Exercices du cours 4

1 Sans faire de preuve précise, expliquer si les boucles suivantes se terminent.

1.

```python
for i in range(10 ** 30):
    print(i)
```

C'est une boucle for, la terminaison est assurée même si le temps mis sera long.

2.

```python
i = 0
while i < 10:
    i = i - 1
```

La variable i vaut initialement 0 et diminue de 1 à chaque passage dans la boucle, elle n'atteindra jamais la valeur 10 : la boucle est infinie.

3.

```python
i = 0
while True:
    i = i + 1
    print(i)
```

La commande $while\ True$ donne une boucle infinie.

4.

```python
n = 100
i = 0
while i < n:
    i = i + 2
    n = n + 1
```

La boucle se termine, la quantité $n - i$ est un variant de boucle. En effet, à chaque passage dans la boucle, i augmente de 2 et n augmente de 1 donc $n - i$ diminue de 1.

5.

```python
n = 10
i = 0
while i < n:
    i = i + 1
    n = n + 1
```

La boucle est infinie, la condition $i < n$ sera toujours vérifiée car $n - i$ reste égal à 10.

6.

```python
i = 1
while (i < 10) or (i % 2 == 1):
    i = i +2
```

La variable i vaut initialement 1 et on lui ajoute 2 à chaque passage dans la boucle, ainsi elle reste impaire. La condition de la boucle $while$ sera donc toujours vérifiée.

7.

```python
x = 1
while x / 2 >0:
    x = x / 2
```

Les nombres flottants positifs que l'on peut représenter en Python vont de $2.2250738585072014\times 10^{-308}$ à $1.7976931348623157 \times 10^{308}$. Ainsi, si l'on programme cette boucle en Python, elle se termine.

8.

```python
import random as rd
n = rd.randint(10 ** 300, 10 ** 400)
while n > 1:
    if n % 2 == 0:
        n = n // 2
    else :
        n = 3 * n + 1
```

On ne sait pas si la boucle se termine. La conjecture de Syracuse affirme que cette boucle se termine pour tout entier naturel n mais cette assertion n'est pas démontrée à ce jour.

$\boxed{2}$

1. En testant quelques valeurs, on conjecture que cette fonction renvoie 2^{2^n}.
2. Un **variant de boucle** est $n-i$. En effet, c'est bien un entier naturel qui décroit strictement à chaque passage dans la boucle car i est incrémenté de 1 et n reste fixe. On en déduit qu'il n'y aura qu'un nombre fini de passage dans la boucle et que la fonction se termine.
3. On note r_i la valeur de r à la fin de la i-ème itération de la boucle avec par convention $r_0 = 2$. Démontrons que la propriété $r_i = 2^{2^i}$ est un **invariant de boucle** pour cette fonction.

   • Initialement, on a $r_0 = 2$ ainsi $2^{2^i} = 2^{2^0} = 2^1 = 2$ comme voulu. La propriété est vérifiée avant l'exécution de la boucle.

   • Soit $i \in [\![1, n]\!]$, on suppose que la propriété est vraie avant la i-ème itération, c'est-à-dire que $r_{i-1} = 2^{2^{i-1}}$. À la fin de la i-ème itération, on a :

   $$r_i = r_{i-1} \times r_{i-1} = \left(2^{2^{i-1}}\right)^2 = 2^{2^{i-1}\times 2} = 2^{2^i}$$

   Si la propriété est vraie avant la i-ème itération alors elle est vraie à la fin de la i-ème itération. On a bien un invariant de boucle.

   La fonction s'arrête à la n-ième itération de la boucle et à ce moment là, on a bien $r_n = 2^{2^n}$.

3 • Si $p \leq 0$, on ne rentre pas dans la boucle et le programme se termine.

• Si $p > 0$, on rentre dans la boucle while. On ne peut pas choisir p comme variant de boucle car ce n'est pas une quantité strictement décroissante à chaque passage dans la boucle. On va démontrer que $2p+3c$ est un variant de boucle, déjà il est clair que c'est un entier naturel. Notons p_i et c_i les variables p et c après l'itération numéro i de la boucle.

▶ Si $c_i = 0$ alors $p_{i+1} = p_i - 2$ et $c_{i+1} = 1$ d'où :

$$2p_{i+1} + 3c_{i+1} = 2p_i - 4 + 3 = 2p_i - 1 < 2p_i + 3c_i$$

▶ Si $c_i = 1$ alors $p_{i+1} = p_i + 1$ et $c_{i+1} = 0$ d'où :

$$2p_{i+1} + 3c_{i+1} = 2p_i = 2p_i + 2 < 2p_i + 3 = 2p_i + 3c_i$$

Dans tous les cas la quantité $2p_i + 3c_i$ diminue strictement à chaque itération de la boucle, ainsi la boucle se termine.

4

1. Cette fonction met en jeu une boucle for qui se termine bien.
2. On choisit comme invariant de boucle, la propriété suivante valable pour $i \in [\![0, n]\!]$.

P_i : après le passage numéro i dans la boucle la valeur de p est $i!$

• Cette propriété est vraie avant de rentrer dans la boucle puisqu'initialement $p = 1 = 0!$.

• On suppose que la propriété est vraie après le passage numéro i dans la boucle, c'est-à-dire qu'à ce moment-là p vaut $i!$. Lors de la boucle numéro $i+1$, p est multiplié par $i+1$, ainsi à la fin de la boucle la valeur de p est $(i+1) \times i! = (i+1)!$, ce qu'il fallait démontrer.

• À la sortie de la boucle *while*, c'est-à-dire pour $i = n$, on a $p = n!$ comme voulu.

5

1. Ce programme affiche a^n, cela ne saute pas aux yeux en regardant les lignes de commande mais on peut l'intuiter en testant quelques valeurs
2. Déjà, il est clair que ce programme se termine : notre **variant de boucle** est N qui est un entier naturel positif et qui décroît strictement à chaque passage dans la boucle. Ceci se démontre sans difficulté en raisonnant selon la parité de N. L'algorithme se termine.
3. On considère la propriété :

$$P \ : \ "A^N \times R = a^n"$$

• **Initialisation.** Cette propriété est vraie avant de rentrer dans la boucle puisqu'au moment de l'initialisation $A = a$, $N = n$ et $R = 1$.

• **Hérédité.** Supposons la propriété vraie avant l'exécution d'une boucle. Il y a trois cas à considérer :

▶ Si $N = 0$, on sort de la boucle et les valeurs des paramètres restent inchangées.

▶ Si N est pair, alors A est transformé en $A' = A^2$, N est changé en $N' = \dfrac{N}{2}$ et $R' = R$ reste inchangé. On a alors :

$$A'^{N'} \times R' = (A^2)^{\frac{N}{2}} \times R = A^N \times R = a^n$$

- Si N est impair, alors A est inchangé $A' = A$, N est changé en $N' = N - 1$ et on a $R' = R \times A$. On a alors :

$$A'^{N'} \times R' = A^{N-1} \times R \times A = A^N \times R = a^n$$

Dans tous les cas notre propriété reste vérifiée à la fin de l'exécution d'une boucle.

• **Conclusion.** Lorsque l'on sort de la boucle, on a nécessairement $N = 0$ et l'on sait que notre propriété P reste encore vérifiée, c'est-à-dire :

$$A^N \times R = a^n \Leftrightarrow R = a^n$$

Ce qui démontre que le programme affiche bien a^n à la fin.

$\boxed{6}$

1. On propose l'algorithme suivant où n est l'entier naturel que l'on veut tester :

```
while n > 70:
    a = int(str(n)[-1]) #a est le chiffre des unités de n
    n = ((n - a) // 10) -2 * a #le pas de l'algorithme
print(n)
```

2. Pour démontrer que l'algorithme se termine, il suffit d'exhiber un variant de boucle. L'entier naturel n décroît strictement à chaque itération de la boucle, en effet si l'on note n' la valeur de n en fin de boucle, on a : $n' \leq \frac{1}{10}n < n$. La boucle se termine.
3. Notons n' la valeur prise par l'entier n en fin de boucle. On va démontrer qu'un invariant de boucle est : "$7|n \Leftrightarrow 7|n'$". Cette propriété justifiera la conformité de notre algorithme et donc la validité de la méthode pour déterminer si un entier est divisible par 7.
   Si $n = 10q + r$ où q et r sont le quotient et le reste de la division euclidienne de n par 10. On a : $n' = \frac{n - r}{10} - 2r = \frac{n - 21r}{10}$, ce qui donne :

$$7|n \Leftrightarrow 7|n - 21r \Leftrightarrow 7|\frac{n - 21r}{10} = n' \text{ car 7 et 10 sont premiers entre eux}$$