Chapitre 3

Boucles

En programmation ou dans les algorithmes, il est souvent intéressant de répéter une action de nombreuses fois.

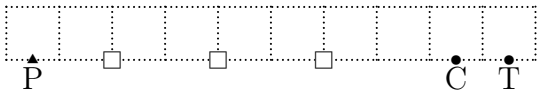
Exemple : Reprenons l'exemple du pirate P qui doit récupérer le trésor T. On rappelle que pour cela, il doit avoir la clé C de celui-ci ; qu'il se déplace sur le quadrillage ; que les carrés blancs représentent des obstacles qui doivent être contournés ; et qu'il n'est capable que d'effectuer les instructions suivantes :

Avancer : avancer d'une case ;

Sauter : sauter par dessus un obstacle jusqu'à la cas suivante ;

Ramasser : ramasser un objet sur la case occupée ;

Ouvrir : Ouvrir le coffre (nécessite la clé).



Nous avons vu que les instructions ci-dessous constituent un algorithme permettant au pirate de récupérer le trésor.

1. Avancer
2. Sauter
3. Avancer
4. Sauter
5. Avancer
6. Sauter
7. Avancer
8. Avancer
9. Ramasser
10. Avancer
11. Ouvrir

Toutefois, on remarque que l'on enchaîne la séquence « Avancer, Sauter » plusieurs fois de suite. Il serait plus concis et clair écrire cela sous la forme

1. Répéter 3 fois :
   (a) Avancer
   (b) Sauter
2. Répéter 2 fois :
   (a) Avancer
3. Ramasser
4. Avancer
5. Ouvrir

La répétition d'un bloc d'instructions déterminés est permise par les **boucles**. Elles permettent d'éviter d'avoir à écrire plusieurs lignes de code identiques. Il existe deux types de boucles :

— les **boucles conditionnelles** qui s'exécutent **tant qu'une** certaine condition est vérifiée ;

— les **boucles non conditionnelles** qui s'exécutent **pour** des valeurs prédéfinies avant le démarrage de la boucle.

3.1 Boucles conditionnelles

3.1.1 Boucles conditionnelles

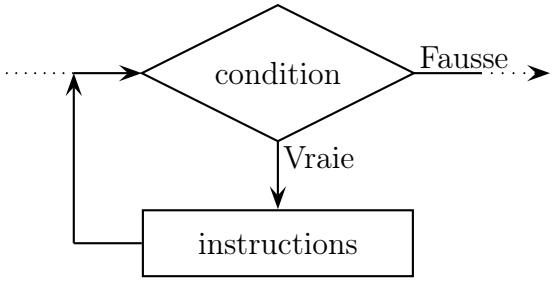
Les boucles conditionnelles s'exécutent **tant qu'une** certaine condition est vérifiée ; elles sont donc plus communément appelées boucles **Tant que** ou boucle *while* en anglais. À l'instar des instructions conditionnelles, un test est effectué pour déterminer si la condition de la boucle est vérifiée ou non.

En pseudo-code, on retrouvera les boucles **Tant que** de la façon suivante.

Algorithme 1 : Structure boucle **Tant que**

```
1 Tant que condition :
2 |  instructions
```

Cela peut également se représenter sous la forme du logigramme ci-dessous sur lequel on voit bel et bien une boucle.



Exemple : Dans l'algorithme ci-dessous, tant que la valeur de N est plus petite que 10, on la multiplie par deux et on arrête dès qu'elle devient plus grande que 10. On peut résumer l'évolution de la boucle par le tableau ci-dessous.

Algorithme 2 : Exemple **Tant que**

```
1 N ← 1
2 Tant que N < 10 :
3 |  N ← 2 × N
4 Renvoyer : N
```

N	1	2	4	8	16
Condition	V	V	V	V	F

À $N = 16$, la condition n'est plus vérifiée et la boucle s'arrête, l'algorithme affiche en sortie la dernière valeur de N : 16.

3.1.2 Début et fin d'une boucle conditionnelle

Lorsqu'on manipule des boucles **Tant que**, il faut être vigilant à deux choses.

1. Que la boucle puisse bien démarrer. En effet, si la condition de la boucle n'est pas vérifiée au départ, elle ne démarre pas et est donc ignorée : on passe à la suite du programme. Par exemple :

Algorithme 3 : Boucle ignorée

```
1 N ← 1
2 Tant que N > 10 :
3 |   N ← 2 × N
4 Renvoyer : N
```

Ici, N n'est pas plus grand que 10 et donc la condition de la boucle étant fausse, elle ne démarre pas. On passe alors directement à l'instruction de la ligne 4 pour afficher la valeur de N, i.e. 1.

2. Que le boucle ait une fin. En effet, si la condition est toujours vérifiée, la boucle ne s'arrête jamais : elle est infinie. Par exemple :

Algorithme 4 : Boucle infinie

```
1 N ← 1
2 Tant que N < 10 :
3 |   N ← N − 1
4 Renvoyer : N
```

Ici, N est toujours plus petit que 10 ($1, 0, -1, \dots$) et donc la condition de la boucle étant toujours vraie, elle continue indéfiniment.

Remarque : pour qu'une boucle **Tant que** ne soit pas infinie, il est important que la condition la définissant cesse d'être Vraie pour devenir Fausse à un moment voulu par le programmateur.

3.1.3 Boucles conditionnelles en Python

La boucle **Tant que** s'exprime à l'aide du mot clé `while` en Python et dans la plupart des langages de programmation. En Python, comme pour les fonctions et les instructions conditionnelles, les instructions contenues dans la boucle sont marquées par une indentation.

Exemple : La traduction de l'algorithme exemple 1 en Python est :

```
N = 1

while N < 10 :

    N = N * 2 # Dans la boucle while car indentation


print(N) # Hors de la boucle while car plus d'indentation
```

Exercices : 3.1 à 3.4 ; 3.9 et 3.10.

3.2 Boucles non conditionnelles

3.2.1 Boucles non conditionnelles

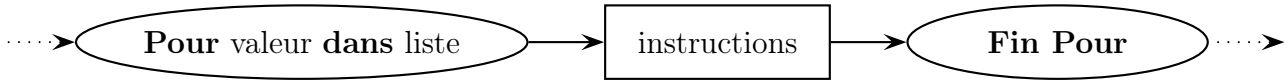
Une boucle inconditionnelle est une boucle dans laquelle sont prédéterminées les valeurs **pour** lesquelles va se réaliser le bloc d'instructions qu'elle contient. Ces boucles sont communément appelées boucles **Pour**. Les valeurs à parcourir sont précisées généralement sous forme de listes ou en donnant la première et la dernière. Comme les valeurs sont définies avant le début de la boucle, il n'y a pas de problèmes de début et de fin à considérer contrairement aux boucles **Tant que**.

En pseudo-code, on retrouvera les boucles **Tant que** de la façon suivante.

Algorithme 5 : Structure boucle **Pour**

```
1 Pour valeur dans liste :
2 |   instructions
```

Remarque : une boucle **Pour** peut toujours se ramener à une Boucle **Tant que**, elle n'a donc pas schéma de logigramme ; on peut utiliser celui de la boucle **Tant que** ou ceux de début et de fin de séquences pour la démarquer comme ci-dessous (on perd cependant l'aspect visuel de la boucle).



Exemple : Le programme ci-dessous affiche successivement "Jour 1", "Jour 2" et "Jour 3".

Algorithme 6 : Exemple **Pour** 1

```
1 Pour numéro_jour Allant de 1 à 3 :
2 |   Afficher("Jour {numéro_jour}")
```

Remarque : si la variable du Pour n'apparaît pas dans les instructions répétées, alors on retrouve le même résultat à chaque étape. C'est ce que l'on peut voir dans l'exemple ci-dessous qui une traduction en pseudo-code de l'enchaînement des trois avancées et sauts que le pirate doit effectuer dans l'exemple initial. La variable « nombre » est complètement muette, elle ne sert qu'à parcourir l'ensemble de valeurs $\{1,\ 2,\ 3\}$.

Algorithme 7 : Exemple **Pour** 2

```
1 Pour nombre Allant de 1 à 3 :
2 |   Avancer()
3 |   Sauter()
```

3.2.2 Boucles non conditionnelles en Python

La boucle **Pour** s'exprime à l'aide des mots-clés `for` et `in` en Python selon la syntaxe suivante.

```
for variable in liste :

    instructions
```

La liste peut être préexistante ou est créée sur le moment à l'aide du mot-clé `range`. Ce dernier permet de créer une liste en précisant :

— obligatoirement sa fin ;

— en première option, son début ;

— en dernière option, le pas auquel avance la variable entre le début et la fin.

Exemples :

1. `range(10)` : on n'a que la fin, le début est par défaut à 0, 10 n'est pas atteint, il s'agit d'un seuil ; on a donc la liste 0, 1, 2,..., 9.
2. `range(1,10)` : on a le début et la fin ; on a ici la liste 1, 2,..., 9.
3. `range(1,11,2)` : on a le début, la fin et le pas qui vaut ici 2 ; cela signifie que l'on avance de 2 en 2 à partir de 1 jusqu'au seuil 11 qui n'est toujours pas atteint ; on a donc la liste 1, 3, 5, 7, 9.

Exercices : 3.5 à 3.8 ; 3.11 et 3.12.

3.3 Capacités attendues

— Lire et exécuter une séquence d'instructions comportant une boucle sous forme d'algorithme ou de logigramme.

— Compléter une séquence d'instructions comportant une boucle sous forme d'algorithme ou de logigramme.

— Écrire une séquence d'instructions comportant une boucle sous forme d'algorithme ou de logigramme.

— Déterminer si une boucle **Tant que** est bien définie : si elle a un début et une fin.

3.4 Exercices

3.4.1 Progresser

Boucles conditionnelles

Exercice 3.1. Qu'affichent les algorithmes ci-dessous en sortie ? Dessiner leurs logigrammes.

Algorithme 8 :

```
1 k ← 1
2 Tant que k < 10 :
3 |  k ← 3 × k
4 Renvoyer : k
```

Algorithme 9 :

```
1 k ← 16
2 Tant que k > 1 :
3 |  k ← k/4
4 Renvoyer : k
```

Exercice 3.2. Déterminer si la condition de l'instruction **Tant que** des algorithmes suivants est bien définie ou non (boucle infinie ou ne démarrant pas).

Algorithme 10 :

```
1 k ← 5
2 Tant que k < 10 :
3 |  Renvoyer : k
4 |  k ← (k + 1)/2
```

Algorithme 11 :

```
1 k ← 10
2 Tant que k ⩽ 1 :
3 |  k ← k + 1
4 Renvoyer : k
```

Algorithme 12 :

```
1 k ← 0
2 Tant que k < 10 :
3 |  Renvoyer : k
4 k ← k + 1
```

Algorithme 13 :

```
1 T ←"Ah"
2 Tant que longueur(T) ⩽ 10 :
3 |  T ← T + T
4 Renvoyer : T
```

Exercice 3.3.

1. Compléter l'algorithme ci-dessous de sorte qu'il affiche le premier *entier* tel que la somme des entiers de 1 à *entier* soit plus grande que 100 (par exemple, *entier* n'est pas égal à 5 puisque $1+2+3+4+5=15<100$ ni 6 puisque $1+2+3+4+5+6=21<100$).

Algorithme 14 : Seuil d'une somme

```
1 somme ← 0
2 entier ← 0
3 Tant que ..... :
4 |  entier ←.....
5 |  somme ←.....
6 Renvoyer : .....
```

2. Donner le logigramme associé à l'algorithme ci-dessus.

Exercice 3.4. Écrire un algorithme ou un logigramme permettant de déterminer P, la première puissance de 2 telle que la somme de celle-ci et des précédentes puissances de deux soit plus grande que 1000 ; par exemple, P n'est pas 16 puisque $1 + 2 + 4 + 8 + 16 = 31 < 1\,000$.

Boucles non conditionnelles

Exercice 3.5. [Des pokémons] On considère que la variable *pokedex* est une liste de tous les pokémons et que ***nom()*** et ***type()*** sont des fonctions permettant d'accéder au nom et au type du pokémon donné en entrée. Que fait l'algorithme suivant ?

Algorithme 15 :

```
1 Pour pokémon dans pokédex :
2 |   Si type(pokémon) = "feu" :
3 |   |   Renvoyer : nom(pokémon)
```

Exercice 3.6. [Encore des pokémons] Dans la lignée de l'exercice précédent, compléter l'algorithme ci-dessous afin qu"il affiche en sortie le nom des pokémons de type foudre ou dragon.

Algorithme 16 :

```
1 Pour ..... :
2 |   Si ..... :
3 |   |   Renvoyer : .....
```

Exercice 3.7. Recopier et compléter cet algorithme de sorte qu'il affiche la somme des 10 premiers entiers naturels.

Algorithme 17 : $1 + 2 + \cdots + 10$

```
1 somme ← .....
2 Pour k allant de ..... à ..... :
3 |   somme ← .....
4 Renvoyer : .....
```

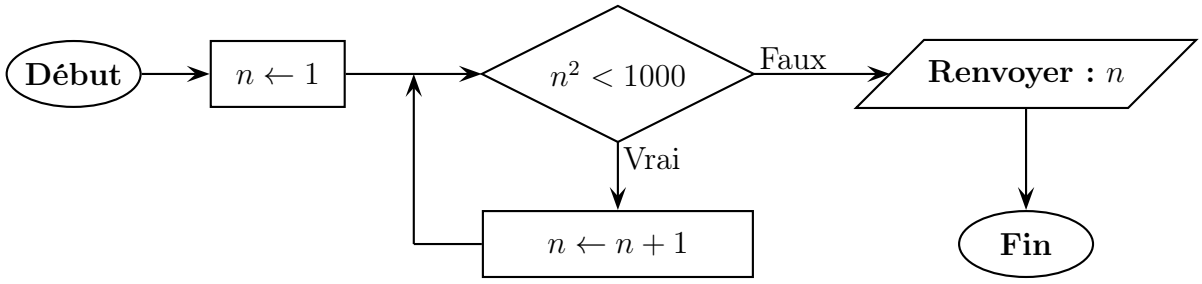
Exercice 3.8. [Toujours des pokémons] On suppose que l'on a accès aux fonctions de l'exercice **??** et en plus que ***taille()*** et ***poids()*** sont des fonctions permettant d'accéder à la taille et au poids du pokémon donné en entrée.

1. Écrire un algorithme renvoyant le nom des pokémons eau mesurant plus de 1m.
2. Écrire un algorithme renvoyant le nom des pokémons pesant entre 50 et 100kg.
3. Écrire un algorithme renvoyant le nom des pokémons de type plante et, pesant moins de 20kg ou plus 30kg.

3.4.2 S'entraîner

Boucles conditionnelles

Exercice 3.9. Le logigramme suivant contient-il une boucle ? Si oui, laquelle ? Que fait l'algorithme correspondant ?



Exercice 3.10. Déterminer si la condition de l'instruction **Tant que** des algorithmes suivants est bien définie ou non (boucle infinie ou ne démarrant pas).

Algorithme 18 :

```
1 k ← 5
2 Tant que k > 10 :
3 |   Renvoyer : k
4 |   k ← k/2
```

Algorithme 19 :

```
1 k ← 10
2 Tant que k ⩾ 1 :
3 |   k ← k²
4 Renvoyer : k
```

Boucles non conditionnelles

Exercice 3.11. Qu'affichent les algorithmes ci-dessous en sortie ?

Algorithme 20 :

```
1 Pour k allant de 0 à 10 :
2 |   Renvoyer : 2 × k
```

Algorithme 21 :

```
1 Pour k allant de 0 à 10 :
2 |   Renvoyer : 2 × k + 1
```

Exercice 3.12. [Toujours plus de pokémons] On suppose l'on a accès aux fonctions des exercices 3.5 et 3.5. On suppose par ailleurs que ***vitesse()*** et ***attaque()*** sont des fonctions permettant d'accéder à la vitesse et l'attaque du pokémon donné en entrée.

1. Écrire un algorithme renvoyant le nom des pokémons roche dont la vitesse est supérieure à 100.
2. Écrire un algorithme renvoyant le nom des pokémons pesant entre 50 et 70kg, mesurant entre 1 et 2m et ayant une attaque supérieure à 200.