TP 1 - Premiers pas -

À la fin de cette séance, vous devrez rendre le TP. En particulier, vous devez récupérer le fichier `TP1rendu.txt` sur la page du cours et le compléter au fur et à mesure.

Avant de commencer le TP :

1. Ouvrez un terminal et dans votre répertoire personel, créez un dossier `Python` :
   ```
   mkdir Python
   ```
2. Placez-vous dans le dossier `Python` et créez un dossier `TP1` :
   ```
   cd Python
   mkdir TP1
   ```
3. Placez-vous dans le dossier `TP1` :
   ```
   cd TP1
   ```

Exercice 1. Il était une fois le *top level*...

Le but de cet exercice est de vous familiariser avec l'interpréteur Python.

1. Lancez le top level Python (tapez la commande suivante dans le terminal) :
   ```
   python3
   ```
   L'interpréteur Python (aussi appelé *shell Python*) apparaît : il est matérialisé par des petits chevrons : `>>>` que l'on appelle *invite commande* (ou *prompt*). Cela signifie que vous allez pouvoir discuter avec votre Python...
2. Tapez les commandes suivantes. Expliquez les résultats que vous obtenez. Essayez d'anticiper les réponses, et lorsque cela ne correspond pas, expliquez pourquoi.

```
1 >>> 20 + 1
2 >>> 20 / 3
3 >>> 20 // 3
4 >>> 20 % 3
5 >>> 5.45 * 10
6 >>> 2 ** 4
7 >>> (3+2) * 5
8 >>> 3+2 * 5
```

Exercice 2. Chaînes de caractères

1. Au fait, vous n'avez pas été très poli avec Python. Essayez de lui dire bonjour. Que se passe-t-il ? En quelle langue vous répond-il ?
2. Ré-essayez en utilisant des apostrophes.
   Remarque : au top level, vous pouvez utilisez l'*historique de commandes* (flèches haut et bas) qui vous permet de ré-afficher des commandes que vous avez déjà tapées.

Cet énoncé de TP s'inspire très largement d'un cours de programmation en Python donné à l'IUT d'Orsay.

3. Que se passe-t-il si on additionne un chat et de l'eau (la chaîne de caractères chat et la chaîne eau) ? Essayez.
4. Que se passe-t-il si on multiplie (la chaîne) bonjour par 3 ? Essayez.
5. Que se passe-t-il si on ajoute 3 à (la chaîne) bonjour ? Essayez.

Exercice 3. Erreurs
Tapez les commandes suivantes et expliquez ce que vous obtenez (ne vous contentez pas de traduire, expliquez ce qui ne va pas).

```
>>> 20 / 0
>>> 20 @ 3
>>> 'bonjour' / 3
>>> 'bonjour' + 5
>>> (3+2)) * 5
>>> (3+2 * 5
```

Exercice 4. Types
Les expressions en Python (comme `20`, `5.45`, `'bonjour'`, ... que nous venons d'utiliser) ont toutes un type.

1. Déterminez le type des expressions utilisées dans les exercices précédents et vérifiez en utilisant la *fonction* `type()` qui prend une expression en paramètre et renvoie son type.
2. Sur quels types de données peut-on utiliser les opérateurs `+`, `*`, `/`, `//`, `%`, `**` ? Quel est le type du résultat ? Vous pouvez faire des tests en tapant d'autres instructions au top-level.
3. Déterminez également le type et le résultat des expressions suivantes :

```
'3' + '4'
3 + '4'
'3' + 4
'3' * '4'
3 * '4'
'3' * 4
'3' * 4.0
```

4. En Python, vous pouvez "forcer" le type d'une expression à être autre chose que son type (on appelle cela le *transtypage* ou *cast* en anglais). Pour cela, on utilise les fonctions `int()`, `float()` et `str()` pour transformer respectivement en entier, flottant ou chaîne de caractères.
   (a) Essayez de transformer les expressions `3.3`, `2.7`, `'2'`, `'2.3'`, `'a'` en `int`. Que se passe-t-il ? Commentez.
   (b) Essayez de transformer les expressions `3`, `3/2`, `'2'`, `'2.3'` en `float`. Que se passe-t-il ? Commentez.
   (c) Essayez de transformer les expressions `3`, `3/2`, `3.2` en `str`. Que se passe-t-il ? Commentez.

Exercice 5. Variables
Qu'affichent les commandes suivantes quand elles sont exécutées les unes à la suite des autres ? Vous devez essayer de deviner avant de tester !

```
>>> foo
>>> foo = 2.1
>>> foo
>>> type(foo)
>>> foo = 2
>>> type(foo)
>>> bar = foo
>>> foo = 3
>>> bar
>>> foo = foo * bar
```

Exercice 6. Premiers pas avec `idle` : nommage des variables

Nous allons maintenant utiliser un éditeur de texte pour lire et écrire des programmes en Python. Celui que vous utiliserez en TP s'appelle `idle` et est généralement installé en même temps que Python (ou python-tk).

Récupérez le fichier `charabia.py` sur la page du cours et lancez la commande (sans oublier le `&`) :

```
idle-python3.1 charabia.py &
```

Attention, un shell Python va se lancer en même temps : nous déconseillons de l'utiliser en tant que top-level, car il est moins pratique que celui que vous obtenez en lançant `python3` dans un terminal.

1. Avez-vous remarqué les couleurs ? À quoi correspondent-elles ?
2. Que calcule ce programme (`a` et `b` sont les données) ?
3. Vous pouvez exécuter votre programme en lançant la commande :

   ```
   python3 charabia.py
   ```

   ou en tapant sur la touche "F5".
4. Maintenant que vous savez ce que fait ce programme, pour le rendre lisible, choisissez des noms de variables plus "explicites". Voici quelques rappels sur les conventions de nommage des variables :
   - le nom contient uniquement :
     - des lettres sans accents,
     - des chiffres,
   - il débute par une lettre minuscule (les noms commençant par des majuscules servent pour autre chose)
   - si l'on veut que le nom de variable contienne plusieurs mots, on colle les mots en mettant une majuscule à tous les mots sauf le premier comme par exemple dans `hauteurCylindre`.
   - il ne doit pas être un mot réservé du langage comme `if`, `True`, `import`,... (ils sont faciles à reconnaître grâce aux couleurs).
5. Utilisez la fonction `input()` pour que l'utilisateur choisisse les valeurs de `a` et `b` et faites en sorte que l'affichage final soit plus clair.
6. Enfin, rajoutez des commentaires dans votre programme pour décrire ce qu'il fait.

À partir de maintenant, tous vos programmes devront être commentés et utiliser des noms de variables explicites afin d'être facilement compréhensibles.

Exercice 7. Tortue : affectation de variables

Récupérez le fichier `tortue.py` sur la page du cours. Ce programme contient 2 parties :

- la première qui contient des variables et que vous pourrez modifier ;
- et la seconde à laquelle vous ne devez pas toucher. Le code de la deuxième partie utilise la *tortue* qui permet de :
  - ouvrir une fenêtre de taille `600 x 600` pixels,
  - poser le stylo au point de coordonnées `(xInit,yInit)`,
  - tracer `nbLignes` de longueur `lgLigne` pixels en tournant de `angle` degrés vers la gauche à chaque fois.

1. Lancez le programme en tapant l'instruction suivante dans le terminal :
   ```
   python3 tortue.py
   ```
2. Que voyez-vous ? Avez-vous remarqué la petite flèche symbolisant la tortue ?
3. Modifiez les variables afin de centrer le carré.
4. Modifiez les variables pour que la longueur du carré soit `300` pixels.
5. Votre carré est-il toujours centré ?
6. Modifier les variables pour que le carré soit centré quelle que soit la longueur de ses côtés (en restant inférieure à 600). Vous pouvez utiliser la fonction `input()` pour que l'utilisateur choisisse la longueur.
7. Modifiez les variables pour que le programme dessine un triangle équilatéral.
8. Modifiez les variables pour que le programme dessine un pentagone régulier.
9. Modifiez les variables pour que le programme dessine un polygone régulier quel que soit le nombre de lignes choisi (utilisez la fonction `input()`).
10. Que se passe-t-il si on demande un polygone à 50 côtés ? Corrigez les variables si nécessaire.
11. ★ Votre dessin est-il toujours centré ?

    Pour centrer votre polygone, vous aurez besoin des fonctions trigonométriques comme cosinus, sinus et tangente. Pour y avoir accès en Python, il faut rajouter au tout début de votre programme la ligne :
    ```
    from math import *
    ```
    Vous pourrez ensuite avoir accès à la constante `pi` et aux fonctions `cos`, `sin` et `tan` qui attendent un angle donné en radian (et non pas en degrés).

```
>>> from math import *
>>> pi
3.141592653589793
>>> cos(pi)
-1.0
```

TP 2 - Branchements conditionnels -

Le but de ce TP est de vous familiariser avec les tests ainsi que les structures de branchements conditionnels.

Exercice 1. Ecrivez un programme qui affiche "PAIR" si un nombre rentré par l'utilisateur est pair et "IMPAIR" sinon. Trouvez 2 façons différentes d'écrire un programme qui fait la même chose.

Exercice 2. Divination

Vous allez devoir écrire un programme `jeu.py` proposant à l'utilisateur de deviner un entier entre 1 et 10 en trois essais.

a. Commencez par choisir un nombre au hasard. Pour cela, vous pouvez utiliser la fonction `randint` :

```
1 from random import randint
2 print( randint(0,2) )    # affiche un entier aleatoire dans [0,2].
```

b. Puis demandez un entier entre 1 et 10 à l'utilisateur et comparez-le à l'entier aléatoire choisi.

c. Faites en sorte que l'utilisateur aie 3 essais et que le programme s'arrête quand il a trouvé, ou quand son nombre d'essais est écoulé, en affichant un message approprié.

d. Testez votre programme avec diverses valeurs.

e. Que se passe-t-il si l'utilisateur rentre un flottant ?

f. Que se passe-t-il si l'utilisateur se trompe et rentre un nombre qui n'est pas entre 1 et 10 ? Modifiez votre programme pour qu'il redonne sa chance à l'utilisateur dans ce cas-là.

g. ★ Et si l'utilisateur se trompe encore ?

Exercice 3. Max, min et leurs amis...

La *fonction* `max()` est une des fonctions intégrées de Python. Elle prend 2 paramètres et *renvoie* le plus grand des 2. Testez les commandes suivantes au top-level.

```
1 >>> max(1,2)
2 >>> max(-1,2)
3 >>> a = max(1,2.2)
4 >>> print(a)
5 >>> print( max(3,4) )
6 >>> max(3+2,4)
```

1. Ouvrez le fichier `charabia2.py` (avec `gedit` ou `idle-python3.1`).
2. Rajoutez des commentaires après chaque test pour permettre de comprendre le programme.
3. Que fait ce programme ?
4. Ouvrez le fichier `charabia3.py`.

5. Changez les noms de variables pour permettre de comprendre le programme.
6. Que fait ce programme ?

Exercice 4. Divisibilité
Dans tout l'exercice, les seuls tests booléens que vous pouvez faire sont des tests de divisibilité par 2, 3 ou 4. Pour chaque question, notez en commentaires les valeurs sur lesquelles vous testez votre programme.

1. Demandez à votre utilisateur de choisir un nombre entier positif.
2. Si le nombre est divisible par 2, 3 ou 4, faites afficher "Divisible par 2, 3 ou 4" (une seule fois, même si le nombre est 24), sans utiliser d'opérateur booléen.
3. Même question, mais en utilisant un seul `if`.
4. Si le nombre est divisible par 2, 3 ou 4, faites afficher "Divisible par 2" ou "Divisible par 3" ou "Divisible par 4" (pour 24, ça fait 3 affichages).
5. Complétez la question précédente pour que le programme affiche "Pas divisible par 2, 3 ou 4" si le nombre n'est divisible par aucun des trois.
6. Afficher "Divisible par 12" si le nombre est divisible par 12.
7. Afficher si le nombre est divisible par 3 ou 4 mais pas par 12.
8. ★ Refaire la question **??** en n'utilisant qu'un seul test de divisibilité pour chaque valeur (2,3,4).

Exercice 5. Récupérer les fichiers `iutk.py` et `formes.py` sur la page du cours. Les deux fichiers doivent être dans le même répertoire. Vous ne travaillerez que sur le fichier `formes.py`. L'autre fichier fournit une bibliothèque graphique que nous utiliserons pendant ce cours et qui est appelée au début de `formes.py` à l'aide de la ligne :

```
1 from iutk import *
```

La première partie du programme `formes.py` (que vous n'avez ni à modifier, ni à comprendre) réalise les tâches suivantes :
- attendre un clic de l'utilisateur ;
- afficher un rectangle bleu de dimensions 100 × 50 (largeur × hauteur) et dont le coin supérieur gauche est donné par le clic ;
- attendre un second clic de l'utilisateur ;
- afficher un rectangle rouge de dimensions 75 × 25 et dont le coin supérieur gauche est donné par le second clic.

Les coordonnées (abscisse, ordonnée) du premier point cliqué sont stockées dans les variables `x1` et `y1` respectivement. Les coordonnées (abscisse, ordonnée) du second point cliqué sont stockées dans les variables `x2` et `y2` respectivement.

1. Complétez la fin du programme de manière à afficher dans le terminal :
   - `Pas d'intersection` si les deux rectangles ne se touchent pas,
   - `Rouge contenu dans Bleu` si le rectangle rouge est contenu dans le rectangle bleu,
   - et `Intersection` si les deux rectangles s'intersectent sans que le rectangle rouge ne soit inclus dans le rectangle bleu.
2. Combien de tests devez-vous faire pour être sûr que tout fonctionne ?
3. Modifiez le programme pour qu'il demande les dimensions des rectangles à l'utilisateur.

Exercice 6. Le jour de la semaine

La *formule de Zeller* permet de déterminer le jour de la semaine correspondant à une date donnée. On l'obtient grâce à l'expression suivante, où la notation $\lfloor x \rfloor$ désigne le *plancher* de x, c'est-à-dire le plus grand entier inférieur ou égal à x (par exemple : $\lfloor 15,8 \rfloor = 15$) :

$$\text{jour} = \left(j + \left\lfloor \frac{26(m+1)}{10} \right\rfloor + a + \left\lfloor \frac{a}{4} \right\rfloor + \left\lfloor \frac{s}{4} \right\rfloor + 5s \right) \%7,$$

où

- jour est le numéro du jour (0 = samedi, 1 = dimanche, ..., 6 = vendredi) ;
- j est le jour du mois (entre 1 et 31) ;
- m est le numéro du mois (3 = mars, 4 = avril, ..., 12 = décembre, 13 = **janvier**, 14 = **février**) ;
- s est le siècle (par exemple 19 si l'année est 1987) ;
- a est l'année dans le siècle (par exemple 87 si l'année est 1987).

1. Ecrivez un programme saisissant une date rentrée par l'utilisateur et lui donnant le jour de la semaine correspondant à cette date. Attention, il faut aussi gérer le cas des dates n'existant pas (exemple : 31 février ou 32 juin) !
2. Il est plutôt inhabituel et peu confortable pour l'utilisateur de considérer que janvier et février sont les $13^{\text{ème}}$ et $14^{\text{ème}}$ mois. Modifiez votre programme pour que l'utilisateur puisse entrer 1 et 2 à la place de 13 et 14, respectivement.

TP 3 - Boucles

Récupérez sur la page du cours le fichier `iutk.py` (celui de ce TP) qui fournit la bibliothèque graphique que nous allons utiliser et placez-le dans votre répertoire de travail. Vous devez écrire un programme par exercice, avec comme nom **exoX.py** (X est le numéro de l'exo) et mettre en commentaires les tests que vous avez faits pour vérifier que votre programme fonctionne.

Exercice 1. Trois mais pas cinq

Ecrire un programme qui prend un entier positif n en entrée, et affiche tous les nombres de 1 à n qui sont divisibles par 3 mais pas divisibles par 5.

Exercice 2. De plus en plus grand

Ecrire un programme qui laisse l'utilisateur rentrer autant de nombres positifs qu'il veut, tant que chaque nombre est strictement supérieur à celui d'avant. Si un nombre rentré ne vérifie pas la règle, le programme s'arrête et indique combien de nombres comporte la suite strictement croissante rentrée par l'utilisateur.

Exercice 3. Nombres premiers

On rappelle qu'un entier $n \geq 2$ est premier quand il n'est divisible que par 1 et lui-même : 7 est premier, mais $6 = 2 \times 3$ ne l'est pas.

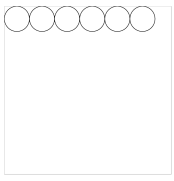
1. Faites un programme qui demande un nombre à l'utilisateur et indique si le nombre est premier ou non.
2. Faites un programme qui demande un nombre n à l'utilisateur et indique *tous* les nombres premiers qui sont inférieurs ou égaux à n.

Exercice 4. Moyenne

1. Ecrire un programme qui permet à l'utilisateur de saisir autant de nombres entiers qu'il veut et qui calcule la moyenne des nombres rentrés. Quand l'utilisateur a terminé, il écrit `stop` au lieu d'un nombre.
2. Modifier le programme pour qu'il redemande les nombres tant qu'ils ne sont pas compris entre 0 et 20.

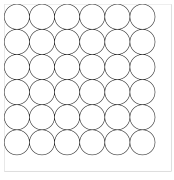
Exercice 5. Récupérez le programme `dessin-init.py` sur la page du cours.

1. Modifiez le programme pour avoir l'affichage suivant :



   Les cercles ont un rayon de 30 pixels et les centres de deux cercles consécutifs sont espacés de 60 pixels. Les cercles s'arrêtent juste avant de sortir de la fenêtre.

   Faites en sorte que votre programme fonctionne si l'on change la valeur de `largeurFenetre` et `hauteurFenetre`.
2. Modifiez votre programme pour obtenir l'affichage suivant :



Exercice 6. Terminer les exercices sur les rectangles et le jour de semaine du TP précédent.

TP 4 - Boucles

Récupérez le fichier `iutk.py`. Vous devez écrire un programme par exercice, avec comme nom **exoX.py** (X est le numéro de l'exo) et mettre en commentaires les tests que vous avez faits pour vérifier que votre programme fonctionne.

Exercice 1. Premières listes

1. Déclarer la liste `lst` suivante : `[14,7,6,12,2,3,3,10]` et l'afficher.
2. Ensuite, modifier la liste de telle façon que son dernier élément soit divisé par 2 et la ré-afficher.
3. Modifier la liste de telle façon que l'on retranche 1 à tous ses éléments et la ré-afficher.
4. Ensuite afficher tous les éléments de la liste sur des lignes différentes.
5. Puis faire afficher seulement les nombres pairs (sans fabriquer de nouvelle liste).
6. Ensuite, faire afficher 10 fois chaque élément de la liste sur la même ligne séparés par des espaces. Attention, la fonction `print('bonjour')` va à la ligne après avoir affiché `bonjour` ; pour éviter ce comportement, il faut utiliser `print('bonjour', end=' ')`.
7. Enfin, afficher chaque élément autant de fois que sa valeur. Par exemple, 14 sera affiché 14 fois, ...

A la fin de l'exercice, votre programme devra afficher exactement :

```
Question 1
[14, 7, 6, 12, 2, 3, 3, 10]
Question 2
[14, 7, 6, 12, 2, 3, 3, 5]
Question 3
[13, 6, 5, 11, 1, 2, 2, 4]
Question 4
13
6
5
11
1
2
2
4
Question 5
6
2
2
4
Question 6
13  13  13  13  13  13  13  13  13  13
6  6  6  6  6  6  6  6  6  6
5  5  5  5  5  5  5  5  5  5
11  11  11  11  11  11  11  11  11  11
1  1  1  1  1  1  1  1  1  1
```

```
2  2  2  2  2  2  2  2  2  2
2  2  2  2  2  2  2  2  2  2
4  4  4  4  4  4  4  4  4  4
Question 7
13  13  13  13  13  13  13  13  13  13  13  13  13
6  6  6  6  6  6
5  5  5  5  5
11  11  11  11  11  11  11  11  11  11  11
1
2  2
2  2
4  4  4  4
```

Exercice 2. Entrées

1. Voici un programme qui demande des entiers à l'utilisateur jusqu'à ce qu'il rentre `stop`.

```
1 while True:
2     valOuStop=input("valeur ou stop ")
3     if valOuStop=="stop":
4         break
5     a=int(valOuStop)
```

   Modifier le programme pour qu'il enregistre les entiers dans une liste `lst`.

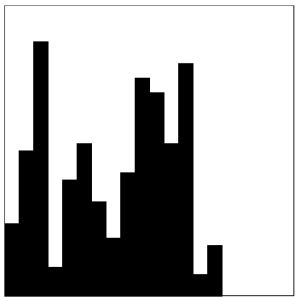
2. A la suite de la boucle qui crée la liste `lst`, afficher les nombres dans l'ordre où ils ont été rentrés sur la même ligne séparés par des espaces.
3. Ensuite construire une liste `lst2` contenant seulement les nombres pairs de `lst` et l'afficher avec la fonction `print`.
4. Ensuite construire une liste `lst3` contenant les entiers `lst` dans l'ordre inverse de leur saisie et l'afficher avec la fonction `print`.

Exercice 3. Dessiner une ville

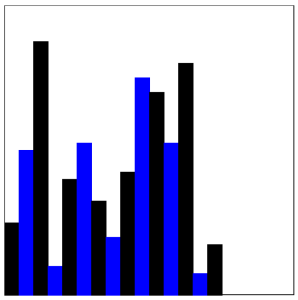
Pour cet exercice, vous pouvez ré-utiliser le fichier `dessin-init.py` fourni au TP précédent. Le but de l'exercice est de réaliser un programme dessinant une ville. Pour voir le résultat que l'on cherche à obtenir, récupérer le fichier `dessin-ville.pyc`. Vous pouvez le lancer en tapant :

```
python3 dessin-ville.pyc
```

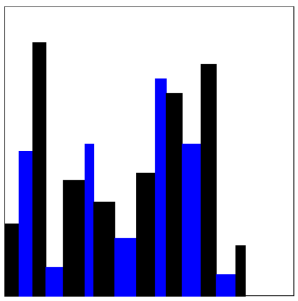
1. Déclarer la liste `lst` des hauteurs des immeubles dans la ville. Dans l'exemple : `lst=[100,200,350,40,160,210,130,80,170,300,280,210,320,30,70]`
2. Faire afficher une suite de rectangles de largeur fixe de 20 et dont les hauteurs sont données par la liste `lst`.



3. Modifier le programme pour que les rectangles soient alternativement coloriés en bleu et noir (pour ce faire, vous utiliserez une variable `couleur` qui vaudra soit “blue” soit “black”).



4. On souhaite maintenant pouvoir faire des rectangles de largeur variable. Pour cela créer la liste :
   `lstLargeurs=[20, 19, 18, 24, 30, 12, 29, 30, 26, 15, 22, 26, 21, 27, 13]`
   Modifier votre programme pour qu'il affiche les rectangles dont les hauteurs sont données par la liste `lst` et les largeurs par `lstLargeurs`. Par exemple, le premier immeuble est de hauteur 100 et de largeur 20.



5. Faites en sorte que la liste `lstLargeurs` soit créée en choisissant des valeurs aléatoires (entre 10 et 30, en utilisant `randint`. Il vous faudra rajouter `from random import randint` au début de votre programme). Afficher la liste `lstLargeurs` pour vérifier.
6. Faire en sorte que la largeur de la fenêtre corresponde exactement à la largeur de tous les immeubles.
7. (facultatif) Modifier votre programme pour qu'il affiche chaque rectangle avec un petit délai, en utilisant les fonction `sleep` et `miseAJour`.

Exercice 4. Facultatif

Récupérez le programme `balle-init.py` sur la page du cours. Ce programme simule le lancer d'une balle dont le vecteur de vitesse initial est donné par le premier clic dans la fenêtre. Le but de l'exercice est de réaliser un petit jeu dans lequel on a 3 tentatives pour toucher une cible.

Pour voir ce que l'on attend de vous, récupérer le fichier `balle.pyc`. Vous pouvez lancer le programme, en tapant :

```
python3 balle.pyc
```

1. Ouvrez et lisez le fichier `balle-init.py` afin de comprendre son fonctionnement. N'hésitez pas à ajouter des commentaires.
2. Utilisez la fonction `randint` vue au TP précédent pour placer une cible qui sera un carré rouge de coté 20 dans le quart inférieur droit de la fenêtre.
3. Ajoutez les tests permettant de savoir si la balle sort de la fenêtre, auquel cas on sort de la boucle et l'on affiche *Perdu* avec le code suivant :

```
1    texteCentre(largeurFenetre/2,hauteurFenetre/2,"Perdu","red")
```

4. Ajoutez les tests permettant de savoir si la balle touche la cible, auquel cas on sort de la boucle et l'on affiche *Gagné* avec le code suivant :

```
1    texteCentre(largeurFenetre/2,hauteurFenetre/2,"Gagne","red")
```

5. Modifiez votre code de manière à donner 3 essais au joueur.

TP 5 - Listes et fonctions

Récupérez sur la page du cours le fichier `iutk.py` (celui de ce TP) qui fournit la bibliothèque graphique que nous allons utiliser et placez-le dans votre répertoire de travail.

Attention : chaque fonction que vous écrivez doit être précédée d'un commentaire qui indique ce qu'elle fait en fonction des paramètres donnés !

Exercice 1. Récupérez le programme `liste100000.py` sur la page du cours. Ce programme construit la liste des nombres entre 0 et 99 999 de deux manières différentes et affiche le temps d'exécution pour chacune des méthodes. Vous pouvez consulter la documentation du module `time` ici :`http://docs.python.org/3.2/library/time.html`

Expliquez l'énorme différence de temps entre les deux méthodes.

Exercice 2. Salutations

Vous devez tester les 3 **fonctions** suivantes dans le programme que vous rendez (`Exo2.py`).

1. Écrire une fonction `bonjour` sans argument qui **affiche** le texte *bonjour*.
2. Écrire une fonction `disBonjour` qui prend un entier `n` et **affiche** `n` fois le texte *bonjour*. Par exemple, `disBonjour(3)` affichera :

   ```
   bonjour
   bonjour
   bonjour
   ```

3. Écrire une fonction `renvoieBonjour` qui prend un entier `n` et **renvoie** une chaîne composée de `n` fois le texte *bonjour* séparés par des espaces. Attention, cette fonction n'affiche rien !

Exercice 3. Carrés

1. Écrire une fonction `carre` qui prend un entier en argument et renvoie son carré.
   Par exemple, `carre(5)` renverra `25`.
2. Écrire un programme utilisant la fonction `carre` qui demande un nombre `n` à l'utilisateur et affiche les `n` premiers carrés.
   Par exemple, si l'utilisateur entre 5, le programme affichera :

   ```
   1
   4
   9
   16
   25
   ```

Exercice 4. Nombre à la demande

Pour chaque question, vous devez utiliser les fonctions des questions précédentes.

1. Écrire (**et tester**) une fonction `demandeEntier` qui ne prend pas d'argument et qui demande à l'utilisateur de rentrer un nombre et le renvoie. Attention, cette fonction doit renvoyer un `int` et pas une chaîne !
2. Écrire (et tester) une fonction `estEntre` qui prend trois arguments entiers `val`, `a` et `b` et renvoie `True` si `val` est compris entre `a` et `b` et `False` sinon.

3. Écrire (et tester) une fonction `demandeEntre` qui prend deux arguments entiers `a` et `b` et redemande à l'utilisateur de rentrer un nombre jusqu'à ce que le nombre entré soit compris entre `a` et `b`.

   Par exemple, `demandeEntre(1,5)` redemande à l'utilisateur de rentrer un nombre jusqu'à ce que ce nombre soit compris entre 1 et 5.

4. Écrire un programme qui tire au hasard un nombre entre 1 et 10 puis demande à l'utilisateur de rentrer un nombre entre 1 et 10 jusqu'à ce qu'il trouve le nombre secret.

Exercice 5. Nombres mystères
Que fait la fonction suivante ?

```
1 def mystere(n):
2     for i in range(2,n-1):
3         if n%i == 0:
4             return False
5     return n>1
```

Écrire un programme qui demande un nombre n à l'utilisateur et indique *tous* les nombres premiers qui sont inférieurs ou égaux à n.

Exercice 6. Liste

1. Écrire une fonction `listeAleatoire` qui prend un entier `n` en argument et renvoie une liste de taille `n` contenant des nombres entre 1 et 10 tirés au hasard. Vous utiliserez pour ce faire la fonction `randint`.
2. Écrire un programme qui crée une liste de taille 20 avec la fonction `listeAleatoire` et l'affiche.
3. Écrire une fonction `maxListe` qui prend une liste `l` en argument et renvoie le plus grand entier apparaissant dans la liste.

   Par exemple, `maxListe([1,2,4,1])` renverra `4`.
4. Compléter le programme de la question 2 pour qu'il affiche en plus le maximum de la liste.
5. Écrire une fonction `singletons` qui prend une liste d'entiers et renvoie la liste contenant les mêmes nombres mais au plus une fois et dans le même ordre.

   Par exemple, `singletons([2,1,2,1,3,2,1,4]` renverra la liste `[2,1,3,4]`.
6. Compléter le programme de la question 4 pour qu'il affiche en plus la liste sans doublons.
7. Écrire une fonction `nbOccurences` qui prend un nombre `n` et une liste `l` et renvoie le nombre de fois que l'entier `n` apparaît dans la liste `l`. Par exemple,

   ```
   nbOccurences(1,[1,2,4,1])
   2
   nbOccurences(5,[1,2,4,1])
   0
   nbOccurences(2,[1,2,4,1])
   1
   ```

8. Compléter le programme de la question 6 pour qu'il affiche pour chaque élément de la liste le nombre de fois où il apparaît dans la liste.

   Par exemple, si la liste est `[1,1,9,1,2,9,4,2,2,1,1,1,9,1,2,9,4,2,2,1]`, le programme affichera :

```
1 apparait 8 fois
2 apparait 6 fois
4 apparait 2 fois
9 apparait 4 fois
```

9. Écrire une fonction `plusFrequent` qui prend une liste `l` d'entiers en argument et qui renvoie le nombre qui apparaît le plus de fois dans la liste. Si plusieurs nombres sont possibles, on renverra le plus grand.

   Par exemple, dans la liste `[1,2,1,1,2,2,3,3,4]`, les nombres 1 et 2 apparaissent tous les deux 3 fois. On renverra donc 2 qui est le plus grand des candidats.

Exercice 7. Morpion ★

Le but de l'exercice est de réaliser le jeu du *morpion*. Le résultat est donné dans le programme `morpion.pyc`. Vous pouvez lancer le programme, en tapant :

```
python3 morpion.pyc
```

Afin que votre programme soit lisible et bien structuré, nous vous conseillons de :

1. fabriquer une liste de taille 9 représentant les cases, et d'utiliser 3 valeurs différentes pour décrire l'état d'une case.
2. découper votre code en fonctions :
   (a) une fonction qui affiche la grille avec les pions joués,
   (b) une fonction qui teste si un joueur a gagné,
   (c) une fonction qui teste si la grille est pleine,
   (d) une fonction qui fait jouer un des 2 joueurs,
   (e) le reste dans la boucle principale.

Si vous avez fini, faites une version où l'on peut jouer contre l'ordinateur.

TP 6 - Snake

Récupérer sur la page du cours le fichier `iutk.py` (celui de ce TP) qui fournit la bibliothèque graphique que nous allons utiliser et placez-le dans votre répertoire de travail.

Exercice 1. Snake

Le but de l'exercice est de réaliser le jeu *serpent*. Dans ce jeu, on contrôle un serpent avec les flèches du clavier. Pour gagner, le serpent doit manger toutes les pommes. On perd si le serpent sort de la fenêtre ou s'il se mord la queue.

Le résultat est donné dans le programme `snake.pyc`. Vous pouvez lancer le programme, en tapant :

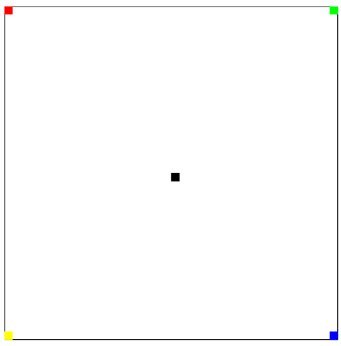
```
python3 snake.pyc
```

Récupérer le programme `snake-init.py`. Le programme contient quelques fonctions déjà faites et des fonctions que vous devrez compléter.

On découpe la fenêtre de `400x400` en carrés de `10x10`. On obtient ainsi `40x40` cases. Dans le programme, une case est représentée par une liste à deux éléments. La case en haut à gauche est représentée par la liste `[0,0]`, la case en bas à gauche par la liste `[0,39]`, ...

La fonction `afficheCase` affiche une case dans une couleur donnée. Par exemple,

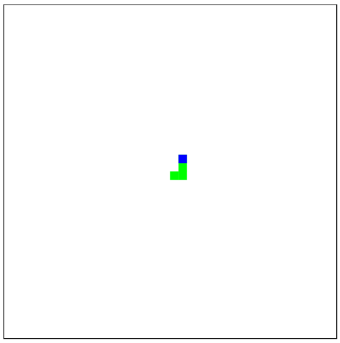
```
1 afficheCase([0,0],'red')
2 afficheCase([39,0],'green')
3 afficheCase([0,39],'yellow')
4 afficheCase([39,39],'blue')
5 afficheCase([20,20],'black')
```



Un serpent est une liste de cases, **la première étant la queue et la dernière case étant la tête**. Par exemple, la liste de cases :

```
1 serpent=[[20,20],[21,20],[21,19],[21,18]]
2 afficheSerpent(serpent)
```

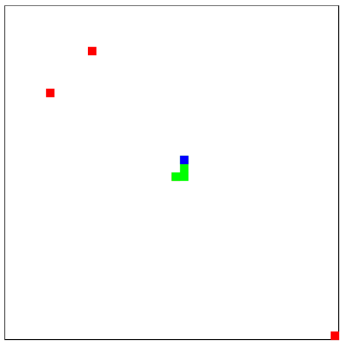
correspondra au serpent ci-dessous :



1. Ecrivez les fonctions `afficheCases` et `afficheSerpent`. La description de ces fonctions est en commentaires dans le fichier `snake-init.py`.
2. Testez vos fonctions avec le programme ci-dessous :

```
1 afficheCases([[10,5],[20,20],[39,39],[5,10]],'red')
2 serpent=[[20,20],[21,20],[21,19],[21,18]]
3 afficheSerpent(serpent)
```

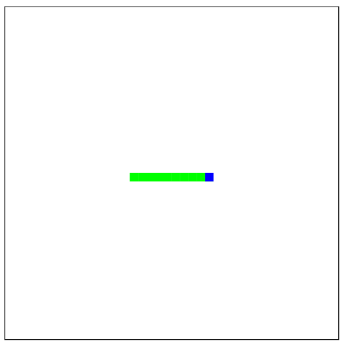
Vous devez obtenir :



3. Ecrivez la fonction `nouveauSerpent`.
4. Testez la fonction avec le programme ci-dessous :

```
1 serpent=nouveauSerpent(10)
2 afficheSerpent(serpent)
```

Vous devez obtenir :

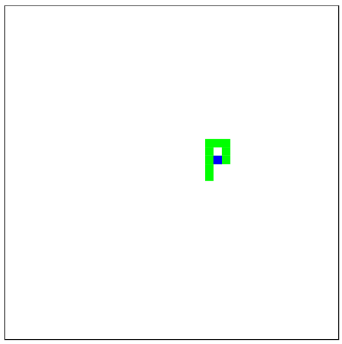


5. Ecrivez la fonction `deplaceSerpent`. La fonction prend un serpent et une direction et renvoie le serpent après un déplacement d'un pas dans cette direction. La direction est une chaîne de caractères qui vaut `'droite'`, `'gauche'`, `'haut'` ou `'bas'`.

6. Testez la fonction avec le programme ci-dessous :

```
serpent=nouveauSerpent(10)
serpent=deplaceSerpent(serpent,'haut')
serpent=deplaceSerpent(serpent,'haut')
serpent=deplaceSerpent(serpent,'haut')
serpent=deplaceSerpent(serpent,'haut')
serpent=deplaceSerpent(serpent,'droite')
serpent=deplaceSerpent(serpent,'droite')
serpent=deplaceSerpent(serpent,'bas')
serpent=deplaceSerpent(serpent,'bas')
serpent=deplaceSerpent(serpent,'gauche')
afficheSerpent(serpent)
```

Vous devez obtenir :



7. Ecrivez les fonctions `estSorti` et `aMordu`.
8. Testez les fonctions avec le programme suivant :

```
serpent1=nouveauSerpent(30)
serpent2=serpent1*1
serpent2=deplaceSerpent(serpent2,'droite')
serpent2=deplaceSerpent(serpent2,'droite')
serpent2=deplaceSerpent(serpent2,'droite')
serpent2=deplaceSerpent(serpent2,'droite')
serpent2=deplaceSerpent(serpent2,'droite')
serpent2=deplaceSerpent(serpent2,'droite')
serpent3=serpent1*1
serpent3=deplaceSerpent(serpent3,'haut')
serpent3=deplaceSerpent(serpent3,'haut')
serpent3=deplaceSerpent(serpent3,'gauche')
serpent3=deplaceSerpent(serpent3,'gauche')
serpent3=deplaceSerpent(serpent3,'bas')
serpent3=deplaceSerpent(serpent3,'bas')

print('Serpent 1 :',estSorti(serpent1),aMordu(serpent1))
print('Serpent 2 :',estSorti(serpent2),aMordu(serpent2))
print('Serpent 3 :',estSorti(serpent3),aMordu(serpent3))
```

```
Serpent 1 : False False
Serpent 2 : True False
Serpent 3 : False True
```

9. Nous allons maintenant écrire la boucle principale du jeu. Voici la code que vous devez compléter pour que votre serpent avance...

```
creeFenetre(400,400)
ready()

### creer et afficher le serpent ###

dir="droite"
temps=0

while True:
    dir=miseAJourDirection(dir)
    if temps%10==0:
        effaceTout()
        ### deplacer et afficher le serpent ###
    temps=temps+1
    miseAJour()
fermeFenetre()
```

10. Modifier la boucle principale pour afficher `Perdu!` lorsque le serpent sort ou qu'il se mord la queue.
11. Il faut maintenant rajouter les pommes sur la grille. Ecrivez les fonctions `creePommes`, `affichePommes`. Testez qu'elles fonctionnent bien.
12. Ecrire la fonction `mangePommes`.
13. Testez votre fonction avec le programme suivant :

```
pommes=[[21,16],[10,10]]
serpent=[[20,20],[21,20],[21,19],[21,18]]
serpent=deplaceSerpent(serpent,'haut')
pommes=mangePommes(pommes,serpent)
print(pommes)
serpent=deplaceSerpent(serpent,'haut')
pommes=mangePommes(pommes,serpent)
print(pommes)
```

```
[[21, 16], [10, 10]]
[[10, 10]]
```

14. Rajoutez les pommes dans votre boucle principale et afficher `Gagné!` lorsqu'il ne reste plus de pommes.
15. Faites en sorte que votre serpent grandisse de 3 cases lorsqu'il mange une pomme.

TP 7 - Fonctions, chaînes de caractères

La documentation en ligne de Python 3 se trouve ici : `http://docs.python.org/3/`. Consultez-la régulièrement si vous avez des doutes sur une fonction ou une fonctionnalité.

Attention : dans tout le TP, chaque fonction que vous écrivez doit être précédée d'un commentaire qui indique ce qu'elle fait en fonction des paramètres donnés !

Exercice 1. Chaînes de caractères

1. Déclarer et faire afficher la chaîne de caractères suivante (une seule chaîne) :

   ```
   Bonjour
   le "Monde"
   et l'Univers
   ```

2. Déclarer et afficher une deuxième chaîne de caractères identique, sans utiliser d'antislash.
3. Écrire une fonction `afficheQuestion` prenant en paramètre une chaîne *numero* et affichant la ligne suivante précédée d'une ligne blanche (avec le bon numéro) :

   ```
   ########## Question n ##########
   ```

   Par exemple `afficheQuestion('2.a')` affichera `Question 2.a`
4. On souhaite écrire une fonction prenant en paramètre une chaîne de caractères et réaffichant celle-ci avec le texte "voyez-vous" après chaque virgule et ", n'est-ce pas" avant chaque point. Par exemple, à partir du texte suivant :

   ```
   Hier, j'ai mangé une pomme.
   Après, j'ai eu des hallucinations, plein d'hallucinations.
   ```

   on obtient :

   ```
   Hier, voyez-vous, j'ai mangé une pomme, n'est-ce pas.
   Après, voyez-vous, j'ai eu des hallucinations, voyez-vous, plein d'halluci-
   nations, n'est-ce pas.
   ```

   Remarque : pour utiliser des caractères accentués comme dans cet exemple, vous devez ajouter le sha-bang suivant au début de votre fichier Python :

   ```
   #!/usr/bin/env python
   # -*- coding: utf-8 -*-
   ```

   (a) Écrire cette fonction nommée `langageV1` en utilisant `for` et `range`.

   (b) Ré-écrire la même fonction nommée `langageV2` en utilisant `for`, **mais pas** `range`.

   (c) Écrire la fonction `langageV3` qui **n'affiche pas** la chaîne obtenue mais la **renvoie**.

   (d) Modifier la fonction `langageV3` pour qu'elle prenne en paramètre les chaînes remplaçant les points et les virgules. On pourra, par exemple, tester :

   ```
   print(langageV3(texte, ", t'as vu,", ", quoi."))
   ```

   (e) **(facultatif)** Écrire la fonction `langageV4` qui fait exactement la même chose, mais sans utiliser de boucle (ni `while`, ni `for`).
   Indice : chercher dans la documentation en ligne relative aux chaînes de caractères : `http://docs.python.org/3/library/stdtypes.html#text-sequence-type-str`.

   (f) **(facultatif)** Refaire la fonction précédente avec une seule instruction.

Exercice 2. Intervalles
Répondre aux questions suivantes en une seule ligne en utilisant `range`. Si vous ne savez pas comment faire, allez voir la doc.

1. Tester si un entier entré par l'utilisateur est entre 0 et 10 (non inclus). Pouvez-vous faire la même chose avec un flottant ?
2. Tester si un entier entré par l'utilisateur est entre 10 et 20 (non inclus).
3. Afficher la liste des entiers entre 100 et 150. Pourquoi ne pouvez-vous pas faire la même chose avec des flottants ?
4. Afficher la liste des entiers pairs entre 100 et 150.
5. Afficher la liste des entiers impairs entre 150 et 200.
6. Afficher la liste décroissante des entiers 150 et 200.
7. Afficher la liste décroissante des entiers multiples de 3 entre -150 et -200.

Exercice 3. Code de César : cryptographie et cryptanalyse
Le but de cet exercice est de réaliser un système permettant de crypter des messages, en utilisant le code de César : on part d'un message en clair et on change toutes les lettres en les décalant d'un même nombre de positions (ce nombre est appelé la *clé*). Pour décoder le message il suffit alors de décaler dans l'autre sens, à condition de connaître la clé. Pour cet exercice, nous vous encourageons à chercher dans la doc quelles sont les fonctions qui peuvent vous être utiles. **Attention, ce qui suit ne fonctionne que pour des caractères sans accent !**

1. On va commencer par écrire la fonction d'encodage du texte : `encrypte(texteEnClair,cle)`. Toutes les lettres sont cryptées, mais pas les blancs et la ponctuation. Par exemple, ce code affiche les deux lignes notées en commentaire :

```
1 cache=encrypte('Toto est une girafe, avec des petites jambes.',12)
2 print(cache)                # Fafa qef gzq sudmrq, mhqo pqe bqfufqe vmynqe.
3 print(decrypte(cache,12)) # Toto est une girafe, avec des petites jambes.
```

   Pour cela, nous allons découper le travail :
   (a) Écrire une fonction `encrypteLettre` qui prend une seule lettre en paramètre et renvoie la lettre encodée (attention à la ponctuation et aux majuscules). Pour décaler les lettres, vous aurez besoin de 2 fonctions de bases de Python : `ord` et `chr`. La doc est là : `http://docs.python.org/3/library/functions.html`. La fonction `ord` prend une chaine contenant un seul caractère et renvoie son *code ascii* (`ord('a')` donne 97, `ord('b')` 98, etc...). Et la fonction `chr` fait l'inverse : elle renvoie le caractère en fonction du code ascii. Par exemple, le programme suivant permet de créer une liste contenant les lettres de l'alphabet :

```
1 alphabet=[]
2 for i in range(26):
3   alphabet.append(chr(ord('a')+i))
```

   Vous devez calculer l'indice de la lettre dans l'alphabet, puis le décaler avec la clé (en prenant soin de rester entre 0 et 25 : par exemple, 'z' décalé de 2 donne 'b') et retrouver le caractère correspondant. Pour vérifier : lorsque l'ont fait `for c in alphabet: print(encrypteLettre(c,10), end=' ')` :, on doit obtenir :
   `k l m n o p q r s t u v w x y z a b c d e f g h i j.`
   Si vous êtes coincé, appelez votre chargé de TP.
   (b) On constate que l'on fait 2 fois le même travail, pour une majuscule ou une minuscule. Rajouter une fonction auxiliaire qui prend 'a' ou 'A' en paramètre et permet de ne faire le calcul qu'une seule fois.

(c) Écrire une fonction `encrypteMot` qui prend un mot et renvoie le mot encodé.

(d) Enfin écrire `encrypte` qui renvoie le texte codé.

2. On passe au décodage. Écrire la fonction `decrypte(texteCache,cle)` qui permet de décoder (et renvoyer) un texte qui a été crypté avec la clé. Soyez astucieux, ne copiez-collez pas votre code, réutilisez plutôt vos fonctions !
3. Passons à la cryptanalyse, c'est-à-dire l'art de déchiffrer les messages sans avoir la clé ! Pour cela, nous allons essayer d'utiliser nos connaissances sur les fréquences des lettres dans un texte. En effet, vous n'êtes pas sans savoir que la lettre la plus commune en français est le 'e'... Donc, pour savoir quelle clé a permis de coder le texte, il suffit de tester toutes les clés possibles et de regarder celle qui donne le texte contenant le plus de 'e'. Écrire la fonction `devineCle(texteCache)` qui renvoie la clé devinée en utilisant ce principe et testez-la sur l'exemple. Attention, cette technique fonctionne d'autant mieux que le texte est long. Essayez de retrouver le texte original dans les deux cas suivants :
   Texte 1 :
   `Tuqd, v'mu ymzsq gzq bayyq. Mbdqe, v'mu qg pqe tmxxgouzmfuaze, bxquz p'tmxxgouzmfuaze.`
   `Vq hakmue pqe qxqbtmzfe, pqe xuoadzqe qf pqe bmzpme. Qf uxe qfmuqzf daeqe, fage daeqe.`

   Texte 2 :
   `Jli c'rsrkkrek ul mrjzjkrj, le rezdrc rl kyfiro zeuzxf, r c'rzxlzccfe jrwire, ez le trwriu, ez le tyriretfe, drzj gclkfk le rikzjfe, j'rmretrzk, kirzerek le size u'rcwr. Zc j'rggiftyr, mflcrek c'rgcrkzi u'le tflg mzw, drzj c'rezdrc gizk jfe mfc, uzjgrirzjjrek urej cr elzk rmrek hl'zc rzk gl c'rjjrzcczi.`

   Si vous voulez savoir pourquoi ça ne marche pas sur le second texte : `http://fr.wikipedia.org/wiki/La_Disparition_%28roman%29`
4. **(facultatif)** Cette façon de faire n'est pas très efficace, elle force à décoder 26 fois... Il vaudrait mieux chercher la lettre la plus fréquente dans le texte et chercher la clé permettant de la faire correspondre avec un 'e'. Écrire la fonction `devineCle2` qui utilise ce principe.
   Indice : (une fois que vous connaissez la position `pos` dans l'alphabet (entre 0 et 25) de la lettre la plus fréquente, il suffit de calculer : `( pos+ord('a')-ord('e') )%26`
5. On vient de voir qu'il n'est pas très difficile de décoder le message, même sans la clé. Cette méthode de cryptographie n'est donc pas très efficace. Pour empêcher de deviner en utilisant les fréquences, on va crypter en décalant de 1 la clé de codage à chaque mot. Avec l'exemple de "Toto...", ça donne : `Fafa rfg ibs vxgput, qlus uvj hwlalwk ctfuxl`.
   Écrire les fonctions `encrypteMieux(texte,cle)` et `decrypteMieux(texte,cle)` qui utilisent ce principe.
6. On se rend compte que les fonctions `encrypte` et `encrypteMieux` sont quasiment les mêmes. Transformez-les en une unique fonction `encrypteEncoreMieux` qui prend également en paramètre le décalage utilisé pour chaque mot : il suffit de mettre 0 pour retrouver `encrypte` et 1 pour `encrypteMieux` :

```
txt='Voici le texte'
print( encrypte(txt,12) )      # affiche: Hauou xq fqjfq
print( encrypteEncoreMieux(txt,12,0) )  # Hauou xq fqjfq
print( encrypteMieux(txt,12) )          # Hauou yr hslhs
print( encrypteEncoreMieux(txt,12,1) )  # Hauou yr hslhs
```

   Pensez à utiliser cette fonction pour décrypter aussi.
7. **(facultatif)** Essayez de modifier votre code pour gérer les accents...

TP 8 - Fonctions, chaînes de caractères

La documentation en ligne de Python 3 se trouve ici : `http://docs.python.org/3/`. Consultez-la régulièrement si vous avez des doutes sur une fonction ou une fonctionnalité.

Attention : Vous devez terminer le TP7 avant de faire l'exercice du TP8.

Exercice 1. Codage par substitution
Attention : dans cet exercice, seules les minuscules sont modifiées.
Dans cet exercice, on s'intéresse à un chiffrement plus évolué que celui vu précédemment. Dans ce chiffrement, on remplace chaque lettre par une autre lettre de l'alphabet. Par exemple,

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

N'importe quelle permutation de l'alphabet est donc autorisée, contrairement au chiffrement de César qui se restreint aux rotations. Dans le cas de la correspondance ci-dessus, le texte `'je vais bien, merci!'` sera codé en `'qv ebrh yrvm, nvixr!'`. La clé est représentée par la chaîne formée du caractère associé à `a`, puis du caractère associé à `b`, et ainsi de suite. Dans notre exemple, la clé de chiffrement est `byxwvutsrqponmlkjihgfedcza`.

Pour déchiffrer un message, il suffit d'appliquer la clé qui remet toutes les lettres à leur place. Dans notre exemple, la correspondance à appliquer est donc :

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

La clé de déchiffrement est donc `zaxwvutsrqponmlkjihgfedcby`.

1. Écrire une fonction `chiffre` qui prend en arguments un texte et un clé et renvoie le texte chiffré.
2. Écrire une fonction `cleDecode` qui prend en argument la clé et renvoie la clé de décodage associée.

Pour déchiffrer un texte dont on ne connaît pas la clé, on ne peut pas essayer comme précédemment toutes les clés possibles. Il y en a 26!, c'est-à-dire plus de $4 \cdot 10^{26}$. On utilise comme précédemment la fréquence des lettres dans le texte. Pour plus de détails sur l'approche à appliquer, voir :

`http://fr.wikipedia.org/wiki/Analyse_fr%C3%A9quentielle`

Récupérez le fichier `secret.txt` et déchiffrez-le suffisamment pour trouver l'œuvre dont est tiré le texte.

TP 9 - Sudoku

Récupérez sur la page du cours le fichier `iutk.py` (celui de ce TP) qui fournit la bibliothèque graphique que nous allons utiliser et placez-le dans votre répertoire de travail.

Récupérez également le fichier `sudoku-init.py` qui est une correction du TP d'Algo sur le sudoku et qui sert de base pour ce TP. Le but, ici, est de créer une interface jouable pour ce sudoku.

Lisez le sujet en entier avant de commencer !

Exercice 1. Version Light

Dans cet exercice, nous allons créer une interface graphique basique pour le sudoku. Le fichier `sudoku-light.pyc` vous permet de voir quel est le résultat attendu. Attention, dans cet exemple, nous avons volontairement réduit le nombre de cases libres, pour tester la fin de partie plus rapidement. Vous pouvez faire de même pour débuguer, mais au final vous pourrez utiliser les valeurs par défaut, qui laissent plus de cases libres.

1. Dans un premier temps, créez (`genererPuzzle`) et affichez une grille de sudoku aléatoire.
2. Ensuite, faites en sorte que l'on puisse sélectionner/dé-sélectionner une case en cliquant dessus. Affichez un cadre bleu autour de la case pour montrer qu'elle est sélectionnée.
3. Si une case est sélectionnée, ajoutez de quoi rentrer au clavier la valeur que l'on souhaite jouer.
4. Rajoutez de quoi tester et afficher que le joueur a gagné.

Conseils :

- Dans l'exemple, nous utilisons les paramètres suivants :
  `largeurFenetre=450`
  `hauteurFenetre=largeurFenetre`
  `tailleCase=largeurFenetre//9`
- Les coordonnées (dans la fenêtre) du milieu de la case (i, j) de la matrice sont :
  `(i+0.5)*tailleCase, (j+0.5)*tailleCase`
- Les cordonnées (x, y) d'un point (dans la fenêtre) correspondent dans la matrice aux indices donnés par :
  `x//tailleCase, y//tailleCase`
- Pensez à utiliser la fonction `attenteClicOuTouche()` de la bibliothèque `iutk.py`.

Exercice 2. Version Full

Modifiez le sudoku pour que l'on ne puisse pas modifier une case de la grille initiale et que les cases jouées par l'utilisateur apparaissent en bleu. Le fichier `sudoku-full.pyc` vous permet de voir quel est le résultat attendu.

Exercice 3. Version Star

S'il vous reste du temps, voici des pistes pour améliorer le sudoku :

1. Rajoutez une touche permettant de réinitialiser le sudoku, c'est-à-dire de n'effacer que les cases que l'utilisateur doit remplir.
2. Rajoutez une touche permettant d'abandonner et d'afficher la solution.
3. Rajoutez de quoi faire afficher les cases mal placées (par rapport à la solution) en rouge.
4. En fait, c'est un peu trop dirigiste, faîtes plutôt afficher en rouge les cases qui ne satisfont pas les conditions du sudoku (par rapport au remplissage partiel de la grille).

TP 10 - Dictionnaires et ensembles

Exercice 1. Analyse d'un fichier texte

Récupérez les fichiers `miserables.txt` et `dico.txt` que vous allez utiliser pour tester vos fonctions.

1. Écrire une fonction `occurrencesLettres` qui prend en paramètre un nom de fichier et renvoie un dictionnaire qui associe à chaque lettre le nombre de fois où elle apparaît dans le fichier. Pour tester qu'un caractère `c` est une lettre, vous pouvez utiliser `c.isalpha()` qui renvoie `True` si `c` est une lettre et `False` sinon.

   Vous ne devez lire le fichier qu'une seule fois !

   Si vous testez votre fonction sur le fichier `miserables.txt`, vous devez obtenir :

   ```
   print(occurencesLettres('miserables.txt'))

   {'d': 17744, 'e': 88828, 'f': 5865, 'g': 4509, 'a': 44998, 'b': 4978,
    'c': 15904, 'l': 30212, 'm': 15440, 'n': 34706, 'o': 26010,
   'h': 5119, 'i': 39376, 'j': 2888, 'k': 20, 't': 38794, 'u': 32391,
    'v': 10174, 'w': 21, 'p': 13345, 'q': 6626, 'r': 31846, 's': 37652,
    'x': 2074, 'y': 1735, 'z': 962}
   ```

2. Écrire une fonction `pourcentages` qui prend en argument un dictionnaire associant à chaque clé un nombre d'occurrences et renvoie un nouveau dictionnaire associant à chaque clé le pourcentage que représente les occurrences de cette clé sur le total des occurrences du dictionnaire.

   Par exemple, si `d = { 'a' : 2 , 'b' : 1 , 'c' : 1}`, `pourcentages(d)` renverra `{ 'a' : 50.0 , 'b' : 25.0 , 'c' : 25.0}`, puisque les deux 'a' représentent 50 % des 4 occurrences du dictionnaire.

   Si vous testez votre fonction sur le fichier `miserables.txt`, vous devez obtenir :

   ```
   occL=occurencesLettres('miserables.txt')
   print(pourcentages(occL))

   {'y': 0.33872362690031765, 'x': 0.40490651423127305, 'z': 0.18781102540524816,
    'u': 6.32368703108253, 't': 7.573743159637419, 'w': 0.004099824878908744,
    'v': 1.986267538953217, 'q': 1.2935923641737779, 'p': 2.6053410956684373,
    's': 7.350790778127239, 'r': 6.217286813987041, 'm': 3.014347434778619,
   'l': 5.898281392456712, 'o': 5.077925957162687, 'n': 6.775643916543183,
    'i': 7.6873668777100335, 'h': 0.9993811216730409, 'k': 0.003904595122770232,
    'j': 0.5638235357280215, 'e': 17.34186877827171, 'd': 3.46415679292175,
    'g': 0.8802909704285489, 'f': 1.1450225197523705, 'a': 8.784948566720745,
    'c': 3.1049340416268887, 'b': 0.9718537260575107}
   ```

3. Écrire une fonction `triParOccurrences` qui prend un dictionnaire associant à chaque clé un nombre d'occurrences et renvoie la liste des clés triées par ordre décroissant d'occurrences.

   Si vous testez votre fonction sur le fichier `miserables.txt`, vous devez obtenir :

```
occL=occurencesLettres('miserables.txt'))
print(triParOccurrences(occL))

['e', 'a', 'i', 't', 's', 'n', 'u', 'r', 'l', 'o', 'd', 'c',
  'm', 'p', 'v', 'q', 'f', 'h', 'b', 'g', 'j', 'x', 'y', 'z', 'w', 'k']
```

4. Écrire une fonction `occurrencesMots` qui prend en paramètre un nom de fichier et renvoie un dictionnaire qui associe à chaque mot le nombre de fois où elle apparait dans le fichier. Pour tester qu'une chaîne `m` est bien un mot vous pouvez utiliser `m.isalpha()`.

   Vous ne devez lire le fichier qu'une seule fois !

   Si vous testez votre fonction sur le fichier `miserables.txt`, vous devez obtenir :

```
occMots=occurencesMots('miserables.txt'))
print(occMots['le'])
print(occMots['agreable'])

2563
1
```

5. Écrire une fonction `top25` qui prend un dictionnaire d'occurrences et renvoie la liste des 25 premières clés qui ont le plus grand nombre d'occurrences par ordre décroissant d'occurrences. Vous pouvez utiliser `triParOccurrences`.

   Si vous testez votre fonction sur le fichier `miserables.txt`, vous devez obtenir :

```
occMots=occurencesMots('miserables.txt'))
print(top25(occMots))

['de', 'la', 'il', 'et', 'a', 'le', 'l', 'un', 'les', 'que', 'd', 'une',
 'qui', 'qu', 'en', 'etait', 'dans', 'est', 'ce', 'des', 'avait', 'ne',
'pas', 'se', 'n']
```

6. Le fichier `dico.txt` contient une liste des mots de la langue française. Écrire une fonction `correcteur` qui prend en paramètre un nom de fichier et qui renvoie la liste des mots de ce fichier qui ne sont pas des mots apparaissant dans `dico.txt`.

```
print(correcteur('miserables.txt'))

['blondeau', 'greif', 'josefa', 'com', 'anywhere', 'ecria', 'xiv', 'xii',
 'ailly', 'nisi', 'dignifier', 'needham', 'barcelonnette', 'please',
 'cythere', 'caron', 'delaverderie', 'archidiaconats', 'arabie', 'toulon',
 'lebrun', 'beure', 'coural', 'parvulos', 'boujean', 'aurele', 'henri' ...
```

7. ★ Écrire une fonction `maxLongueur` prenant un nom de fichier en paramètre et affichant pour chaque taille la liste des mots de cette taille qui apparaissent le plus dans le texte.

```
maxLongueur('miserables.txt')

0 []
1 ['a']
2 ['de']
3 ['les']
4 ['dans']
5 ['etait']
6 ['eveque']
```

```
7 ['valjean']
8 ['monsieur']
9 ['madeleine']
10 ['thenardier']
11 ['monseigneur']
12 ['champmathieu']
13 ['conventionnel']
14 ['habituellement']
15 ['continuellement', 'malheureusement']
16 ['particulierement']
17 []
18 ['consciencieusement']
```

Exercice 2. Cherchez l'assassin !
Un meurtre horrible vient d'être commis et vous êtes chargé de l'enquête.

- **1ère partie de l'enquête :**
  Après plusieurs jours d'investigation, la liste de coupables potentiels est encore longue (le fichier `noms.txt`), mais vous avez tout de même réussi à déterminer que l'assassin se trouvait forcément au bar ou en train de faire une partie de poker à 15h et qu'il avait rendez-vous avec Erin à 20h. Ceci va vous permettre d'établir la liste des suspects.
- **2ème partie de l'enquête :**
  Chaque suspect a été interrogé sur son agenda le jour du meurtre. Une seule personne à intérêt à mentir : le coupable ! Recoupez les réponses des suspects et les informations données pour chaque activité et démasquez l'assassin...

Chaque fichier fourni ne peut être lu qu'UNE SEULE FOIS.

Le dossier de police se trouve dans l'archive `enquete.tgz`. Dans la ville où a été commis le meurtre, les différentes activités possibles sont listées ci-dessous :

```
activites = ["Bar", "Poker", "Golf", "Patinoire", "Cinema"]
```

Grâce au système de surveillance des établissements où elles ont lieu, vous avez recueilli les noms de toutes les personnes qui ont participé aux activités ainsi que les heures où elles étaient présentes (ces informations sont fiables). Ces données sont conservées dans les fichiers du répertoire `activites`, chaque ligne de fichier étant de la forme : `nom:heure`.

Vous avez également obtenu les agendas de toutes les personnes visées par l'enquête (ces informations proviennent des interrogatoires, elles ne sont pas fiables). Ils se trouvent dans le répertoire `activites`, chaque ligne de fichier étant de la forme : `activité:heure`.

Écrivez le programme permettant d'établir la liste de suspects de la première partie de l'enquête, puis de déterminer qui est le coupable.

TP 12 - Python avancé

Exercice 1. Exceptions

Dans cet exercice, on va écrire un programme permettant de chercher les occurrences d'une expression régulière dans un fichier.

1. Écrire une fonction `afficheRegexp(chaine,regexp)` qui prend une chaîne `chaine` et une expression régulière `regexp` et affiche tous les "morceaux" de `str` correspondants. Pour cela, vous utiliserez la fonction `findall` du module `re` de Python.
   Attention, si l'expression est mal formée vous devez afficher le message suivant et renvoyer l'exception provoquée : `"L'expression reguliere ... est incorrecte"`
   Par exemple, l'appel :

   ```
   afficheRegexp("(totot-1)*(2+3)","\([^)]*\)")
   ```

   donne : `(totot-1)` et `(2+3)` et l'appel `afficheRegexp("(()(()))","\())")` donne :

   ```
   L'expression reguliere \()) est incorrecte
   Traceback (most recent call last):
   ...
   sre_constants.error: unbalanced parenthesis
   ```

2. Faire en sorte de ne pas afficher la partie `"Traceback"` pour l'appel précédent.
3. Écrire une fonction `essaieOuvrir()` qui demande à l'utilisateur un nom de fichier et essaie de l'ouvrir. S'il l'ouverture ne pose pas de problème, la fonction renvoie le résultat de `open`. Malheureusement, il peut y avoir des problèmes lors de l'ouverture... Nous allons en traiter 2 :
   (a) Si le fichier n'existe pas, la fonction redemande un nom de fichier à l'utilisateur.
   (b) Si le fichier n'a pas les bonnes permissions, on renvoie une `NameError("Permission")`

   Dans la doc, vous trouverez comment distinguer ces 2 cas en utilisant la variable `errno`.
4. Nous pouvons maintenant écrire le programme principal. Utiliser `essaieOuvrir()` pour demander un nom de fichier à l'utilisateur. En cas de problème de permissions, le programme s'arrête en disant : "Changez les permissions de votre fichier !". Si le fichier n'existe pas le programme redemande un nom de fichier.
   Ensuite, le programme demande une expression régulière à l'utilisateur et utilise la fonction `afficheRegexp` pour la chercher dans le fichier. Si elle est mal formée, le programme s'arrête en disant : "L'expression reguliere ... est incorrecte".

Exercice 2. En une ligne...

1. ... afficher l'alphabet : ['a', 'b', 'c', ...].
2. ... afficher la liste des triplets croissants d'entiers < 10 : [(1, 2, 3), (1, 2, 4), ..., (1, 3, 4), ..., (2, 3, 5), ..., (7, 8, 9)].
3. ... afficher la liste [('a', 0), ('b', 1), ('a', 2), ('b', 3), ..., ('a', 18), ('b', 19)].
4. ... afficher **l'ensemble** des nombres premiers inférieurs à 100.
5. ... afficher **la liste** du 10e au 20e nombre premier : [26, 29, 31, 33, 34, 37, 38, 39, 41, 43].

Exercice 3. Modules pour les dictionnaires français et anglais

Dans cet exercice, vous allez manipuler des modules pour gérer des dictionnaires dans des langues différentes. Les modules sont simplement des fichiers indépendants auxquels vous pouvez accéder par la commande `import`.

1. Dans un module nommé `fonctionsDico.py`, écrire les 2 fonctions suivantes :
   (a) la fonction `fabriqueDicoSet(fichier)` qui prend un fichier de mots (1 par ligne) et renvoie l'ensemble de tous les mots de ce fichier.
   (b) la fonction `prefixeDico(mot, dico)` qui renvoie `True` si le mot a un préfixe dans le dictionnaire et `False sinon`. Par exemple, en français, "sympa" n'a pas de préfixe dans le dictionnaire alors que "ballon" en a ("bal").
2. Vous pouvez ensuite récupérer les modules `dicoFR.py` et `dicoAN.py` et les fichiers de dictionnaire associés sur la page web. Ces modules permettent de créer des dictionnaires en français et en anglais.
   En utilisant les dictionnaires (`dico`) qui se trouvent dans ces modules, créer l'ensemble de tous les mots communs de l'anglais et du français (il y en a environ 12 000). Est-il possible d'appeler cet ensemble `dico` ?
3. Afficher la liste des 100 premiers mots communs dans l'ordre alphabétique : ['a', 'abandon', 'abattoir', 'abbatial', ...]
4. Afficher la liste de mots qui commencent par un 'x' et finissent par un 'e' dans le dictionnaire français : ['xyste', 'xerographie', 'xylocope', ...]
5. En une ligne, calculer le pourcentage de mots qui n'ont pas de préfixes dans le dictionnaire anglais (et comparer avec le français).
   Quel est le problème ? Corriger les modules fournis pour que cela fonctionne.

Exercice 4. Parcours

Un `plateau` est un tableau à deux dimensions qui contient des Booléens. Si `plateau[i][j]` vaut `True`, il y a un mur et sinon la case est libre.

```
1  plateau = [[True, False, False, False],
2             [False, True, True, False]]
```

Le but de l'exercice est d'écrire une fonction `chemin` prenant un plateau et les coordonnées de deux cases `deb` et `fin` et renvoie `True` si l'on peut aller de la case `deb` à la case `fin` en se déplaçant horizontalement et verticalement.
Dans l'exemple, `chemin(plateau,(1,3),(1,0))` renvoie `False` et `chemin(plateau,(1,3),(0,1))` renvoie `True`.

1. Écrire une fonction `voisinsCase` qui prend un plateau et une case et renvoie l'ensemble de ses voisins immédiats horizontaux ou verticaux qui sont sur le plateau et qui sont libres.
2. Écrire une fonction `voisinsCases` qui prend un plateau et un ensemble de cases et renvoie l'ensemble de tous les voisins de ces cases.
3. Écrire une fonction `accessibles` qui prend un plateau et une case et renvoie l'ensemble des cases que l'on peut atteindre depuis cette case en se déplaçant horizontalement et verticalement.
4. Écrire la fonction `chemin`.

TP 13 - Révisions

Exercice 1. Conditions

Compléter le code ci-dessous de manière à ce qu'à la fin `a`, contiennent la plus grande valeur, `b` celle du milieu et `c` la plus petite valeur.

```
1  a=int(input())
2  b=int(input())
3  c=int(input())
4  # A faire
```

Exercice 2. Boucles

1. Écrire une fonction `factorielle` qui prend un entier `n` et renvoie n factorielle.
   On rappelle que $n!$ vaut $1 \times 2 \times 3 \cdots \times n$ pour tout $n \geq 1$ et que $0! = 1$.
2. Écrire une fonction `demande` qui demande un nombre à l'utilisateur jusqu'à ce qu'il soit compris entre 0 et 20 et le renvoie.
3. Écrire une fonction `listeCroissante` qui laisse l'utilisateur rentrer autant de nombres positifs qu'il veut, tant que chaque nombre est strictement supérieur à celui d'avant. Elle renvoie ensuite la liste des nombres entrés sans le dernier.

Exercice 3. Listes

1. Écrire une fonction `maximum` qui prend une liste d'entiers non nécessairement positifs et renvoie un tuple avec son maximum et la liste des indices où il apparait.
   Vous ne pourrez pas utilisez la fonction `max` de Python.
2. Écrire une fonction `sansDoublons` qui prend une liste d'entiers et renvoie la liste contenant les mêmes nombres mais au plus une fois et dans le même ordre.
   Par exemple, `singletons([2,1,2,1,3,2,1,4])` renverra la liste `[2,1,3,4]`.

Exercice 4. Chaînes

1. Écrire une fonction `sansPonctuation` qui prend une chaîne et renvoie la chaîne dans laquelle on a supprimé tous les symboles de ponctuation.
2. Écrire une fonction `nbMots` qui prend une chaîne et renvoie son nombre de mots.
3. Écrire une fonction `replace` qui prend une chaîne et un dictionnaire et qui renvoie la chaîne obtenue en remplaçant chaque clé du dictionnaire par sa valeur.
   Par exemple, `replace("Bien le bonjour madame",{'madame' : 'monsieur','bonjour' : 'aurevoir'})` renverra `"Bien le aurevoir monsieur"`.

Exercice 5. Dictionnaires

1. Écrire une fonction `freqChaine` qui prend un chaîne de caractères et renvoie un dictionnaire qui contient pour chaque lettre de la chaîne son nombre d'occurrences.
   Par exemple, `freqChaine('abbabba!')` renverra { `'a' : 3 , 'b' : 4, '!' : 1`}.
2. Écrire une fonction `copie` qui renvoie une copie d'un dictionnaire de listes.