# 05_boucles_conditionnelles

November 29, 2023

Activité : boucles conditionnelles

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

Exercices

Exercice 1 : mystère…

On considère l'algorithme suivant :

fonction mystere() somme ← 2000 nbAnnees ← 0 tant que somme $<$ 3000 somme ← somme $\times 1,02$ nbAnnees ← nbAnnees + 1 fin tant que renvoyer nbAnnees et une valeur approchée de somme à $10^{-2}$ près fin fonction

1. Que renvoie la fonction `mystere` ? En donner une interprétation concrète.

Ecrire la réponse ici

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

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

Exercice 2 : placement bancaire

On dépose une somme d'argent sur un compte rémunéré et on désire savoir combien d'années on doit attendre avant que la somme disponible sur ce compte ait atteint une valeur particulière. On veut aussi connaître la somme disponible réellement à cet instant, au centime d'euro près.

En s'aidant de la fonction écrite à l'exercice précédent, compléter la fonction `placementBancaireSeuil` pour qu'elle renvoie la réponse à ce problème.

[ ]:
```
def placementBancaireSeuil(placement,taux,sommeCible):
    # à compléter, éventuellement sur plusieurs lignes
```

Les trois cellules suivantes sont un jeu de tests pour tester votre fonction. Chacune doit renvoyer `True` lors de l'exécution de la cellule.

[ ]:
```
placementBancaireSeuil(2000,2,3000) == (21, 3031.33)
```

[ ]:
```
placementBancaireSeuil(5000,1.5,8000) == (32, 8051.62)
```

[ ]:
```
placementBancaireSeuil(2000,5.25,5000) == (18, 5023.75)
```

Exercice 3 : culture de bactéries

A l'instant $t = 0$, on met en culture une population de bactéries qui augmente de $10,5\%$ toutes les heures.

En s'aidant des fonctions écrites aux exercices 1 et 2, écrire une fonction `attenteSeuilBacteries` prenant en paramètres deux entiers `popInitiale` et `popFinale` correspondant respectivement à la taille initiale de la population mise en culture et à la taille de la population que l'on veut atteindre, et renvoyant le nombre d'heures à attendre pour atteindre cette taille.

[ ]:
```
# Ecrire la fonction attenteSeuilBacteries ici
```

Les trois cellules suivantes sont un jeu de tests pour tester votre fonction. Chacune doit renvoyer `True` lors de l'exécution de la cellule.

[ ]:
```
attenteSeuilBacteries(100000,200000) == 7
```

[ ]:
```
attenteSeuilBacteries(100000,500000) == 17
```

[ ]:
```
attenteSeuilBacteries(100000,1000000) == 24
```

Exercice 4 : jeu de rôle

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

Lorsque le nombre de points de vie du joueur devient négatif ou nul, on dit que le joueur s'évanouit.

1. Compléter le code de la fonction ci-dessous afin qu'elle renvoie le nombre d'attaques que le joueur peut subir jusqu'à ce qu'il s'évanouisse.

[ ]:
```
from ...

def nbAttaquesAleatoires():
    pointsVie = ...
    nbAttaques = ...
    while ... :
        attaque = randint(10,40)
        pointsVie = ...
        nbAttaques = ...
    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 d'attaques que le joueur peut subir dans ces conditions jusqu'à ce qu'il s'évanouisse.

[ ]:
```
from ...

def nbAttaquesAleatoiresDefense():
    pointsVie = ...
    nbAttaques = ...
    while ... :
        attaque = randint(10,40)
        if ... :
            pointsVie = ...
        nbAttaques = ...
    return ...
```

3. On modifie la fonction précédente en une fonction `nbAttaquesAleatoiresPointsVieDefense` pour qu'elle fonctionne avec n'importe quel nombre de points de vie, n'importe quelle défense et des 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.

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 ; - `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␣
 ↪nbAttaquesAleatoiresPointsVieDefense(pointsVie,defense,valMinAttaque,valMaxAttaque):
 ↪
    # code à compléter
```

Exercice 5 : lancers d'un dé

On lance plusieurs fois un dé équilibré à six faces numérotées de 1 à 6 et on regarde le nombre de lancers qu'il a fallu effectuer pour obtenir le 6 pour la première fois.

Compléter la fonction suivante afin de simuler cette expérience.

[ ]:
```
from ...

def nbLancersDe6():
    nbLancers = ...
    face = 0
    while ... :
        nbLancers = ...
```

```
        face = ...
    return nbLancers
```

A quoi sert l'instruction `face = 0` avant la boucle `while` ?

Exercice 6 : lancers de deux dés (partie 1)

On lance plusieurs fois deux dés équilibrés à six faces numérotées de 1 à 6 et on regarde le nombre de lancers qu'il a fallu effectuer pour obtenir deux faces identiques pour la première fois.

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

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

Pour aller plus loin

Exercice 1 : lancers de deux dés (partie 2)

On lance plusieurs fois deux dés équilibrés à six faces numérotées de 1 à 6 et on regarde le nombre de lancers qu'il a fallu effectuer pour obtenir : 1. un double-six pour la première fois (expérience 1); 2. un double-quatre pour la première fois (expérience 2).

En s'aidant de la fonction écrite à l'exercice précédent, écrire deux fonctions `nbLancersDeuxDesDoubleSix` et `nbLancersDeuxDesDoubleQuatre` permettant de simuler ces deux expériences.

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

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

Exercice 2 : lancers de deux dés (partie 3)

On lance plusieurs fois deux dés équilibrés à six faces numérotées de 1 à 6 et on regarde le nombre de lancers qu'il a fallu effectuer pour obtenir un double-six pour la seconde fois.

En s'aidant de la fonction écrite à l'exercice précédent, compléter la fonction `nbLancersDeuxDesDoubleSixDeuxFois` ci-dessous permettant de simuler cette expérience.

[ ]:
```
from ...

def nbLancersDeuxDesDoubleSixDeuxFois():
    nbLancers = ... # compteur pour le nombre de lancers effectués
    nbDoubleSix = ... # compteur pour le nombre de double-six obtenus
    while nbDoubleSix ... :
        nbLancers = ...
        faceDe1 = ...
        faceDe2 = ...
        if ... :
            nbDoubleSix = ...
    return ...
```

Exercice 3 : tirages de cartes

On considère le jeu suivant : - On choisit au hasard une carte dans un jeu classique de 32 cartes. - Si la carte tirée est un as, on s'arrête ; sinon, on la remet dans le paquet et on recommence.

On s'intéresse au nombre de tirages que l'on doit effectuer pour obtenir un as pour la première fois.

Compléter la fonction `nbTiragesAs` ci-dessous permettant de simuler cette expérience.

[ ]:
```
from ...

def nbTiragesAs():
    nbTirages = ...
    carte = ...
    while carte > 4: # les cartes du jeu sont assimilées à des nombres entiers..
 ↪.
        nbTirages = ...
        carte = randint(...,...)
    return ...
```