IFT-1015

Démo 5

Exercice 1

Tâche : voir le commentaire ci-dessous
Paramètres : aucun
Résultat : booléen
Codage :

```
# La fonction lancerMonnaie prends aucun paramètre et retourne un
# booléen aléatoire, avec une probabilité égale de retourner True
# et False.  Cela simule le lancer d'une pièce de monnaie.

def lancerMonnaie():
    return random() < 0.5
```

Exercice 2

Tâche : voir le commentaire ci-dessous
Paramètres : aucun
Résultat : entier de 1 à 6
Codage :

```
# La fonction lancerDe prends aucun paramètre et retourne un
# entier de 1 à 6, avec une probabilité égale de retourner chacun
# de ces nombres.  Cela simule le lancer d'un dé à 6 faces.

def lancerDe():
    return math.floor(6 * random()) + 1
```

Exercice 3

Tâche : voir le commentaire ci-dessous
Paramètres : entier n plus grand ou égal à 1
Résultat : entier de 0 à n-1
Codage :

```
# La fonction evenementAleatoire prends un paramètre n qui est un
# entier plus grand ou égal à 1 et retourne un entier de 0 à n-1,
# avec une probabilité égale de retourner chacun de ces nombres.
# Cela simule un événement aléatoire discret avec une distribution
# de probabilité uniforme.

def evenementAleatoire(n):
    return math.floor(n * random())
```

Exercice 4

Tâche : voir le commentaire ci-dessous
Paramètres : entier n plus grand ou égal à 0
Résultat : un entier de n à 6*n
Codage :

```
# La fonction lancerNDes prends un paramètre n qui est un entier
# plus grand ou égal à 0 et retourne un entier de n à 6*n, qui
# correspond a la somme des dés d'un lancer aléatoire de n dés
# à 6 faces.

def lancerNDes(n):
    somme = 0
    for _ in range(n):
        somme += lancerDe()
    return somme

# La fonction lancer2Des prends aucun paramètre et retourne un
# entier de 2 à 12, qui correspond à la somme de 2 dés à six faces
# lancés aléatoirement.

def lancer2Des():
    return lancerNDes(2)
```

La fonction lancer2Des ci-dessous n'est pas équivalente car la probabilité d'obtenir un certain résultat est égale pour toutes les valeurs de 2 à 12, mais la fonction lancer2Des ci-dessus a des probabilité non égales (par exemple le 7 a une probabilité 1/6 et le 12 a une probabilité 1/36).

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

Exercice 5

```
# La fonction lancerMonnaie prends aucun paramètre et retourne un
# booléen aléatoire, avec une probabilité égale de retourner True
# et False.  Cela simule le lancer d'une pièce de monnaie.

def lancerMonnaie():
    return evenementAleatoire(2) == 0

# La fonction lancerDe prends aucun paramètre et retourne un
# entier de 1 à 6, avec une probabilité égale de retourner chacun
# de ces nombres.  Cela simule le lancer d'un dé à 6 faces.

def lancerDe():
    return evenementAleatoire(6) + 1

# La fonction lancerRoulette prends aucun paramètre et retourne un
# entier de 0 à 36, avec une probabilité égale de retourner chacun
# de ces nombres.  Cela simule le lancer de bille au jeu de
# roulette francaise.

def lancerRoulette():
    return evenementAleatoire(37)
```

Exercice 6

```
# La fonction jourDeLaSemaine prends 3 paramètres et retourne un
# entier de 1 à 7 qui représente un jour de la semaine (avec
# 1=dimanche, 2=lundi, etc.)  Les 3 paramètres sont des entiers
# qui représentent une date.  Le premier paramètre indique l'année
# et doit être de 1900 à 2099.  Le deuxième paramètre indique le
# mois et doit être de 1 à 12.  Le troisième paramètre indique
# le quantième et doit être de 1 à 28, 29, 30 ou 31 ).
# (en fonction du mois et de l'année La valeur retournée indique
# sur quel jour de la semaine tombe la date indiquée en paramètre.

def jourDeLaSemaine(annee, mois, quant):

    # Calculer  l  ajustement  des années bisextiles

    nbMois = annee*12 + mois - 3 # mois écoulés, en partant de mars
    nbAns = nbMois // 12         # années écoulées
    nb4Ans = nbAns // 4          # périodes de 4 ans écoulées
    nb100Ans = nbAns // 100      # périodes de 100 ans écoulées
    nb400Ans = nbAns // 400      # périodes de 400 ans écoulées

    ajustBisext = (mois+9)//12*4 + nbAns + nb4Ans - nb100Ans +
    nb400Ans

    # Calculer le jour de la semaine (1=dimanche, 2=lundi, etc)

    return (23*mois//9 + ajustBisext + quant + 5) % 7 + 1
```

Exercice 7

```
def testJourDeLaSemaine():
    assert jourDeLaSemaine(2020, 9, 29) == 3  # 2020-09-29 -> mardi
    assert jourDeLaSemaine(2020, 9, 31) == 5  # 2020-09-31 -> jeudi
    assert jourDeLaSemaine(2020, 2, 29) == 7  # 2020-02-29 -> sam.
    assert jourDeLaSemaine(2020, 3,  1) == 1  # 2020-03-01 -> dim.
    assert jourDeLaSemaine(1900, 1,  1) == 2  # 1900-01-01 -> lundi
    assert jourDeLaSemaine(2099,12, 31) == 5  # 2099-12-31 -> jeudi
    assert jourDeLaSemaine(1961, 1,  1) == 1  # 1961-01-01 -> dim.

testJourDeLaSemaine()
```

Notez qu'on a testé non seulement des dates banales, mais aussi des dates pour lesquelles on obtient le plus petit et le plus grand résultat (1 et 7) et aussi des dates spéciales (comme les premier et dernier jours de l'année, pour la première et dernière année valide, et aussi le 29 février et 1 mars d'une année bisextile).

Exercice 8

```
# La fonction celciusAFarenheit prend un nombre en paramètre qui
# représente une température en degrés Celcius et retourne un
# nombre qui correspond à cette température en degrés Farenheit.

def celciusAFarenheit(tempC):
    return tempC * 9 / 5 + 32

# La fonction farenheitACelcius prend un nombre en paramètre qui
# représente une température en degrés Farenheit et retourne un
# nombre qui correspond à cette température en degrés Celcius.

def farenheitACelcius(tempF):
    return (tempF - 32) * 5 / 9

def testConversionTemperature():

    assert celciusAFarenheit(0)   == 32   # 0 C = 32 F
    assert celciusAFarenheit(100) == 212  # 100 C = 212 F
    assert celciusAFarenheit(-40) == -40  # -40 C = -40 F

    assert farenheitACelcius(32) == 0     # 32 F = 0 C
    assert farenheitACelcius(212) == 100  # 212 F = 100 C
    assert farenheitACelcius(-40) == -40  # -40 F = -40 C

testConversionTemperature()
```

Le test unitaire proposé est intéressant pour vérifier que les deux fonctions sont des inverses. Cependant, à cause des erreurs d'arrondi des calculs sur les flottants le test == va détecter des différences (minuscules) pour certaines valeurs de t.

Il est mieux de faire :

```python
def tresProche(x, y):
    return abs(x - y) < 1e-10

def testConversionTemperature():
    for t in range(-100, 101):
        assert tresProche(t, farenheitACelcius(celciusAFarenheit(t)))
        assert tresProche(t, celciusAFarenheit(farenheitACelcius(t)))
```

Les valeurs utilisées pour les tests unitaires ont été choisi avec une attention particulière pour ne pas causer d'erreurs d'arrondi donc l'utilisation de == est correct dans ce cas.

Exercice 9

Le programme affiche 140.0 (la moyenne de 100, 120 et 200).

Lorsqu'on retire l'énoncé `global` les variables `somme` et `compteur` sont des variables locales de la fonction `ajouter`. Cependant la variable `somme` n'a pas encore de valeur lorsque `somme + x` est évaluée, donc une erreur est signalée à l'exécution du corps de `ajouter`.

Un énoncé `global` n'est pas nécessaire dans la fonction `moyenne` car les variables `somme` et `compteur` ne seront pas traitées comme des variables locales. Il faudrait des déclarations de variables `somme = ...` et `compteur = ...` pour que ces variables soient considérées comme des variables locales (et qu'il n'y ait pas de `global somme, compteur` dans la déclaration de fonction).

L'énoncé `global` devient nécessaire si dans le corps d'une fonction on veut faire une affectation à une variable globale.