

Institut Préparatoire aux Études d'Ingénieurs de Tunis



Leila Ben Othman

Première année des classes préparatoires
MP-PC

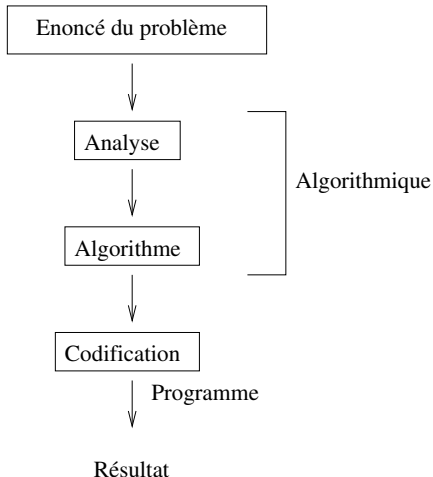
Année universitaire: 2016-2017

Éléments de base de l'Algorithmique: Application en PYTHON

Plan

- ① Introduction
- ② Objets algorithmiques
- ③ Types simples et opérateurs
- ④ Structure d'un algorithme
- ⑤ Actions élémentaires
- ⑥ Structures conditionnelles
 - Structure conditionnelle simple
 - Structure conditionnelle complète
 - Condition
 - Structures conditionnelles en PYTHON
- ⑦ Structures itératives
 - Structure itérative Tant que
 - Structure itérative Répéter jusqu'à
 - Structure itérative Pour
 - Structures itératives: PYTHON
- ⑧ Choix de la boucle en Algorithmique

Étapes de résolution d'un problème - (1)



Étapes de résolution d'un problème - (2)

- **Énoncé du problème** : la première chose à faire quand on aborde un problème, c'est de le cerner et de le définir précisément.
- **Analyse du problème** : cette étape consiste à :
 - ① Décomposer le problème en sous-problèmes plus simples à résoudre (fonction/procédure).
 - ② Associer à chaque sous problème, une spécification (les données nécessaires et les données résultantes).
- **Algorithme** : description des étapes d'analyse sous forme d'une suite d'actions convenablement enchaînées.
- **Codification** : traduire les actions en un langage compréhensible par la machine. Un programme est un algorithme codé dans un langage compréhensible par l'ordinateur à l'aide d'un compilateur (traducteur).

⇒ Le langage de programmation qu'on va utilisé est PYTHON. Dans ce cas, le traducteur PYTHON est appelé l'interpréteur.

Qu'est ce qu'un algorithme?

Définitions :

- Une suite d'instructions, qui une fois exécutées correctement, conduit à un résultat donné.
- Une séquence de pas de calcul qui prend un ensemble de valeurs comme entrée (input) et produit un ensemble de valeurs comme sortie (output).

Les objets algorithmiques

Un algorithme manipule des objets qui représentent des données ou des résultats. Ces objets peuvent être des constantes (valeur non modifiable) ou des variables (valeur modifiable), de type simple ou de type structuré.

Un objet est caractérisé par :

- **un identificateur** : est un moyen permettant d'identifier l'objet de façon unique au niveau de la mémoire de l'ordinateur. C'est une suite quelconque de caractères alphanumériques (sans espace) commençant obligatoirement par une lettre.
- **une nature** : constante ou variable
- **un type** : caractérisant l'ensemble des valeurs possibles (son domaine de définition) ainsi que les opérateurs applicables.

En PYTHON, on n'a pas besoin de spécifier le type d'un objet qu'on manipule car le typage en PYTHON est dynamique.

Identificateur - Exemple

- Les identificateurs suivants sont valables : Montant, ma_moyenne, note_1, note2.
- Par contre, ceux-ci ne le sont pas : 1note, quota?, chaîne, demi-mot, deux mots.

Constante

- Une constante est un objet dont la valeur reste inchangée tout le long d'un algorithme.
- Une constante est initialisée à la déclaration.
- Seule la consultation est autorisée sur une constante.

Déclaration d'une constante

identificateur_constant= valeur

- **Exemple** : $\pi=3.14$
- **Remarque** : par convention, on préfère noter les constantes en majuscules.

Constante - PYTHON

En PYTHON, on peut modifier la valeur de pi:

```
> import math as m
> m.pi
3.141592653589793
> m.pi=2
> m.pi
2
```

⇒ Les constantes n'existent pas en Python

- On utilise généralement des variables globales. Cependant, cela entraîne un risque de sécurité du fait que la valeur de ces variables peut être modifiée.
- On peut définir une classe spéciale qui permet de simuler une constante et interdit toute modification de sa valeur une fois créée.

Variable

- Une variable est un objet dont le contenu peut varier tout au long de l'algorithme : elle peut donc être consultée ou modifiée par une action.

Déclaration d'une variable

identificateur_valeur: type

- **Exemple** : moyenne: réel

Variable - PYTHON

- PYTHON est un langage orienté Objet
⇒ Tout est objet : données, fonctions, modules, etc
- En PYTHON, une variable est en réalité une référence (étiquette) vers un objet ⇒ Pour manipuler des objets et les distinguer l'un de l'autre, on utilise des variables. On dit qu'une variable fait référence à un objet.

Types simples

Un type permet d'indiquer l'ensemble des **valeurs possibles** ainsi que les **opérations** que l'on peut effectuer sur ce type.

Type	Valeurs possibles
Entier	Sous-ensemble des entiers relatifs
Réel	Sous-ensemble des nombres réels
Booléen	Vrai/Faux
Caractère	Chiffres, lettres (majuscules et minuscules) et les symboles spéciaux
Chaîne de caractères	Succession de caractères

Types simples - en PYTHON

- NoneType: seule valeur possible est **None**. C'est la valeur retournée par une fonction qui ne retourne rien;
- int: les nombres entiers;
- float: les nombres réels;
- complex: les nombres complexes;
- bool: **True**, **False**
- str: les chaînes de caractères

Les opérateurs

Opérateurs arithmétiques

Opérateur	Algorithmique	PYTHON
Addition	+	+
Soustraction	-	-
Multiplication	*	*
Puissance	^	**
Division	/	/
Division entière	<i>Div</i>	//
Reste de la division entière	<i>Mod</i>	%

Les opérateurs

Opérateurs relationnels

Opérateur	Algorithmique	PYTHON
strictement inférieur	$x < y$	$x < y$
strictement supérieur	$x > y$	$x > y$
supérieur ou égal	$x \geq y$	$x \geq y$
inférieur ou égal	$x \leq y$	$x \leq y$
égalité	$x = y$	$x == y$
différence	$x \neq y$	$x != y$

Les opérateurs

Opérateurs logiques

Opérateur		Algorithmique	PYTHON
Conjonction		Et	and
Disjonction		Ou	or
Négation		Non	not

Opérateurs logiques

Table de vérité

X	Y	NON(X)	X ET Y	X OU Y
V	V	F	V	V
V	F	F	F	V
F	V	V	F	V
F	F	V	F	F

Structure d'un algorithme

Algorithme nom_algorithme

CONST

Déclaration des constantes

VAR

Déclaration des variables

Debut

Corps de l'algorithme (Instructions)

Fin

Affectation

Principe : L'affectation permet d'attribuer une valeur à une variable. Cette valeur peut elle-même être une autre variable, le résultat d'une expression arithmétique ou logique ou une constante ou encore le résultat retournée par une fonction.

Syntaxe

variable 1 $\leftarrow$ expression

Une expression est un ensemble d'opérandes, reliées par des opérateurs.

Examples :

- $X \leftarrow 12.5$: la variable X reçoit la valeur 12.5 (**cette affectation sous-entend que la variable X est de type réel**).
- $Y \leftarrow X$: la variable Y reçoit la valeur de la variable X (**la valeur de X n'est pas modifiée**).
- $Z \leftarrow X * Y$ (l'expression $X * Y$ est évaluée et sa valeur est attribuée à la variable Z).

Affectation

- L'affectation ne vide pas la variable émettrice de sa valeur. Par contre, le contenu de la variable réceptrice est écrasé et remplacé par la nouvelle valeur.
- L'affectation d'une valeur à une variable peut être effectuée autant de fois que l'on veut au cours d'un algorithme. La valeur de la variable sera alors modifiée à chaque affectation.
- La variable réceptrice d'une affectation doit être de même type que la valeur à affecter ou de type compatible. Le type est dit compatible s'il est inclus dans le type de la variable réceptrice. Exemple : REEL $\leftarrow$ ENTIER est possible mais pas l'inverse.
 $\Rightarrow$ Ceci n'est plus valable en PYTHON car il est dynamiquement typé.

Affectation - PYTHON

Syntaxe

```
variable = expression
```

Une affectation $x=5$, implique:

- la création d'un objet de type entier (int) avec une valeur égale à 5.
- Le référencement de cet objet par la variable x.

Affectation - PYTHON

> x=4; # affectation simple

> x=y=z=4; # affectation multiple

> x,y,z=4,5,6; # affectation parallèle

> x,y=y,x; # permutation

Entrée-Sortie

Instruction d'écriture (Sortie)

L'instruction **Ecrire** permet d'afficher un message ou un résultat à l'utilisateur.

- Ecrire("Bonjour")
- Ecrire(moyenne)
- Ecrire("Votre moyenne est", moyenne)

Entrée-Sortie - PYTHON

Affichage avec la fonction print()

```
In [1]: print('Bonjour');
```

```
Out [1]: Bonjour
```

```
In [2]: print("Bonjour");
```

```
Out [2]: Bonjour
```

```
In [3]: print('Bonjour \n les futurs ingénieurs')
```

```
Out [3]: Bonjour
```

```
les futurs ingénieurs
```

Entrée-Sortie - PYTHON

Affichage avec la fonction print()

```
In [1]: x=4;y=5;y=6print(x,y,z)
```

Out [1]: 4 5 6

```
In [2]: print(x,y,z,sep='::')
```

Out [2]: 4::5::6

```
In [3]: print(x,y,z,sep='\n')
```

Out [3]: 4

5

6

```
In [4]: print(x,y,z,sep='::', end=';')
```

Out [4]: 4::5::6;

Entrée-Sortie

Instruction de lecture (Entrée)

L'instruction **Lire** permet à l'utilisateur d'introduire des valeurs pour qu'elles soient utilisées par le programme.

- Lire(X)

Remarques:

- Dès que le programme rencontre une instruction **Lire**, il cède la main à l'utilisateur pour qu'il introduise une valeur au clavier.
- Lire(X) $\Rightarrow$ introduction d'une valeur par l'utilisateur et affectation de cette dernière valeur à la variable X .

Entrée-sortie - PYTHON

La saisie avec la fonction input()

```
In [1]: x=input('introduire une valeur')
```

Out [1]: introduire une valeur15

```
In [2]: type(x)
```

Out [2]: `<class 'str'>`

⇒ Convertir la saisie en type adéquat avec les fonctions `int()`, `float()` ou `complex()`

Application 1

Écrire un algorithme qui permet de lire :

- Le montant hors taxes d'un article : Montant_{HT} (un réel)
- Le taux de TVA : Taux_{TVA} (un réel)
- Le nombre d'articles : NbArticles (un entier)

puis affiche le montant total toutes taxes comprises **TotalTTC** sachant que **TotalTTC = Nbarticles * MontantHT * (1 + TauxTVA)**

Application 1

Algorithme Application_1

VAR

MontantHT, TauxTVA, TotalTTC: réel

Nbarticles: entier

Début

Ecrire("Donner le nombre d'articles")

Lire(Nbarticles)

Ecrire("Donner le montant hors taxe")

Lire(montantHT)

Ecrire("Donner le taux de TVA")

Lire(TauxTVA)

 TotalTTC $\leftarrow$ Nbarticles * MontantHT * (1 + TauxTVA)

Ecrire("Le montant total Toutes taxes comprises est ", TotalTTC)

FIN

Structures conditionnelles

En algorithmique, il existe deux structures possibles pour un test :

- ① La structure conditionnelle simple.
- ② La structure conditionnelle complète.

Structure conditionnelle simple

Syntaxe

Si condition Alors
Traitement
FinSi

Description

- Si la condition est vérifiée (sa valeur est égale à vrai), alors on exécute la partie traitement
- Si la condition a pour valeur faux, alors il n'y aura pas exécution de la partie traitement et le programme passe directement à l'instruction située après le **FinSI**.

Structure conditionnelle complète

Syntaxe

Si condition Alors

Traitement 1

Sinon

Traitement 2

FinSi

Description

- Si la condition est vérifiée, alors on exécute la partie traitement 1
- Au moment où le programme arrive à **SINON**, il passe directement à l'instruction située après le **FinSi**.
- Si la condition a pour valeur faux, alors le programme exécute la partie traitement 2 située après le **SINON**.

Condition

- Une condition est une expression logique qui a pour résultat une valeur logique (Vrai ou Faux).
- Exemples :
 - $X \geq 5$
 - $chaîne = "mimi"$
 - $Y <> 7$
 - $A = vrai$
- Notons que $(A=vrai) \Leftrightarrow (A)$ et $(A=Faux) \Leftrightarrow Non(A)$.

⇒ Remarquons donc qu'une condition emploie nécessairement un opérateur de comparaison (opérateur relationnel).

Condition composée

- Comment exprimer la condition : **X est compris entre 5 et 8?**
- La condition s'exprime comme suit :
$$(X \geq 5) \textbf{ ET } (X \leq 8)$$
- Condition composée: Il s'agit de deux ou plusieurs conditions simples reliées par un opérateur logique.

Structure conditionnelle imbriquée

Structure imbriquée

Si condition 1 **Alors**

Traitement 1

Sinon

Si condition 2 **Alors**

Traitement 2

Sinon

Traitement 3

FinSi

FinSi

Les structures conditionnelles

- Une structure conditionnelle en PYTHON est définie par son indentation.

```

if condition:
    .....
    Bloc d'instructions 1
    .....
else:
    .....
    Bloc d'instructions 2
    .....

```

```
# script qui teste la parité d'un entier
x=int(input('Donner un entier'))
if x%2==0 :
    print('Entier pair')
else:
    print('Entier impair')
```

Les structures conditionnelles

- Les structures conditionnelles en PYTHON peuvent être imbriquées et par conséquent, elles doivent être indentées correctement.
- Il est possible d'utiliser le mot-clé `elif` pour combiner `else-if`.

```
if condition1:  
    Bloc d'instructions 1  
else:  
    if condition 2:  
        Bloc d'instructions 2  
    else:  
        Bloc d'instructions 3
```

```
if condition1:  
    Bloc d'instructions 1  
elif condition 2:  
    Bloc d'instructions 2  
else:  
    Bloc d'instructions 3
```

La structure itérative **Tant que** (1)

- Ce type de schéma permet l'exécution d'un traitement donné plusieurs fois tant qu'une condition spécifiée est vraie.
- La condition est une expression logique qui peut être simple ou composée.
- Ce type de schéma est utilisé chaque fois que le nombre d'itérations qui seront exécutées n'est pas connu à l'avance.

Syntaxe

Tant que Condition **Faire**
 Traitement
Fin Tant que

Description

- Test de la valeur de la condition : si elle est vérifiée, alors exécution de la partie traitement et retour au test;
- Sinon arrêt de la boucle.

La structure itérative **Tant que** (2)

- ① Les paramètres de la condition doivent être initialisés avant la boucle.
- ② La partie traitement peut ne jamais être exécutée $\Rightarrow$ c'est lorsque la condition n'est pas vérifiée dès le départ.
- ③ Il doit y avoir une instruction dans la partie traitement qui modifie la valeur de la condition $\Rightarrow$ boucle infinie !!!

La structure itérative **Tant que** (3)

Que se passe t-il?

Exemple 1:

$i \leftarrow 12$

Tant que $i \leq 10$ **Faire**

Ecrire($i*2$)

$i \leftarrow i + 1$

Fin Tant que

Exemple 2 :

$i \leftarrow 1$

Tant que $i \leq 10$ **Faire**

Ecrire($i*2$)

Fin Tant que

La structure itérative **Tant que** (4)

Que se passe t-il?

Exemple 3 :

$i \leftarrow 1$

Tant que $i \leq 10$ **Faire**

Ecrire($i*2$)

$i \leftarrow i + 1$

Fin Tant que

Exemple 4 :

Ecrire("donner un entier strictement positif")

Lire(i)

Tant que $i < 0$ **Faire**

Ecrire("donner un entier strictement positif")

Lire(i)

Fin Tant que

La structure itérative **Tant que** (5)

- Lorsqu'une boucle contient une autre boucle, on parle dans ce cas de **boucles imbriquées**.
- Étant données deux boucles, seulement deux situation peuvent se présenter : les deux boucles peuvent être imbriquées ou successives
⇒ **elle ne doivent jamais être croisées!**

La structure itérative **Tant que** (6)

Application : Écrire un algorithme qui saisit un nombre pair et qui détermine combien de fois ce nombre est divisible par 2. Exemple : Le nombre 8 est divisible 3 fois par 2 (Ne pas contrôler la parité du nombre).

La structure itérative **Tant que** (7)

Algorithme DIV2

VAR

NB, N : entier

Début

Ecrire("Introduire un nombre pair")

Lire(NB)

$N \leftarrow 0$

Tant que (NB Mod 2=0) Faire

$NB \leftarrow NB \text{ Div } 2$

$N \leftarrow N + 1$

Fin Tant que

Ecrire ("Le nombre que vous avez introduit est divisible par 2", N, "fois")

FIN

La structure itérative **Répéter jusqu'à** (1)

- Ce type de schéma permet l'exécution d'un traitement donné plusieurs fois jusqu'à une condition soit vraie.
- Ce type de schéma est utilisé à chaque fois que le nombre d'itérations n'est pas connu à l'avance.

Syntaxe

Répéter

Traitement

Jusqu'à Condition

Description

- 1 Quelque soit la valeur de la condition, il y a exécution de la partie traitement
- 2 Test de la valeur de la condition
 - si la condition est vérifiée, alors la boucle s'arrête
 - sinon reexécution de la partie traitement

La structure itérative **Répéter jusqu'à** (2)

- A la différence du schéma **Tant que**, la condition n'est évaluée qu'après avoir exécuté une fois la partie traitement $\Rightarrow$ Le traitement est effectué au moins une fois.
- La condition est évaluée après chaque passage dans la boucle.

La structure itérative **Répéter jusqu'à** (3)

Exemple 1:

$i \leftarrow 1$

Répéter

Ecrire($i*2$)

$i \leftarrow i + 1$

Jusqu'à ($i > 10$)

Exemple 2:

$i \leftarrow 10$

Répéter

Ecrire($i*2$)

$i \leftarrow i + 1$

Jusqu'à ($i > 5$)

La structure itérative **Répéter jusqu'à** (4)

Exemple 3:

$i \leftarrow 1$

Répéter

Ecrire($i*2$)

Jusqu'à ($i > 5$)

La structure itérative **Répéter jusqu'à** (5)

Application : Traiter l'application précédente en effectuant le contrôle sur la parité du nombre et en utilisant la boucle **Répéter jusqu'à**.

La structure itérative **Répéter jusqu'à** (5)

Algorithme DIV2

VAR

NB, N : entier

Début

Répéter

Ecrire("Introduire un nombre pair")

Lire(NB)

jusqu'à (NB Mod 2=0)

$N \leftarrow 0$

Répéter

$NB \leftarrow NB \text{ Div } 2$

$N \leftarrow N + 1$

Jusqu'à (NB Mod 2 $\neq$ 0)

Ecrire ("Le nombre que vous introduit est divisible par 2", N, "fois")

FIN

La structure itérative **Pour** (1)

- Ce schéma itératif permet de répéter un traitement donné un certain nombre de fois connu à l'avance.

Syntaxe

Pour compteur de *Valeur_initiale* à *Valeur_finale* **Faire**
 Traitement
Fin Pour

Description

- Au début de la boucle, il y a initialisation du compteur par *Valeur_initiale*;
- Exécution de la partie traitement et incrémentation du compteur;
- Si le compteur ne dépasse pas *Valeur_finale*, alors réexécution de la partie traitement;
- Sinon, arrêt de la boucle.

La structure itérative **Pour** (2)

Attention :

Pour i de 1 à 5

Ecrire(i)

$i \leftarrow i * 2$

Fin Pour

⇒ Le traitement ne doit pas modifier le compteur.

Les structures itératives

La boucle **Pour** est traduite en PYTHON avec la boucle **for** (avec la fonction **range**).

La fonction range est employée avec un, deux ou trois paramètres:

- `range(b)` énumère les entiers $0, 1, 2, \dots, b-1$ ($\text{pas}=1$)
- `range(a,b)` énumère les entiers $a, a+1, a+2, \dots, b-1$ ($\text{pas}=1$)
- `range(a,b,c)` énumère les entiers $a, a+c, a+2c, \dots, a+nc$, où n est le plus grand entier vérifiant $a + nc < b$ ($\text{pas}=c$)

Les structures itératives

```
for i in range(5):  
    print(i)
```

0
1
2
3
4

```
for i in range(2,5):  
    print(i)
```

2
3
4

```
for i in range(2,10,3):
    print(i)
```

258

Les structures itératives

La boucle **Tant que** est traduite en PYTHON avec la boucle **while**.

```
while condition:
```

.....
Bloc d'instructions
.....

```
i=0
while i<=5:
    print(i)
    i+=1
```

0
1
2
3
4
5

L'instruction break

- L'instruction *break* dans une structure itérative (for ou while) permet l'arrêt immédiat de la boucle dans laquelle se trouve l'instruction. Ainsi:
 - l'instruction *break* a presque le même effet que l'utilisation d'une variable booléenne pour sortir d'une boucle while (exemples 1 & 2).
 - l'instruction *break* rend possible l'arrêt d'une boucle for (exemple 3)
⇒ ceci n'est pas recommandé!!! Si vous avez un traitement itératif qui doit s'arrêter si une condition est vérifiée, il est plus adéquat d'utiliser une boucle while.
- On l'utilise avec la syntaxe suivante au sein d'une boucle:

```
for i in range(a,b,c):
    if condition:
        break
```

```
while condition_boucle:
    if condition:
        break
```



```
for x in range(10,0,-1):
    print(x)
    if x == 5:
        break
print( "Au revoir , x=", x)
```

Au revoir, $x=5$

- while 10>5:
- while True:
- while 1:

NB: Évitez l'utilisation de la boucle while de cette manière!!!

exemple 4

```
x=10
while True:
    print(x)
    if x == 5:
        break
    x=x-1
print("Au revoir , x=",x)
```

10

9

8

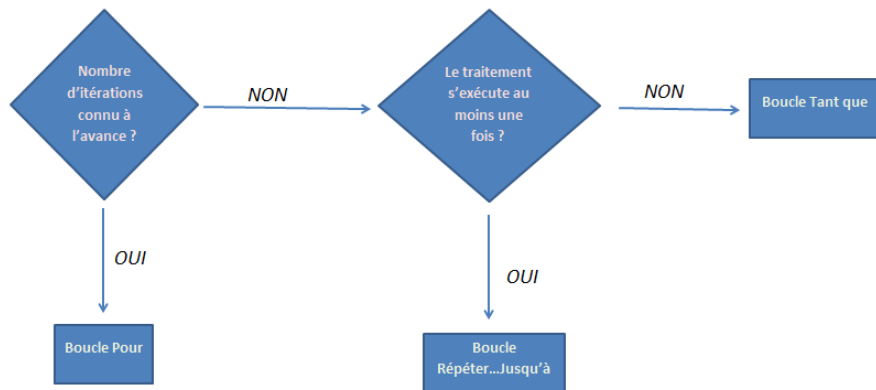
7

6

5

Au revoir, x=5

Choix de la boucle en Algorithmique (1)



Choix de la boucle en en Algorithmique (2)

- On peut facilement passer d'une boucle **Pour** aux autres boucles. Mais, l'inverse, n'est pas possible.

Pour cpt de VI à VF Faire
Traitements
Fin Pour

$cpt \leftarrow VI$
Répéter
Traitements
 $cpt \leftarrow cpt + 1$
Jusqu'à ($cpt > VF$)

$cpt \leftarrow VI$
Tant que $cpt \leq VF$ Faire
Traitements
 $cpt \leftarrow cpt + 1$
Fin Tant que