04-boucles-iteratives

November 29, 2023

Activité : boucles itératives

Le but de cette activité est de manipuler les boucles itératives.

Exercices

Exercice 1 : produit d'entiers

On considère l'algorithme suivant :

fonction fact(n) P $\leftarrow$ 1 pour k allant de 2 à n P $\leftarrow$ P $\times$ k fin pour renvoyer P fin fonction

1. Que renvoient les instructions `fact(1)`, `fact(5)` et `fact(n)` ?

Ecrire la réponse ici

2. Implémenter l'algorithme dans la cellule ci-dessous.

[ ]:
```
# Ecrire l'implémentation de l'algorithme ici
```

Les quatre cellules suivantes permettent de tester votre fonction. Elles doivent renvoyer `True` lors de leur exécution.

[ ]:
```
fact(1) == 1
```

[ ]:
```
fact(3) == 6
```

[ ]:
```
fact(4) == 24
```

[ ]:
```
fact(6) == 720
```

Exercice 2 : placement bancaire

1. On place une somme de 2000 euros sur un compte remunéré au taux annuel de 2%. Compléter la fonction suivante de telle sorte qu'elle renvoie la somme disponible sur le compte au bout de 20 ans. On arrondira le résultat au centime d'euro près.

[ ]:
```
def placement():
    somme = ...
    for annee in range(...):
        somme = somme * ...
    return round(...,...)
```

La cellule suivante permet de tester votre fonction. Elle doit renvoyer `True` lors de son exécution.

[ ]:
```
placement() == 2971.89
```

2. Modifier la fonction précédente en une fonction `placementBancaire` de telle sorte qu'elle puisse fonctionner avec n'importe quel placement initial, n'importe quel taux et sur n'importe quelle durée.

[ ]:
```
# Ecrire ici le code de la fonction placementBancaire
```

Les trois cellules suivantes permettent de tester votre fonction. Elles doivent renvoyer `True` lors de leur exécution.

[ ]:
```
placementBancaire(2000,2,20) == 2971.89
```

[ ]:
```
placementBancaire(4000,2,15) == 5383.47
```

[ ]:
```
placementBancaire(3000,1.75,10) == 3568.33
```

Exercice 3 : jeu de rôle

Dans un jeu, un joueur dispose de `250` points de vie. Il subit cinq attaques successives dont les valeurs sont des nombres entiers aléatoires entre `10` et `40`.

1. Compléter le code de la fonction ci-dessous afin qu'elle renvoie le nombre de points de vie du joueur à l'issue de ces cinq attaques.

[ ]:
```
# Il existe d'innombrables "bibliothèques" Python qui sont mises à disposition␣
 ↪par leurs auteurs.
# Une bibliothèque Python une collection de modules associés qui contient des␣
 ↪fonctions
# réutilisables dans vos programmes. Pour accéder à l'intégralité des fonctions␣
 ↪d'une bibliothèque, on
# utilise la syntaxe (en début de programme): "from <nom_bibliothèque> import␣
 ↪*" (le "*" signifiant "tout").
# On peut alors appeler directement les fonctions du module si on en connait le␣
 ↪nom.

# C'est ce qu'on fait ici avec le module "random" (aléatoire) et la fonction␣
 ↪"randint(a, b)" qui fournit
# un entier aléatoire compris entre a et b (inclus)

from random import * # module aléatoire que l'on importe

def attaquesAleatoires5():
    pointsVie = ...
    for i in range(5):
        attaque = randint(10, 40) # renvoie un entier au hasard compris entre␣
 ↪10 et 40 inclus
        pointsVie = ...
    return ...
```

2. Le joueur dispose à présent d'une défense de `25`. Lorsqu'il subit une attaque de valeur `attaque`, on applique la règle suivante :

- si la valeur de l'attaque est strictement supérieure à celle de la défense, le joueur perd `attaque - 25` points de vie ;
- si la valeur de l'attaque est inférieure ou égale à celle de la défense, le joueur ne perd aucun point de vie.

Compléter la fonction suivante afin qu'elle renvoie le nombre de points de vie du joueur à l'issue des cinq attaques précédentes.

[ ]:
```
from ...

def attaquesAleatoires5Defense():
    pointsVie = ...
    for i in range(5):
        attaque = ...
        if ... :
            pointsVie = ...
    return ...
```

3. On modifie la fonction précédente en une fonction `pointsVieAttaquesAleatoiresMultiplesDefense` pour qu'elle fonctionne avec n'importe quel nombre de points de vie, n'importe quelle défense et n'importe quel nombre d'attaques successives prenant des valeurs entières aléatoires entre deux nombres entiers prédéfinis en respectant la règle suivante :

- si la valeur de l'attaque est strictement supérieure à celle de la défense, le joueur perd `attaque - defense` points de vie ;
- si la valeur de l'attaque est inférieure ou égale à celle de la défense, le joueur ne perd aucun point de vie.
- si le nombre de points de vie à l'issue de toutes les attaques est strictement négatif, on le ramène à 0.

Ecrire ci-dessous le code de cette fonction où les paramètres sont définis de la façon suivante : - `pointsVie` est le nombre de points de vie initial du joueur ; - `defense` est la défense du joueur ; - `nbAttaques` est le nombre d'attaques successives subies par le joueur ; - `valMinAttaque` est la valeur minimale que peut prendre une attaque ; - `valMaxAttaque` est la valeur maximale que peut prendre une attaque.

Tous les paramètres de la fonction sont des nombres entiers strictement positifs.

[ ]:
```
from ...

def␣
 ↪pointsVieAttaquesAleatoiresMultiplesDefense(pointsVie,defense,nbAttaques,valMinAttaque,valM
 ↪
    # code à compléter
```

Exercice 4 : lancer d'un dé

On lance 50 fois un dé équilibré à six faces numérotées de 1 à 6 et on regarde le nombre de fois où on a obtenu 6.

Compléter la fonction suivante afin de simuler cette expérience.

[ ]:
```
from random ...

def lancers50De6():
    nb6 = ... # initialisation du compteur donnant le nombre de 6 obtenu
    for lancer in range(...):
        face = randint(...,...)
        if face ... :
            nb6 = nb6 + ...
    return ...
```

Exercice 5 : pile ou face

Compléter la fonction suivante simulant le jeu du Pile ou Face avec une pièce équilibrée et renvoyant pour un nombre de lancers `nbLancers` donné la fréquence d'apparition du Pile.

[ ]:
```
from ...

def freqPile(nbLancers):
    nbPile = ...
    for i in ... :
        piece = randint(0,...)
        if piece ... :
            nbPile = ...
    return ...
```

Pour aller plus loin

Exercice 1 : flux de populations

On considère un flux de populations entre trois zones géographiques A, B et C sous les conditions suivantes : - La population totale des trois zones reste constante au cours du temps. - Chaque année, la zone A perd 20% de sa population au profit de la zone B et 10% de sa population au profit de la zone C. - Chaque année, la zone B perd 15% de sa population au profit de la zone A et 5% de sa population au profit de la zone C. - Chaque année, la zone C perd 25% de sa population au profit de la zone A et 10% de sa population au profit de la zone B.

Le tableau suivant donne les populations initiales des trois zones :

Zone A	Zone B	Zone C
100 000	80 000	120 000

Compléter la fonction ci-dessous de telle sorte qu'elle renvoie les populations des trois zones A, B et C dans cet ordre au bout de `nbAnnees` années.

[ ]:
```
def evolutionPop(nbAnnees):
    assert ... # test d'assertion sur nbAnnees
    popA = ... # population initiale de la zone A
    popB = ... # population initiale de la zone B
```

```
    popC = ... # population initiale de la zone C
    for annee in ...:
        popA_temp = ... # calcul de la nouvelle population de la zone A
        popB_temp = ...
        popC_temp = ...
        popA = ...
        popB = ...
        popC = ...
    return ...
```

Les trois cellules suivantes permettent de tester votre fonction. Elles doivent renvoyer `True` lors de leur exécution.

[ ]:
```
evolutionPop(0) == (100000,80000,120000)
```

[ ]:
```
evolutionPop(3) == (116270.0, 117460.0, 66270.0)
```

[ ]:
```
evolutionPop(5) == (114624.07499999998, 128251.85000000002, 57124.075000000004)
```

Exercice 2 : lancer de deux dés

On lance 25 fois deux dés équilibrés à six faces numérotées de 1 à 6 et on regarde le nombre de fois où on a obtenu deux faces identiques.

En s'aidant de la fonction écrite à l'exercice précédent, écrire une fonction `lancers25DeuxDesFacesIdentiques` permettant de simuler cette expérience.

[ ]:
```
# Ecrire ici le code de la fonction lancers25DeuxDesFacesIdentiques
```

Exercice 3 : méthode de Monte-Carlo

On considère le disque de centre O et de rayon 1, ainsi que le carré circonscrit à ce disque.

On choisit au hasard n points dans le carré et on regarde la fréquence d'apparition de ces points à l'intérieur du disque.

1. En s'aidant des fonctions écrites ci-dessus, écrire une fonction `monteCarlo` permettant de simuler cette expérience.
2. Que peut-on dire de cette fréquence lorsque le nombre de points choisis est très grand ?

[ ]:
```
from random import uniform

def monteCarlo(nbPoints):
    # code à compléter
```