

ALGORITHMIQUE

↔ Structures répétitives : les boucles ↔

I. Introduction

I.1. Exemples introductifs

Exemple 1

On souhaite écrire un algorithme qui demande à l'utilisateur de saisir un nombre réel positif puis qui calcule la racine carrée de ce nombre.

Résolution

Lors de la saisie, une **erreur** peut être commise et si le réel tapé au clavier est négatif, le programme sera bloqué. Pour éviter cela, on fait un *contrôle de saisie* : l'algorithme demande la saisie d'un réel positif et en cas d'erreur de l'utilisateur, la demande est réitérée.

L'instruction de saisie est donc réalisée *au moins une fois* puis répétée jusqu'à ce que le nombre soit positif.

Exemple 2

On souhaite écrire un algorithme qui demande à l'utilisateur de saisir un nombre réel, qui lui enlève 2, puis qui retranche 2 du résultat jusqu'à obtenir un nombre strictement négatif.

Résolution

Il faut ici effectuer plusieurs soustractions successives (on *ignore combien*). De plus, si le nombre saisi est négatif, les opérations sont inutiles. Le signe du nombre saisi doit être **testé**, puis il y aura, le cas échéant, **répétition** de soustractions jusqu'à obtenir le résultat voulu.

Exemple 3

On souhaite écrire un algorithme qui permet la saisie de 10 entiers et qui teste leurs signes.

Résolution

Dans cet exemple, il faut répéter 10 fois la saisie d'un entier et le test de son signe. On **sait à l'avance combien de fois** le traitement (saisie puis test) doit être effectué.

Exemple 4

On souhaite écrire un algorithme qui affiche la table de multiplication par 7 (de 7 fois 0 jusqu'à 7 fois 12).

Résolution

Ici, il faut effectuer 13 multiplications par 7 et afficher leur résultat. Et on connaît le nombre de répétitions du traitement.

De plus, 7 est multiplié par un entier qui prend les valeurs successives : 0, 1, 2, etc jusqu'à 12.

Il est donc astucieux de réitérer 13 fois le traitement en commençant par 7×0 et en *augmentant* à chaque répétition le multiplicateur.

I.2. Bilan

► Bilan

Pour résoudre chacun de ces problèmes, on utilise une **structure répétitive** (ou **boucle**) ; celle-ci permet de faire exécuter plusieurs fois la même chose à la machine jusqu'à ce que l'on obtienne un résultat conforme aux consignes du problème.

💬 Définition

Une **boucle** permet d'exécuter plusieurs fois de suite une même séquence d'instructions.

Cet ensemble d'instructions s'appelle le **corps** de la boucle.

Chaque exécution du corps d'une boucle s'appelle une **itération**, ou encore un passage dans la **boucle**.

📄 Algorithme

Il existe trois types de **boucles** :

- la boucle **RÉPÉTER** ;
- la boucle **TANT QUE** ;
- la boucle **POUR**.

👉 Remarque

En **python**, il n'existe que les boucles **TantQue** et **Pour**.

⚠ Attention

Chaque **boucle** comporte **OBLIGATOIREMENT** un **test** (condition de sortie) qui permet soit de recommencer la boucle, soit de passer à la suite si le résultat voulu est atteint.

Si on ne contrôle pas cette condition, on peut se retrouver avec une **boucle infinie**.

Dans la présentation de l'algorithme, il conviendra de préciser la condition de sortie.

II. Les trois types de boucles

II.1. Répéter jusqu'à

📄 Algorithme

La structure algorithmique d'une boucle **Répéter** est donnée ci-dessous :

```
Répéter
<instructions>
Jusqu'à <conditions>
```

📄 Pseudo-Code

⚙ Propriétés

La partie **<instructions>** est exécuté une fois, puis **<conditions>** est évaluée ; si celle-ci est fausse, **<instructions>** est à nouveau exécuté.

Algorithme 1




On souhaite calculer la racine carrée d'un réel (positif) saisi par l'utilisateur.

Algorithme : Racine carrée avec contrôle de saisie
Variables : Nb (réel)

Début
 Répéter
 Afficher("Veuillez saisir un réel positif :") et Saisir(Nb)
 Jusqu'à (Nb ≥ 0)
 Nb ← sqrt(Nb)
 Afficher(Nb)
Fin

Pseudo-Code

Python


Ce type de boucle n'existe pas en . On la remplace par une boucle  ().

```
1 from math import * # pour charger les commandes math
2 Nb = float(input("Saisir un nombre réel positif : "))
3 while Nb < 0 :
4     Nb = float(input("Saisir un nombre réel positif : "))
5 print(f"La racine carrée de {Nb} est {sqrt(Nb)}")
```

Code Python

II.2. Tant Que


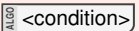

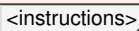
Algorithme

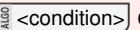
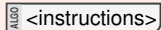
La structure algorithmique d'une boucle  est donnée ci-dessous.

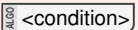

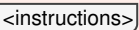
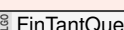
TantQue <condition> Faire
 <instructions>
FinTantQue

Pseudo-Code

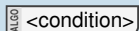

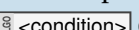

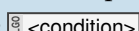
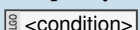

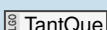

Propriétés

Lorsque le mot-clé  est rencontré,  est évaluée. Si sa valeur est ,  est exécuté (entrée dans la boucle).


À la rencontre du mot-clé ,  est de nouveau évaluée. Si sa valeur est ,  est exécuté une nouvelle fois.

Lorsque la valeur de  est ,  est ignoré et on passe directement à l'instruction suivant le  (sortie de boucle).


Remarques

- Avant la première exécution de la boucle, si  est , l'entrée dans la boucle est assurée. Mais si  est , la boucle ne sera pas réalisée.
- Nous devons nous assurer que  est modifié par la boucle, et qu'il y a bien une condition de sortie, c'est à dire qu'à un moment donné du déroulement du programme  sera .
- Une fois la condition d'entrée réalisée, une boucle  est exécutée tant que la condition reste .


Python

Analyser les programmes  suivants :

```
1 #Algo v1
2 while (n<7) :
3     n=n+1
4     print(n)
```

 Code Python

```
1 #Algo v2
2 n=0
3 while (n>0) :
4     n=n+1
5     print(n)
```

 Code Python

```
1 #Algo v3
2 n=0
3 while (n>=0) :
4     n=n+1
5     print (n)
```

 Code Python

```
1 # Exemple 2 de l'introduction
2 Nb = float(input("Saisir un réel : "))
3 while Nb >= 0:
4     Nb = Nb - 2
5     print(Nb)
```

 Code Python


Algorithme 2

On souhaite écrire un algorithme dont le but est de deviner un nombre entier aléatoire compris entre 1 et 100 choisi par la machine. Selon que le nombre choisi par l'utilisateur est trop grand ou trop petit, la machine affiche un message adéquat.

Python

En , cela peut donner :

```
1 from random import * # pour charger le module aléatoire
2 secret = randint(1,100)
3 essai = 0
4 while essai != secret :
5     essai=int(input("Quelle est votre proposition ?"))
6     if essai > secret :
7         print("Trop grand, réessayez !")
8     elif essai < secret :
9         print("Trop petit, rejouez !")
10    else :
11        print("Gagné !")
```

 Code Python

II.3. Pour

Algorithme

La structure algorithmique d'une boucle **Pour** est donnée ci-dessous :

```
Pour <VarComptage> allant de <ValDébut> à <ValFin>[par pas de <Pas>] Faire
    <instructions>
FinPour
```

Pseudo-Code

Propriétés

- L'indication du **pas** est facultative ; par défaut, celui-ci est égal à 1.
- Lorsque l'instruction **Pour** est rencontrée la première fois, le protocole suivant s'enclenche :
 1. affecter **VarComptage** par **ValDébut** ;
 2. tester la condition **VarComptage ≤ ValFin** : si elle est **Vraie** aller au **3.** ; si elle est **Fausse** aller au **6.** ;
 3. exécuter **<instructions>** ;
 4. une fois en **FinPour**, incrémenter **VarComptage** (augmenter de **Pas** ou de 1 par défaut) ;
 5. aller au point **2.** ;
 6. continuer l'algorithme après le **FinPour**.

Python

En **Python**, cela se traduit par **for** :

```
1 for <VarComptage> in range(<ValDébut>, <ValFin>+1) :
2     <instructions>
3 <suite du programme>
```

Code Python

Remarques

En **Python** :

- **range(1, n+1)** permet de balayer tous les entiers de **1** (en 1) jusqu'à **n** ;
- de manière générale, **in** permet à une variable de balayer un **ensemble** (voir plus tard...).

Méthode

On utilise une boucle **Pour** lorsque l'on connaît à l'avance **le nombre d'itérations**.

Python

L'exemple proposé en introduction concernant une tableau de multiplication peut donner, en **Python** :

```
1 # Table de multiplication par 7 jusqu'à 12*7
2 for i in range(0, 13) :
3     Res = 7*i
4     print(f"7*{i}={Res}")
```

Code Python

Algorithme 3

Pour tout entier $n \geq 1$, on appelle factorielle de n et on note $n!$ l'entier $n! = 1 \times 2 \times \dots \times (n-1) \times n$.
Par exemple, $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$.

Algorithme : Calcul de factorielle n ($n > 0$)
Variables : Fact, i (entiers)

 Pseudo-Code

Début
Afficher("Donner la valeur de n") et Saisir(n)
Fact \leftarrow 1
Pour i allant de 1 à n Faire
 Fact \leftarrow Fact \times i
FinPour
Afficher("Factorielle de", n, "=", Fact)
Fin

Python

En  python, cela peut donner :

```
1 # Factorielle
2 n = int(input("Valeur de n : "))
3 Fact = 1
4 for i in range(1,n+1):
5     Fact = Fact*i
6 print(f"Factorielle de {n}={Fact}")
```

 Code Python

III. Boucles classiques

III.1. Répétition d'un algorithme

Intro

On souhaite recommencer l'exécution d'un algorithme **jusqu'à** ce que l'on donne l'ordre d'arrêter. Pour cela, on utilise une **variable** (en général de type caractère ou chaîne) dans laquelle sera saisi le choix de recommencer ou non.

On se sert des instructions de l'algorithme comme d'un **bloc** qui sera le corps d'une **boucle** gérée par l'indicateur **choix**.

Cet indicateur sera convenablement initialisé puis saisi à chaque passage dans la **boucle**.

Algorithme

On souhaite écrire un algorithme dont le but est le calcul et l'affichage du périmètre d'un rectangle. Pour cela on doit :

- demander à l'utilisateur de saisir la largeur et la longueur;
- permettre à l'utilisateur de recommencer en faisant afficher un message « Voulez-vous continuer? »

Algorithme : Périmètre de rectangles

Variables :
 Long, Larg, Perim (réels)
 Choix (chaîne) #pour refaire ou non l'exécution

Début
 #initialisation de l'indicateur pour entrer dans la boucle
 Choix ← "o"
 #boucle TantQue permettant de refaire le traitement selon le choix
 TantQue Choix = "o" Faire
 #traitement
 Afficher("Donner les dimensions du rectangle") et Saisir(Long, Larg)
 Perim ← 2 × (Long + Larg)
 Afficher("Le périmètre du rectangle est", Perim)
 #saisie du choix de recommencer ou non
 Afficher("Voulez-vous continuer (o/n) ?") et Saisir(Choix)
 FinTantQue
 Fin

Pseudo-Code

Python

En **python**, cela peut donner :

```
1 choix = "o"
2 while choix == "o" :
3     larg = float(input("Entrer la largeur du rectangle : "))
4     long = float(input("Entrer la longueur du rectangle : "))
5     perim = 2*(larg+long)
6     print(f"Le périmètre du rectangle est {perim}")
7     choix = input("Voulez-vous continuer (o/n) ? ")
```

Code Python

III.2. Comptage

Intro

On veut **compter** le nombre d'éléments d'un ensemble que l'on parcourt à l'aide d'une **boucle**.
Pour cela, on utilise une variable **Compteur** qui sera incrémentée à chaque exécution de la **boucle**.

Algorithme

On souhaite écrire un algorithme qui compte des notes entrées au clavier et qui s'arrête lorsqu'on saisit une note qui n'est pas comprise entre 0 et 20.

Pseudo-Code

```

Algorithme : Nombre de notes
Variables :
    Note (réel)
    Compteur (entier)

Début
    #on initialise le compteur à 0
    Compteur ← 0
    #saisie de la première note
    Afficher("Saisir une note") et Saisir(Note)
    #on exécute la boucle si la note est conforme
    TantQue (Note ≥ 0) et (Note ≤ 20) Faire
        #la note est conforme, on l'ajoute au Compteur
        Compteur ← Compteur + 1
        #on saisit une autre note
        Afficher("Saisir la note suivante ou un nb non compris entre 0 et 20 pour finir")
        Saisir(Note)
    FinTantQue
    #si la dernière note saisie n'est pas conforme, la boucle n'est pas exécutée
    #on sort, et cette dernière note n'est pas comptée
    #on affiche alors le compteur (nombre de notes)
    Afficher("Le nombre de notes est :", Compteur)
Fin
  
```

Python

En , cela peut donner :

Code Python

```

1  compteur = 0
2  note = int(input("saisir une note : "))
3
4  while note <=20 and note >=0 :
5      compteur = compteur+1
6      note=int(input("saisir la note suivante ..."))
7  print(f"le nombre de notes est {compteur}")
  
```


III.3. Accumulation

Algorithme

On veut ajouter au fur et à mesure des réels saisis au clavier. L'algorithme doit renvoyer la somme totale de ces réels.

Pour cela, on utilise une variable `Cumul` qui contiendra la somme des réels au fur et à mesure de leur saisie. On entrera 0 pour arrêter.

Pseudo-Code

```

Algorithme : Somme de réels
Variables :
    Nombre, Cumul (réel)

Début
    #on initialise la variable Cumul à 0
    Cumul ← 0
    #saisie du premier nombre
    Afficher("Saisir un nombre :") et Saisir(Nombre)
    TantQue (Nombre != 0) Faire
        Cumul ← Cumul + Nombre
        Afficher("Saisir le nombre suivant ou 0 pour arrêter")
        Saisir(Nombre)
    FinTantQue
    #affichage du résultat
    Afficher("La somme des nombres saisis est :", Cumul)
Fin
  
```

Python

En `Python`, cela peut donner :

Code Python

```

1 cumul = 0
2 nombre = float(input("saisir un nombre non nul : "))
3 while (nombre != 0) :
4     cumul = cumul+nombre
5     nombre = float(input("saisir le nombre suivant ou 0 pour stop : "))
6 print(cumul)
  
```

III.4. Exemple de boucles imbriquées

Algorithme - Exemple 1

On souhaite écrire un algorithme qui affiche les tables de multiplication de 2 à 9 (jusqu'à 10 × « table »).

Pseudo-Code

```


Algorithme : Tables de multiplication de 2 à 9
Variables :
    Table, i (entiers)

Début
    #début de la boucle numéro de la table
    Pour Table allant de 2 à 9 Faire
        Afficher("Table du", Table, ":")
        # début de la boucle du calcul de la table
        Pour i allant de 1 à 10 Faire
            Afficher(Table, " fois ", i, " = ", Table × i)
        FinPour # fin de la boucle i
    FinPour # fin de la boucle Table
Fin
  
```

Python

En  python, cela peut donner :

```
1 for table in range (2, 10):  
2     print(f"table du {table}")  
3     for i in range (1,11):  
4         print(f"{table} fois {i} = {table*i}")
```

 Code Python


Python - Exemple 2

Comparer les deux algorithmes suivants :

```
1 # Algorithme 1  
2 for truc in range(1,11) :  
3     print("bonjour")  
4     for machin in range(1,6) :  
5         print("hello")
```

 Code Python

```
1 # Algorithme 2  
2 for truc in range(1,11) :  
3     print("bonjour")  
4 for machin in range(1,6) :  
5     print("hello")
```

 Code Python