous allez implémenter, ainsi qu'une constante correspondant à son numéro de version.
2. Écrivez une fonction `main()` appelée uniquement lorsque le module est exécuté en tant que script, qui effectuera quelques appels aux fonctions implémentées dans ce module. Pour le moment, votre fonction initialise une liste de 10 variables en la remplissant avec des valeurs entières pseudo-aléatoires entre 0 et 100 (0 est inclus, 100 est exclus). Pour générer des valeurs pseudo-aléatoires, vous pourrez utiliser le module `random` en faisant attention à l'initialisation de votre générateur (voir la fonction `random.seed()`).
3. Implémentez dans ce module la fonction `moyenne()`, qui calcule la moyenne d'une liste et la retourne (attention, on veut un réel et on calcule sur des entiers).
4. Implémentez dans ce module les fonctions `mini()` et `maxi()`, qui retournent respectivement le minimum et le maximum de la liste de valeurs.
5. Une fonction statistique utile consiste à calculer la moyenne non pas sur toutes les valeurs fournies, mais sur les valeurs après retrait des deux valeurs les plus grandes et des deux valeurs les plus petites. Implémentez dans votre module la fonction `listeSansExtremes()` qui retourne une liste privée des quatre valeurs extrêmes évoquées ci-avant, et la fonction `moyenneSansExtremes()` qui retourne la moyenne d'une liste sans ces quatre valeurs extrêmes. Cette fonction devra utiliser les fonctions écrites précédemment.
6. L'écart-type correspond à la racine carrée de la variance. La variance est la moyenne de l'écart entre les valeurs et la moyenne. Il existe beaucoup de façons plus rapides pour la calculer, mais vous pourrez la calculer en calculant dans un premier temps la moyenne des valeurs, puis en calculant la moyenne des carrés de l'écart entre les valeurs et la moyenne. Implémentez dans votre module les fonctions `variance()` et `ecartType()`.
7. Modifiez votre fonction `main()` pour faire quelques appels aux fonctions de votre module de statistiques.
8. Vous avez maintenant une boîte à outils de statistiques, que vous allez pouvoir utiliser sur des données. Écrivez un programme Python qui utilise les fonctions de votre module afin de calculer des statistiques sur un texte. Ce texte sera fourni à votre programme sous forme d'une variable contenant une chaîne de caractères entrée en dur dans le programme ou saisie par l'utilisateur. Vous devrez dans un premier temps transformer votre texte en une liste de mots puis obtenir la liste contenant longueurs des mots. Les statistiques à afficher sont le nombre de mots dans le texte, le nombre moyen de lettres par mot, la longueur du mot le plus long et du mot le plus court, l'écart-type des nombres de lettres par mot et la moyenne et l'écart-type des les longueurs des mots en ayant retiré les deux mots les plus longs et les deux mots les plus courts.