Démonstration # 4 et Laboratoire # 4

Exercice 1

Nous voulons compter le nombre de fois qu'il faut tirer à pile-ou-face pour que 5000 des tirs aient donné pile. En principe le nombre de tirs devrait être proche de 10000 si **`random()`** est aléatoire.
Codez un programme qui effectue cette simulation dans un premier temps avec une seule boucle «tant-que» et ensuite avec une boucle «tant-que» imbriquée dans une boucle «for». Notez que pour tirer à pile-ou-face on peut se servir du **`test random() < 0.5`**. Un résultat **`True`** signifie "pile", et un résultat **`False`** signifie "face".

Exercice 2

Codez un programme qui pour une variable **`n`**, un entier >= 0, calcule et affiche la somme des **`n`** premiers nombres impairs (par exemple, pour **`n=4 : 1 + 3 + 5 + 7`**). Quel type de boucle est approprié ici : «tant-que», «répéter» ou «for» ?

Exercice 3

Dans un programme nous voulons obtenir de l'usager un nombre positif. Si l'usager entre un nombre <= 0 nous voulons continuer à lui demander un nombre positif jusqu'à ce qu'il entre un nombre positif. Codez cette partie de programme.
Quel type de boucle est approprié ici : «tant-que», «répéter» ou «for» ?

Exercice 4

Utilisez la réponse de l'exercice précédent pour coder un programme qui affichera la somme de tous les nombres positifs qui ont été entrés par l'usager. L'usager indiquera qu'il a fini d'entrer des nombres en cliquant sur le bouton « Cancel » de la boîte de dialogue de la fonction **`prompt`**. Le programme doit indiquer l'index du nombre qui est demandé (i.e. le premier, le second, etc) avec le message « Entrez le nombre positif #1 », « Entrez le nombre positif #2 », etc Utilisez l'énoncé **`break`** judicieusement pour traiter le bouton « Cancel ».

Exercice 5

Modifiez le programme de l'exercice précédent pour qu'il affiche le plus petit nombre, le plus grand nombre et la moyenne des nombres entrés par l'usager. Assurez-vous que le programme fonctionne correctement si l'usager clique sur « Cancel » tout de suite.

Exercice 6

Codez un programme qui, étant donné une variable **`n`** contenant un entier positif, va imprimer tous ses facteurs (les entiers >= 2 qui divisent exactement **`n`**).

Exercice 7

Qu'est-ce qui est imprimé par le programme suivant?

```
i = 2
while i <= 31:
    print(i)
    i += 1 + (i>2) + 2*(i%6==1) + 4*(i==23)
```

Exercices du laboratoire #4

Exercice 1

Concevez une abstraction procédurale pour un lancer aléatoire de pièce de monnaie. Quelle est la tâche de cette abstraction procédurale? Quels sont les paramètres formels? Quel est le type du résultat? Codez votre abstraction procédurale comme une déclaration de fonction.

Exercice 2

Concevez une abstraction procédurale pour un lancer de dé à six faces. Quelle est la tâche de cette abstraction procédurale? Quels sont les paramètres formels? Quel est le type du résultat? Codez votre abstraction procédurale comme une déclaration de fonction.

Exercice 3

Concevez une abstraction procédurale pour un événement aléatoire discret pour lequel il y a N résultats possibles qui ont la même probabilité. Quelle est la tâche de cette abstraction procédurale? Quels sont les paramètres formels? Quel est le type de résultat? Codez votre abstraction procédurale comme une déclaration de fonction nommée **`evenementAleatoire`**.

Exercice 4

Concevez et codez la fonction **`lancerNDes`** qui simule le lancer de **`N`** dés à six faces où on s'intéresse à la somme des dés. Utilisez cette fonction pour coder la fonction **`lancer2Des`** qui simule le lancer de 2 dés à six faces. Est-ce que la définition suivante est équivalente?

```
def lancer2Des():
    return evenementAleatoire(11) + 2
```

Exercice 5

Utilisez la fonction **`evenementAleatoire`** pour coder à nouveau les deux premiers exercices et aussi une abstraction procédurale qui simule un lancer au jeu de roulette (37 résultats possibles, 1 à 36 et le 0).

Exercice 6

À la séance de démonstration #2, on a donné la formule pour calculer le jour de la semaine à partir de la date. Transformez cette formule en abstraction procédurale. Quels sont les paramètres formels et le type du résultat ? Codez des tests unitaires pour la fonction **`jourDeLaSemaine`** dans une procédure **`testJourDeLaSemaine`**. Vérifiez que ça fonctionne dans codeBoot avec une exécution pas-à-pas. Introduisez une erreur dans la fonction jourDeLaSemaine pour voir ce qui se passe. Introduisez une erreur dans la fonction **`testJourDeLaSemaine`** pour voir ce qui se passe.

Exercice 7

Concevez une abstraction procédurale pour la conversion d'une température Celsius à Fahrenheit et une abstraction procédurale pour la conversion d'une température Fahrenheit à Celsius. Codez vos abstractions procédurales comme des déclarations de fonctions, et codez des tests unitaires. Discutez de la validité de ce test unitaire :

```
for t in range(-100, 101):
    assert t == farenheitACelcius(celciusAFarenheit(t))
    assert t == celciusAFarenheit(farenheitACelcius(t))
```

Exercice 8

Faire l'exécution pas-à-pas de ce programme. Quel est le résultat? Que se passe t'il si on enlève l'énoncé **`global somme, compteur`** ? Pourquoi un énoncé **`global somme`**, **`compteur`** n'est pas nécessaire dans la fonction **`moyenne`** ?

```
somme = 0
compteur = 0

def ajouter(x):
    global somme, compteur
    somme = somme + x
    compteur = compteur + 1

def moyenne():
    return somme / compteur

ajouter(100)
ajouter(120)
ajouter(200)

print(moyenne())
```