4. Il suffit d'exécuter les lignes de haut en bas. La dernier ligne change la valeur de b, mais pas celle de c. Lors d'une affectation (avec un signe = en python), on fait d'abord le calcul de ce qui est à droite du signe égal et on « l'enregistre » ensuite dans la variable qui est à gauche du signe égal. On trouve (2*2 – 3*5) c'est-à-dire -11.

5. (a) la condition n'est pas vérifiée, donc l'instruction b = 1 n'est pas exécutée. La valeur que contient b est donc inchangée et vaut toujours 3.
(b) b>2 est vrai, donc (compte tenu du ou) la condition est vraie et b change de valeur et vaut 1
(c) idem (a), donc 3

6. la boucle for k in range(n) s'exécute exactement n fois … mais « a » ne change pas de valeur, on fait donc toujours le même calcul et à la fin b contient 2

7. cette fois, la valeur de b change au cours de l'exécution. Initialement b vaut 0 et on fait 5 fois b = b + 2. La première fois cela revient à faire b = 0 + 2, la deuxième fois puisque b vaut maintenant 2, on fait donc b = 2+2, la 3^{e} fois b = 4+2 etc… À la fin, b contient donc 10

8. ici on fait b = 0+1 à la première itération, puis b = 1 + 2 à la 2^{e} puis b = 3 + 3 à la 3^{e} puis b = 6+4, puis b = 10+5, soit à la fin b = 15

9. c'est presque le même calcul que précédemment, mais il ne faut pas oublier que dans une boucle for k in range(n), la boucle s'exécute n fois, et que k prend les entiers successifs en commençant à 0 … et donc en s'arrêtant à « n-1 ». Ainsi le calcul est plutôt a = 0+0, puis a = 0+1, puis a = 1+2, puis a = 3+3, puis a = 6+4, soit à la fin a = 10

10. a contient initialement 0, puis tant qu'il est plus petit que 10 (strictement) on lui ajoute 3 et on recommence. Une erreur fréquente est de répondre 9, mais ce n'est pas possible, puisque 9 est plus petit que 10, on continue donc à ajouter 3. Il faut donc noter qu'avec une boucle while, lorsque l'on s'arrête c'est que la condition de boucle est fausse. Ici on s'arrête donc lorsque a contient 12.

11. 5 étant plus petit que 10, on ne passe pas dans le if mais dans le else : on aura donc « perdu »

12. On est en train de calculer les termes de la suite u0 = 2 ; $u_{n+1} = 5 \times u_n - 3$; les calculs sont donc res = 5*2-3 (=7), puis res = 5*7-3 (=32), puis res = 5*32 – 3 = 157. On a bien fait 3 fois les instructions dans la boucle for

13. Ici, inutile de réfléchir, a n'est pas modifié dans la boucle while. On peut se convaincre que le while va bien s'arrêter et a est donc inchangé et vaut toujours 0 à la fin.

14. (a) Il suffit de changer le programme de la question 12 en changeant la formule res = 5*res-3 en res = res**2 + 3 et en changeant l'initialisation res=2 en res = 5
(b) On a intérêt cette fois à faire un while car on ne connait pas à l'avance le nombre d'itération

```
def exo14b(a) :
    u = 5
    n = 0
    while u < a :#s'arrête bien dès que un>= a ; ce que l'on souhaite
        n +=1 #ou n = n + 1
        u = u**2 + 3
    return n #attention, c'est bien « n » qui est attendu et pas un
```

15. nom_d_une_liste.append(élément) modifie la liste contenue dans la variable « nom_d_une_liste » en ajoutant l'élément « élément » à la fin. Les indices commencent à 0 et vont jusque len(nom_d_une_liste) – 1 ; lst[3] contient donc 7, l'élément qui vient d'être ajouté et qui est placé en « 4^{e} » position (0,1,2,3) ; lst contient donc [5, 6, 10, 7]. Pop n'est pas vraiment à votre programme donc oublier les questions c et d.

16. Il suffit dans le programme de la question 14a de rajouter avant la boucle une initialisation du type liste = [u] (pour avoir le premier terme) et dans la boucle (après le calcul) liste.append(u). De cette façon, on a bien u0 en premier et les n termes calculés ensuite sont ajoutés un par un.

```
def exo16(n):
    u = 5 #u0
    liste = [u] #liste contenant un seul élément : u0
    for i in range(n):
        u = u**2 + 3 #on calcule "u{n+1}" en fonction de "u{n}"
        liste.append(u) #on "enregistre" à la fin de la liste
    return liste
```

```
def exo16bis(n):
    u = 5
    liste = [0] * (n+1) #on connait à l'avance le nombre de termes
    for i in range(n+1):
        liste[i] = u #on enregistre à la bonne position
        u = u**2 + 3 #on calcule "u{n+1}" en fonction de "u{n}"
    return liste
```

17. Cela s'appelle une liste en compréhension il suffit d'exécuter le calcul pour chaque valeur de i, chaque résultat est un élément de la liste. Ici on aura donc [1, 4, 7, 10, 13]