Exercices – **Consolidation des acquis**

Exercice 1 : Une croix

Faire un programme qui demande à l'utilisateur une valeur impaire n et qui affiche une croix. « X ».
Voici l'affichage attendu pour n = 5 :

```
X   X
 X X
  X
 X X
X   X
```

Aide : On rappelle que l'on peut afficher les caractères les uns après les autres en rajoutant l'argument `end=""` *à la commande* `print`*. On pourra ainsi parcourir les lignes et les colonnes pour représenter le caractère* `"X"` *ou* `" "`.

Exercice 2 : Suite de Fibonacci

La suite de Fibonacci est une suite mathématique définie par :
$F_0 = 0$; $F_1 = 1$; $F_n = F_{n-1} + F_{n-2}$ pour $n > 1$

1) Ecrire un programme qui affiche les k premiers éléments de la suite de Fibonacci où k est un entier demandé à l'utilisateur.
2) Ecrire un programme qui affiche la suite de Fibonacci jusqu'à ce qu'on trouve un terme qui dépasse 3000.

Exercice 3 : Travail autour d'un tableau

Dans cet exercice vous devrez programmer plusieurs fonctions et procédures. A chaque question, en plus de la fonction, vous devrez écrire les quelques lignes de programme principal qui permettent de tester votre fonction.

1) Ecrire une procédure `affiche_tableau` qui prend en argument un tableau et affiche son contenu sous la forme ci-contre.
2) Ecrire une fonction `demande_tableau` qui demande à l'utilisateur des nombres et les stocke au fur et à mesure dans un tableau. La saisie s'arrête lorsque aucun nombre n'est rentré (la saisie vaut ""). A la fin la fonction renvoi le tableau rempli.

```
t = [45, 6, 57, 82]
affiche_tableau(t)
```
Affiche :
```
Indice 0 : 45
Indice 1 : 6
Indice 2 : 57
Indice 3 : 82
```

Aide : Comme on ne connaît pas à l'avance le nombre d'éléments du tableau, on crée au départ un tableau vide puis on le rempli avec la méthode `append`*.*

3) Ecrire une fonction `tableau_hasard` qui prend en argument deux entiers `n` et `k` et qui renvoi un tableau de `n` entiers pris au hasard entre 0 et `k`.
4) Ecrire une procédure `multiplie_par_k` qui prend en argument un tableau d'entiers et un entier k et qui multiplie toutes les cases du tableau par k. Le tableau `[3, -2, 5, 8]` avec k = 3 donnera `[9, -6, 15, 24]`.
5) Ecrire une fonction `recherche_valeur` qui prend en argument un tableau d'entier et un entier `val` et qui renvoi un booléen indiquant si la valeur `val` se trouve dans une des cases du tableau. `recherche_valeur([4, 0, 7], 5)` donnera `False` ; tandis que `recherche_valeur([2, 5, 14, 11, 13], 11)` donnera `True`.

Exercice 4 : Trois fois … pas forcément de suite

Ecrire un programme qui demande des nombres entre 0 et 99 à l'utilisateur et qui s'arrête dès qu'il a entré trois fois le même nombre. Par exemple : 1 2 3 1 7 4 23 4 23 0 36 12 23 (23 apparaît trois fois).
Aide : Vous pouvez utiliser un tableau d'entier de taille 100 qui mémorisera combien de fois chaque nombre a été entré.

Exercice 5 : Il me les faut tous !

Ecrire un programme qui demande des nombres entre 0 et 9 à l'utilisateur et qui s'arrête quand il a tapé au moins une fois chacun. Il faudra probablement utiliser un tableau de taille 10, et une variable qui compte le nombre d'entiers différents que l'utilisateur a déjà tapé. Par exemple, s'il tape 1 2 3 1 0 4 7 4 7 6 9 5 7, cette variable contiendra 9.

Exercice 6 : Pendu simplifié

Le but de cet exercice est de réaliser un programme permettant de jouer au jeu du pendu.

1) Version sans décompte des erreurs

Le programme aura en mémoire un mot (toujours le même – choisissez celui que vous voulez). A chaque tour, il demandera à l'utilisateur de proposer un caractère, et affichera le mot en mémoire en masquant les lettres qui n'ont pas encore été proposées par l'utilisateur. Le programme s'arrêtera lorsque toutes les lettres du mot en mémoire auront été proposées.

2) Version avec plusieurs mots

Modifier le programme précédent pour faire en sorte que le mot à deviner soit tiré au hasard dans une liste de mots prédéfinie.

3) Version avec décompte des erreurs

Modifier encore le programme pour que celui-ci affiche le nombre d'erreurs. Modifier encore le programme pour que le jeu s'arrête sur une défaite si l'utilisateur a fait trop d'erreurs.

4) Mémorisation des lettres proposées

On ajoute au programme précédent la mémorisation des lettres déjà proposées. Si l'utilisateur propose une lettre qu'il a déjà donné, le programme la refuse avec un message explicite et redemande une lettre sans décompter d'erreur.

5) Prise en compte des minuscules

Faire en sorte que le programme prenne aussi bien des lettres majuscules que minuscule en entrée (un « e » ou un « E » permettent de deviner la lettre E).

Aide : Voir la méthode `upper()` *du type* `str` *(utilisez l'aide de python pour avoir la description de la fonction et faites des essais dans la console comme* `"aBcD".upper()`*).*

Si on le souhaite (mais ce n'est pas le but de cet exercice), on pourra après chaque erreur représenter un « pendu » de plus en plus avancé.

1NSI	**Langages et programmation** Bases de la programmation	Exercices **Consolidation des acquis**	**CORRIGE**

Exercice 1 :

```
n = int(input("Donnez une valeur entière impaire : "))
for y in range(n):
    for x in range(n):   # représentation d'une ligne
        if (x == y) or (x == n-y-1):
            print("X", end="")
        else :
            print(" ", end="")
    print()      # Saut de ligne après l'affichage d'une ligne
```

Exercice 2 :

1)

```
k = int(input("Nombre d'éléments à représenter ?
"))
F2 = 0
F1 = 1
if k >= 0:
    print("Terme de rang 0 :", F2)
    if k >= 1:
        print("Terme de rang 1 :", F1)
        for i in range(2, k):
            Fn = F1 + F2
            print("Terme de rang", i, " : ", Fn)
            F2 = F1
            F1 = Fn
```

2)

```
F2 = 0
F1 = 1
Fn = 0  # Valeur inférieure à 3000 pour entrer
        # dans la boucle while
print("Terme de rang 0 :", F2)
print("Terme de rang 1 :", F1)
i = 2
while (Fn < 3000):
    Fn = F1 + F2
    print("Terme de rang", i, " : ", Fn)
    F2 = F1
    F1 = Fn
    i = i + 1
```

Exercice 3 :

1)

```
def affiche_tableau(tab):
    for i in range(len(tab)):
        print("Indice", i, ":", tab[i])

t = [45, 6, 57, 82]
affiche_tableau(t)
```

2)

```
def demande_tableau():
    tab = []
    saisie = input("Entrez un nombre ou rien
pour arrêter : ")
    while (saisie != ""):
        tab.append(int(saisie))
        saisie = input("Entrez un nombre ou rien
pour arrêter : ")
    return tab

print(demande_tableau())
```

3)

```
import random
def tableau_hasard(n, k):
    tab = [0]*n   # Crée un tableau de n cases
    for i in range(n):
        tab[i] = random.randint(0, k)
    return tab

print(tableau_hasard(15, 10))
```

4)

```
def multiplie_par_k(tab, k):
    for i in range(len(tab)):
        tab[i] = tab[i] * k

t = tableau_hasard(15, 10)
print(t)
multiplie_par_k(t, 3)
print(t)
```

5)

```
def recherche_valeur(tab, val):
    for element in tab:
        if element == val: # Dès qu'on trouve
            return True    # on renvoi True
    return False  # Sinon à la fin on renvoi False

t = [12, 35, 8, -3, 0, 9]
print(recherche_valeur(t, 77))
print(recherche_valeur(t, -3))
```

Exercice 4 :

```
nombre_de_fois = [0]*100
termine = False
while (not termine):
    n = int(input("Entrez un nombre de 0 à 99 : "))
    nombre_de_fois[n] += 1
    if nombre_de_fois[n] == 3:
        termine = True
print("Vous avez entré trois fois le nombre", n)
```

Exercice 5 :

```
nb_entiers_differents = 0
nombre_present = [False]*10
while nb_entiers_differents != 10:
    n = int(input("Entrez un nombre de 0 à 9 : "))
    if nombre_present[n] == False:
        nombre_present[n] = True
        nb_entiers_differents += 1
print("Tous les nombres ont été saisis au moins une fois")
```

Exercice 6 :

1)

```
mot_mystere = "ENTONNOIR"
mot_devine = ["-"]*len(mot_mystere)
gagne = False  #  Drapeau pour indiquer si on a gagné
while not gagne:
    # Affichage du mot deviné
    for lettre in mot_devine:
        print(lettre, end="")
    print()   # Passe à la ligne après avoir écrit le mot

    # Teste la proposition et l'écrit éventuellement dans le mot deviné
    proposition = input("Quelle lettre proposez-vous ? ")
    for i in range(len(mot_mystere)):   # On parcours le mot
        if proposition == mot_mystere[i]: # et on teste avec la proposition
            mot_devine[i] = proposition   # si c'est correct, on modifie le mot deviné

    # Teste si on a gagné
    gagne = True
    for i in range(len(mot_mystere)):
        if mot_devine[i] != mot_mystere[i]:
            gagne = False  # Il suffit d'une erreur pour
                           # qu'on ait pas gagné

print("Bravo, vous avez gagné !, c'était bien ", mot_mystere)
```

2) Il suffit de changer les premières lignes en :

```
import random

liste_mots = ["ENTONNOIR", "PYTHON", "ORDINATEUR", "SOLEIL", "LYCEE",
              "INFORMATIQUE", "DUALITE", "PARADIGME"]
mot_mystere = liste_mots[random.randint(0,len(liste_mots)-1)]
```

3)

```
import random

liste_mots = ["ENTONNOIR", "PYTHON", "ORDINATEUR", "SOLEIL", "LYCEE",
              "INFORMATIQUE", "DUALITE", "PARADIGME"]
NB_ERREURS_MAX = 7
mot_mystere = liste_mots[random.randint(0,len(liste_mots)-1)]
mot_devine = ["-"]*len(mot_mystere)
nb_erreurs = 0
gagne = False  # Drapeau pour indiquer si on a gagné
while (not gagne) and (nb_erreurs < NB_ERREURS_MAX):
    # Affichage du mot deviné
    for lettre in mot_devine:
```

```
        print(lettre, end="")
    print("     Nombre d'erreurs :", nb_erreurs,"/", NB_ERREURS_MAX)

    proposition = input("Quelle lettre proposez-vous ? ")

    # Teste la proposition et l'écrit éventuellement dans le mot deviné
    lettre_correcte = False
    for i in range(len(mot_mystere)):   # On parcours le mot
        if proposition == mot_mystere[i]: # et on teste avec la proposition
            mot_devine[i] = proposition   # si c'est correct, on modifie le mot deviné
            lettre_correcte = True        # et on note que la lettre est bonne
    if not lettre_correcte:    # Si la lettre n'est pas bonne
        nb_erreurs += 1        # C'est une erreur de plus

    # Teste si on a gagné
    gagne = True
    for i in range(len(mot_mystere)):
        if mot_devine[i] != mot_mystere[i]:
            gagne = False  # Il suffit d'une erreur pour
                           # qu'on ait pas gagné

if nb_erreurs == NB_ERREURS_MAX:
    print("Perdu ! Le mot à deviner était", mot_mystere)
else:
    print("Bravo, vous avez gagné !, c'était bien", mot_mystere)
```

4)

```
import random

liste_mots = ["ENTONNOIR", "PYTHON", "ORDINATEUR", "SOLEIL", "LYCEE",
              "INFORMATIQUE", "DUALITE", "PARADIGME"]
NB_ERREURS_MAX = 7
mot_mystere = liste_mots[random.randint(0,len(liste_mots)-1)]
mot_devine = ["-"]*len(mot_mystere)
lettres_proposees = []
nb_erreurs = 0
gagne = False  # Drapeau pour indiquer si on a gagné
while (not gagne) and (nb_erreurs < NB_ERREURS_MAX):
    # Affichage du mot deviné
    for lettre in mot_devine:
        print(lettre, end="")
    print("     Nombre d'erreurs :", nb_erreurs,"/", NB_ERREURS_MAX)

    proposition = input("Quelle lettre proposez-vous ? ")

    if proposition in lettres_proposees:
        print("Lettre déjà proposée, essayez autre chose !")
    else:
        lettres_proposees.append(proposition)   # rajoute la proposition à la liste

        # Teste la proposition et l'écrit éventuellement dans le mot deviné
        lettre_correcte = False
        for i in range(len(mot_mystere)):   # On parcours le mot
            if proposition == mot_mystere[i]: # et on teste avec la proposition
                mot_devine[i] = proposition  # si c'est correct, on modifie le mot deviné
                lettre_correcte = True        # et on note que la lettre est bonne
        if not lettre_correcte:    # Si la lettre n'est pas bonne
            nb_erreurs += 1        # C'est une erreur de plus

        # Teste si on a gagné
        gagne = True
        for i in range(len(mot_mystere)):
            if mot_devine[i] != mot_mystere[i]:
                gagne = False  # Il suffit d'une erreur pour
                               # qu'on ait pas gagné

if nb_erreurs == NB_ERREURS_MAX:
    print("Perdu ! Le mot à deviner était", mot_mystere)
else:
    print("Bravo, vous avez gagné !, c'était bien", mot_mystere)
```

5) Il suffit de rajouter `proposition = proposition.upper()` juste après la saisie de la proposition pour transformer la saisie en majuscule au cas où l'utilisateur aurait mis une minuscule.