Structures de contrôle

Objectifs

— Savoir utiliser les structures de contrôle

Exercice 1 Le programmes suivant contient des erreurs.

```
# Volontairement, ni commentaires, ni identifiants significatifs
def main():
    print('début')
    x == 0;
    while x < 3
        print(x)
        if x = 2
            print('...')
    x =+ 1
    print(fin)

main
```

1.1. Indiquer les erreurs en précisant s'il s'agit d'erreurs de syntaxe ou d'erreurs sémantiques. En particulier, seront-elles signalées lors de la lecture du programme ou lors de son exécution.
1.2. Corriger le programme. Indiquer ce qu'affichera son exécution.

Exercice 2 : Score au 21
Le jeu du 21 se joue à deux dés à 6 faces (valeurs de 1 à 6). Chaque joueur lance les dés et le gagnant est celui qui obtient le plus de points. Ici, nous ne nous intéressons qu'au calcul du score d'un lancé.

Celles des 36 combinaisons possibles qui ont un score non nul sont :

1. Le **21** (un dé vaut 2 et l'autre 1). C'est la plus forte combinaison avec un score de 21.
2. Les 6 **paires** (les deux dés ont la même valeur). Le score d'une telle combinaison est la somme de 10 et la valeur d'un des deux dés. Par exemple, le score de 66 est 16 (10 + 6).
3. Les 5 **suites** (les valeurs des dés se suivent). Le score est alors la somme des valeurs des dés. Par exemple le score de 45 est 9.

Calculer le score en fonction des valeurs des deux dés lancés. On n'utilisera pas la multiplication. On complètera le fichier `score_21.py`, le fichier `test_score_21.py` permettra de le tester.

Exercice 3 : Argent pour Jules
À la naissance de Jules, ses parents lui ont ouvert un compte sur lequel ils ont versé 100 euros. Ensuite, à chaque anniversaire de Jules, ils ont versé 100 euros et le double de l'âge en euros sur ce compte. Par exemple, pour les 2 ans de Jules, ses parents ont versé 104 euros sur son compte.

Combien les parents de Jules ont versé d'argent (en euros) sur son compte au lendemain d'un anniversaire donné. On complètera le fichier `compte_jules_versements.py` et on utilisera le programme de test `test_compte_jules_versements.py`.

Exercice 4 : Objectif de Jules

À la naissance de Jules, ses parents lui ont ouvert un compte sur lequel ils ont versé 100 euros. Ensuite, à chaque anniversaire de Jules, ils ont versé 100 euros et le double de l'âge en euros sur ce compte. Par exemple, pour les 2 ans de Jules, ses parents ont versé 104 euros sur son compte.

On considère que le compte n'est pas rémunéré et qu'aucun autre versement ou retrait n'est fait sur ce compte.

Quel âge devra atteindre Jules avant de disposer d'une certaine somme sur son compte ? On complètera le fichier `compte_jules_objectif.py`, le fichier `test_compte_jules_objectif.py` permettra de le tester.

Exercice 5 : Réviser les tables de multiplication

On souhaite aider à la révision des tables de multiplication. Le principe est de disposer d'un programme qui pose 10 multiplications pour une table donnée (le nombre de gauche est celui de la table choisie par l'utilisateur entre 0 et 10, celui de droite est choisi aléatoirement entre 0 et 10). Le programme demande à l'utilisateur le résultat et lui indique s'il a bien répondu (« Correct » ou « Erreur »). À la fin, le programme affiche le nombre de bonnes réponses ainsi qu'un message. Le message est :

— « Excellent !» si toutes les réponses sont justes,
— « Très bien. » s'il n'a commis qu'une seule erreur,
— « Bien. » s'il n'a pas fait plus de trois erreurs,
— « Moyen. » s'il a 4, 5 ou 6 bonnes réponses,
— « Il faut retravailler cette table. » s'il a 3 bonnes réponses ou moins et
— « Est-ce que tu l'as fait exprès ?" si toutes les réponses sont fausses.

La table à réviser sera demandée en début de programme (pas de contrôle).

Le listing 1 donne un exemple d'exécution du programme.

5.1. Pour obtenir un nombre aléatoire, on utilisera la fonction `randint` du module `random`. Faire `help('random.randint')` sous l'interpréteur Python pour avoir sa documentation.

5.2. Écrire le programme demandé.

Pour cette question et les suivantes on utilisera le fichier `tables_multiplicationsN.py` avec `N` remplacé par le numéro de la question, par exemple `tables_multiplications2.py` pour cette question. D'une question à l'autre, on copiera le travail fait à la question précédente dans le nouveau fichier.

5.3. Modifier le programme pour qu'il ne pose pas deux fois de suite la même multiplication.

5.4. Compléter le programme pour afficher la question pour laquelle l'utilisateur a mis le plus de temps pour répondre ainsi la moyenne des temps de réponse. On utilisera la fonction `time` du module `time` pour récupérer l'heure actuelle.

On conseillera de réviser la table correspondant au nombre de droite de la multiplication pour laquelle l'utilisateur a mis le plus de temps pour répondre, à condition que ce temps soit supérieur d'une seconde au temps moyen. Peu importe que la réponse soit juste ou fausse ; ce qui est pris en compte ici, c'est l'hésitation à répondre. Il est possible qu'aucun conseil ne soit affiché.

Listing 1 – Exemple d'exécution du programme (version initiale)

```
Table à réviser (0 à 10) ? 7

(M1)  7 * 4 = 28
Correct

(M2)  7 * 3 = 21
Correct

(M3)  7 * 3 = 21
Correct

(M4)  7 * 7 = 48
Erreur

...

(M10)  7 * 4 = 26
Erreur

Nombre de bonnes réponses : 7/10
Bien.
```

5.5. Dès que l'utilisateur a fait 5 erreurs, on arrête les révisions et on affiche le message « 5 erreurs ! Il faut apprendre cette table de XXX » (où XXX est à remplacer par la table qui faisait l'objet de la révision).
5.6. Même question que la précédente dans le cas où l'utilisateur fait 3 erreurs consécutives. Le message affiché commencera donc par « 3 erreurs consécutives ! ».
5.7. Ajouter la possibilité pour l'utilisateur de continuer à réviser ses tables de multiplications. On lui posera la question « On continue les révisions (o/n) ? ». Le programme s'arrêtera si l'utilisateur répond « n » ou « N ». Dans le cas, contraire il fait réviser une table choisie par l'utilisateur.