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 5 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.