

Raffinages

Corrigé

Objectifs

- Comprendre les raffinages
- Savoir produire des raffinages
- Mettre ne pratique la méthode des raffinage
- Écrire des programmes plus ambitieux

Exercice 1 : Des raffinages vers le programme	1
Exercice 2 : Retrouver un raffinement	2

Exercice 1 : Des raffinages vers le programme

Produire un programme à partir de ses raffinages donnés ci-après. On écrira le programme dans le fichier `gerer_compteur.py`.

```
1  R0 : Gérer un compteur avec un menu
2
3  R1 : Comment « Gérer un compteur avec un menu »
4      compteur = 0
5      choix = '1'
6      while choix != '0':
7          Afficher le compteur                (compteur: in)
8          Afficher le menu
9          Demander le choix de l'utilisateur   (choix: out)
10         Traiter le choix de l'utilisateur    (compteur: in out ; choix: in)
11
12  R2 : Comment « Afficher le menu »
13      print("1. Incrémenter")
14      print("2. Décrémenter")
15      print("0. Quitter")
16
17  R2 : Comment « Afficher le compteur »
18      print()
19      print("Compteur :", compteur)
20      print()
21
22  R2 : Comment « Demander le choix de l'utilisateur »
23      choix = input("Votre choix : ")
24
25  R2 : Comment « Traiter le choix de l'utilisateur »
26      if choix == '1':
```

```
27         compteur += 1
28     elif choix == '2':
29         compteur -= 1
30     elif choix != '0':
31         print("Je n'ai pas compris.")
```

Solution :

```
1
2  def gerer_compteur():
3      '''
4      Gérer un compteur avec un menu
5      '''
6      compteur = 0
7      choix = '1'
8      while choix != '0':
9          # Afficher le compteur
10         print()
11         print("Compteur :", compteur)
12         print()
13
14         # Afficher le menu
15         print("1. Incrémenter")
16         print("2. Décrémenter")
17         print("0. Quitter")
18
19         # Demander le choix de l'utilisateur
20         choix = input("Votre choix : ")
21
22         # Traiter le choix de l'utilisateur
23         if choix == '1':
24             compteur += 1
25         elif choix == '2':
26             compteur -= 1
27         elif choix != '0':
28             print("Je n'ai pas compris.")
29
30
31 if __name__ == '__main__':
32     gerer_compteur()
```

Exercice 2 : Retrouver un raffinement

L'objectif de cet exercice est de retrouver les raffinages qui sont à l'origine du programme ci-après. On écrira les raffinages dans le fichier `reviser_tables_multiplications_raffinages.txt`.

```
1  import random
2
3  def reviser_table():
4      '''
5      Réviser une table de multiplication.
```

```

6
7     Principe : On pose plusieurs multiplications à l'utilisateur sur une table puis
8     on lui affiche un bilan.
9     '''
10
11     # Demander la table à réviser
12     table = int(input('Table à réviser : '))
13
14     # Poser 10 multiplications
15     nb_erreurs = 0
16     for _ in range(10):
17         # Poser une multiplication
18         # Choisir un nombre entre 0 et 10
19         nombre = random.randint(0, 10)
20
21         # Construire l'énoncé de la multiplication
22         enonce = '{} * {} = '.format(table, nombre)
23
24         # Demander la réponse à l'utilisateur
25         reponse = int(input(enonce))
26
27         # Evaluer la réponse
28         if reponse == table * nombre:    # Réponse correcte ?
29             print('Correct')
30         else:
31             print('Erreur')
32             nb_erreurs += 1
33
34     # Afficher le bilan
35     # Calculer le nombre de bonnes réponses
36     nb_bonnes_reponses = 10 - nb_erreurs
37     # Afficher le nombre de bonnes réponses
38     print('Nombre de bonnes réponses :', nb_bonnes_reponses)
39     # Afficher la mention
40     if nb_erreurs == 0:
41         print('Excellent !')
42     elif nb_erreurs == 1:
43         print('Très bien.')
44     elif nb_erreurs <= 3:
45         print('bien.')
46     elif nb_bonnes_reponses >= 4:
47         print('Moyen.')
48     elif nb_bonnes_reponses == 0:
49         print("Est-ce que tu l'as fait exprès ?")
50     else:
51         print('Il faut retravailler cette table.')
52
53     reviser_table()

```

2.1. Retrouver tous les raffinages. On ne détaillera pas les dernières actions complexes, celles

dont le raffinement ne contient que des actions élémentaires.

Pour cette question, il ne s'agit pas de faire preuve d'imagination : la réponse est dans le programme fourni ! Il suffit juste de retrouver les actions complexes et de les mettre en forme en respectant la présentation des raffinages utilisée dans le cours.

Solution :

```

1  R0 : Réviser une table de multiplication
2
3  Exemples : Omis. Non présents dans le programme fourni.
4
5  R1 : Comment « Réviser une table de multiplication » ?
6      Demander la table à réviser          (table: out int)
7      Poser 10 questions                   (table: in ; nb_erreurs: out int)
8      Afficher le bilan                    (nb_erreurs: in)
9
10 R2 : Comment « Poser 10 questions » ?
11     nb_erreurs = 0
12     for _ in range(10):
13         Poser une question                (table: in ; nb_erreurs: in out)
14
15 R3 : Comment « Poser une question » ?
16     Choisir un nombre entre 0 et 10      (nombre: out int)
17     Construire l'énoncé de la question   (table, nombre: in ; enonce: out str)
18     Demander la réponse à l'utilisateur (enonce: int ; reponse: out int)
19     Evaluer la réponse                   (table, nombre, reponse: in ; nb_erreurs: in out)
20
21 R2 : Comment « Afficher le bilan » ?
22     Calculer le nombre de bonnes réponses (nb_erreurs: in ; nb_bonnes_reponses: out)
23     Afficher le nombre de bonnes réponses (nb_bonnes_reponses: in)
24     Afficher la mention                  (nb_erreurs, nb_bonnes_reponses: in)
25
26 R3 : Comment « Evaluer la réponse » ?
27     if réponse correcte:                  (table, nombre, reponse: in)
28         print('Correct')
29     else:
30         print('Erreur')
31         nb_erreurs += 1
32
33 -- Voici le raffinement des actions complexes qui n'introduisent pas de
34 -- nouvelles actions complexes.
35 -- Il n'y a pas d'ordre pour afficher les raffinages. La seule contrainte est
36 -- qu'ils ne doivent pas apparaître le raffinement qui introduit l'action
37 -- complexe décomposée.
38
39 R4 : Comment « réponse correcte » ?
40     Résultat = reponse == table * nombre
41
42 R4 : Comment « Choisir un nombre entre 0 et 10 » ?
43     nombre = random.randint(0, 10)
44

```

```
45 R4 : Comment « Constuire l'énoncé de la question » ?
46     enonce = '{} * {} = '.format(table, nombre)
47
48 R4 : Comment « Demander la réponse à l'utilisateur » ?
49     reponse = int(input(enonce))
50
51 R3 : Comment « Calculer le nombre de bonnes réponses » ?
52     nb_bonnes_reponses = 10 - nb_erreurs
53
54 R3 : Comment « Afficher le nombre de bonnes réponses » ?
55     if nb_erreurs == 0:
56         print('Excellent !')
57     elif nb_erreurs == 1:
58         print('Très bien.')
59     elif nb_erreurs <= 3:
60         print('bien.')
61     elif nb_bonnes_reponses >= 4:
62         print('Moyen.')
63     elif nb_bonnes_reponses == 0:
64         print("Est-ce que tu l'as fait exprès ?")
65     else:
66         print('Il faut retravailler cette table.')
```

2.2. Ajouter les flots de données.

En analysant le code d'une action complexe, on identifie les données qu'elle manipule. Attention à ne pas confondre *donnée manipulée* par l'action (et donc exploitées par d'autres actions appartenant au même raffinement) et *variable locale* à une action complexe (utilisée seulement en interne de cette action complexe).

Solution : Ils ont été ajoutés dans la réponse à la question précédente.

2.3. Indiquer les sous-programmes qu'il aurait été judicieux de faire.

Solution : Il aurait été judicieux de définir les sous-programmes suivants :

* **def** poser_multiplication(table) : poser une multiplication * **def** mention(nb_erreurs) : la mention correspondant à un nombre d'erreurs

On aurait aussi pu définir des sous-programmes pour :

- Demander la table à réviser (surtout si on veut que la saisie soit robuste), ceci correspond au sous-programme `input_int`.
- Poser plusieurs multiplications avec comme paramètres la table à réviser et le nombre de multiplications à poser ;

Notons qu'il serait préférable de nommer la constante littérale 10, par exemple `NB_MULTIPLICATIONS`.