Algorithmique I

Module : Fondement de la programmation

Douglas Teodoro

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

2018-2019

Sommaire

Objective

- Maîtriser les structures algorithmique interactives
  - tant que .. faire
  - pour .. faire
- Mettre en œuvre quelques structures pour résoudre des problèmes simples

Sommaire

RAPPEL

Saisie et affichage

- la pseudo-instruction `afficher` (ou `output`) est utilisée pour simuler l'affichage d'un texte ou d'une valeur sur l'écran
- elle prend à sa suite une valeur numérique, une chaîne de texte ou une variable

```
afficher ...
```

Python - output

```
print(...)
```

Saisie

- la pseudo-instruction `saisir` (ou `input`) est utilisée pour inviter un utilisateur à rentrer au clavier une valeur
- elle prend à sa suite une variable

```
saisir ...
```

- après exécuter cette instruction, l'programme attendra une entrée au clavier qui sera validée avec la touche d'entrée
- la valeur saisie sera affectée à la variable indiquée à la suite de `saisir`

Python - input

```
... = input()
```

Les opérateurs de comparaison

Condition : comparaison

égalité	x	==	y
différence	x	!=	y
inférieur	x	<	y
inférieur ou égal	x	<=	y
supérieur	x	>	y
supérieur ou égal	x	>=	y

```
valeur 1 OPERATEUR valeur 1
```

Quasiment tout peut être testé en algorithmique (et en programmation) : les opérateurs s'appliquent sur quasiment tous les types de données, y compris les chaînes de caractères

Expression composée

Expression composée

Plusieurs expressions peut être liées entre elles à l'aide d'un opérateur booléen, éventuellement regroupées avec des parenthèses pour en modifier la priorité

Expression composée

$$(a == 1 \texttt{ OR } (b \times 3 == 6)) \texttt{ AND } c > 10$$

Celle-ci sera vraie si (a vaut 1 et c est supérieur à 10) ou si (b vaut 2 et c est supérieur à 10)

Tests

Structure de contrôle effectuant un test et permettant un choix entre diverses parties du programme

Structure algorithmique

```
si condition alors bloc
[else si condition bloc]*
[sinon bloc]
```

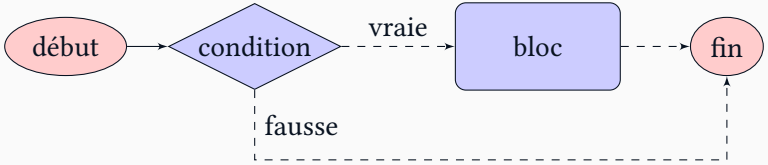
l'instruction entre crochets […] est optionnel
* signifie que l'instruction peut être répétée 0 ou plusieurs fois

On sort ainsi de l'exécution purement séquentielle des instructions

Forme simple

Structure algorithmique

```
si condition alors bloc
```



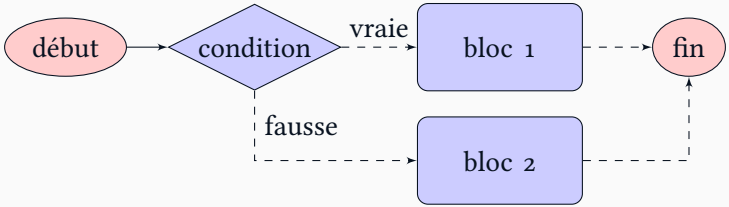
Notez ici que la condition (ou expression) est du type booléenne

La condition peut aussi être représentée par une seule variable : souvent, si elle contient 0, elle représente le booléen `FALSE`, sinon le booléen `TRUE`

Forme alternative simple

Structure algorithmique

```
si condition alors bloc 1
sinon bloc 2
```



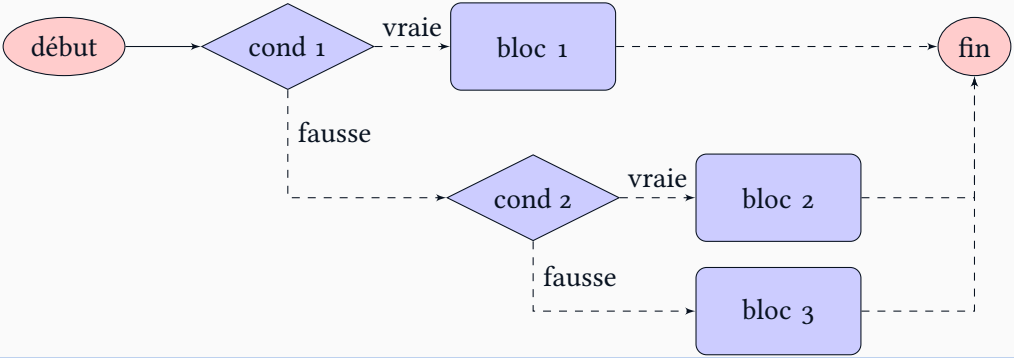
La forme alternative simples est utilisé dans le cas où il faut exécuter quelques instructions si la condition est fausse

Alternative multiple ou test imbriqué

Structure algorithmique

```
si condition 1 alors bloc 1
sinon si condition 2 alors bloc 2*
sinon bloc 3
```

* signifie que l'instruction peut être répétée 0 ou plusieurs fois



Structure du pseudo-code pour le test

Algorithme : Test

```
// séquence d'opérations
...
si condition 1 alors
    instruction 1
    ...
sinon si condition 2 alors
    instruction 2
    ...
sinon
    instruction 3
    ...
```

Python - Test

```python
#séquence d'opérations
if condition_1:
    instruction_1
    ...
elif condition_2:
    instruction_2
    ...
else:
    instruction_3
    ...
```

La loi AND (ET)

énoncé

a ET *b* est VRAI si et seulement si *a* est VRAI **et** *b* est VRAI

La loi AND est aussi appelée la conjonction

Notation de la loi AND :

- $\cdot$ (le point) : $a \cdot b$
- $\wedge$: $a \wedge b$
- & , && ou *and* en programmation, selon les langages

AND		
a \ b	**0**	**1**
0	0	0
1	0	1

Associé à la multiplication arithmétique : $0.a = 0$

La loi OR (OU)

énoncé

a `OU` *b* `est VRAI si et seulement si` *a* `est VRAI` **ou** *b* `est VRAI`

La loi OR est aussi appelée la disjonction

Notation de la loi OR :

- $+$ (signe plus) : $a + b$
- $\vee$: $a \vee b$
- $|$, $||$ ou *or* selon les langages

OR		
a \ b	**0**	**1**
0	0	1
1	1	1

Associé à la addition arithmétique : $1 + a = 1$

La loi NOT (négation)

énoncé

la négation de a est VRAI si et seulement si a est FAUX

Notation de la loi AND :

- $\bar{}$: $\bar{a}$
- $\neg$: $\neg a$
- ! ou *not* selon les langages

Propriétés :

- $a + \bar{a} = 1$
- $a \cdot \bar{a} = 0$

NOT		
a\ a	**0**	**1**
0	1	
1		0

Sommaire

LA PRÉCÉDENCE DES OPÉRATEURS

Précédence

Les opérateurs n'ont pas tous la même priorité

- la priorité (ou précédence) des opérateurs définit l'ordre dans lequel l'expression doit être analysée
- si deux opérateurs ont la même priorité, ils peuvent être évalués de gauche à droite (c'est cas le plus courant) ou de droite à gauche

`var_1` **`op1`** `var_2` **`op2`** $\texttt{var}_3$

Précédence

Les opérateurs n'ont pas tous la même priorité

- la priorité (ou précédence) des opérateurs définit l'ordre dans lequel l'expression doit être analysée
- si deux opérateurs ont la même priorité, ils peuvent être évalués de gauche à droite (c'est cas le plus courant) ou de droite à gauche

`var_1` **`op1`** `var_2` **`op2`** $\texttt{var}_3$

Multiplication

une multiplication est prioritaire sur une addition : $3 + 2 \times 2 = 7$

Précédence

Les opérateurs n'ont pas tous la même priorité

- la priorité (ou précédence) des opérateurs définit l'ordre dans lequel l'expression doit être analysée
- si deux opérateurs ont la même priorité, ils peuvent être évalués de gauche à droite (c'est cas le plus courant) ou de droite à gauche

`var_1` **`op1`** `var_2` **`op2`** $\texttt{var}_3$

Multiplication

une multiplication est prioritaire sur une addition : $3 + 2 \times 2 = 7$

Chaque langage dispose de sa propre table de précédence : pour contourner ces problèmes, on utilise les parenthèses !

Précédence - Python

Opérateurs	Description	Priorité
`x or y`	ou logique	− − −
`x and y`	et logique	−−
`not x`	négation logique	−
`<, <=, >, >=, ==, <>, !=`	opérateurs de comparaison	...
`x + y, x - y`	addition ou concaténation/soustraction	...
`x * y`	multiplication ou répétition	+
`x / y, x % y`	division/reste de la div. (modulo)	++
`-x`	négation unaire	+ + +

Exercice - Precedence I

Question : Quel est l'affichage du programme `alg_booleen` si

- `var_1 = TRUE`
- `var_2 = FALSE`
- `var_3 = TRUE`

A) TRUE, TRUE, FALSE, FALSE
B) FALSE, TRUE, FALSE, TRUE
C) TRUE, TRUE, TRUE, FALSE
D) TRUE, TRUE, TRUE, TRUE

Algorithme : alg_booleen

Données : a,b,c : booleen

```
1  si (var_1 AND var_2) OR var_3 alors
2  |  afficher "TRUE"
3  sinon
4  |  afficher "FALSE"
5  si var_1 AND (var_2 OR var_3) alors
6  |  afficher "TRUE"
7  sinon
8  |  afficher "FALSE"
9  si NOT var_1 OR var_3 alors
10 |  afficher "TRUE"
11 sinon
12 |  afficher "FALSE"
13 si NOT (var_1 OR var_3) alors
14 |  afficher "TRUE"
15 sinon
16 |  afficher "FALSE"
```

Exercice - Precedence I

Question : Quel est l'affichage du programme `alg_booleen` si

- `var_1 = TRUE`
- `var_2 = FALSE`
- `var_3 = TRUE`

A) TRUE, TRUE, FALSE, FALSE
B) FALSE, TRUE, FALSE, TRUE
C) TRUE, TRUE, TRUE, FALSE
D) TRUE, TRUE, TRUE, TRUE

Algorithme : alg_booleen

Données : a,b,c : booleen

```
1  si (var_1 AND var_2) OR var_3 alors
2  |  afficher "TRUE"
3  sinon
4  |  afficher "FALSE"
5  si var_1 AND (var_2 OR var_3) alors
6  |  afficher "TRUE"
7  sinon
8  |  afficher "FALSE"
9  si NOT var_1 OR var_3 alors
10 |  afficher "TRUE"
11 sinon
12 |  afficher "FALSE"
13 si NOT (var_1 OR var_3) alors
14 |  afficher "TRUE"
15 sinon
16 |  afficher "FALSE"
```

Exercice - Precedence II

Question : Quel c'est l'affichage du programme precedence si exécuté en python ?

A) FALSE et FALSE
B) FALSE et TRUE
C) TRUE et FALSE
D) TRUE et TRUE

precedence - Python

```
var_1 = 0
var_2 = 1
var_3 = 1

if var_1 and var_2 + var_3:
    print("True")
else:
    print("False")

if (var_1 and var_2) + var_3:
    print("True")
else:
    print("False")
```

Exercice - Precedence II

Question : Quel c'est l'affichage du programme `precedence` si exécuté en python ?

A) FALSE et FALSE
B) FALSE et TRUE
C) TRUE et FALSE
D) TRUE et TRUE

precedence - Python

```
var_1 = 0
var_2 = 1
var_3 = 1

if var_1 and var_2 + var_3:
    print("True")
else:
    print("False")

if (var_1 and var_2) + var_3:
    print("True")
else:
    print("False")
```

Sommaire

STRUCTURES INTERACTIVES

Les boucles

Définition

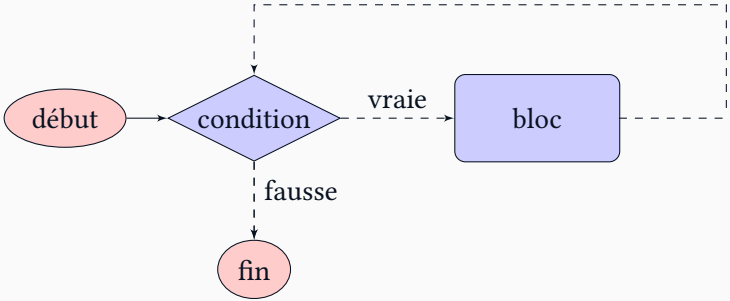
Les boucles sont des structures itératives : une itération ou structure itérative est une séquence d'instructions destinée à être exécutée plusieurs fois

- tant que (while) : `tant que condition faire`
- pour (for) : `pour element ←debut à fin faire`

Principe

- La boucle est un élément très simple au premier abord
- Le but d'une boucle est de répéter un bloc d'instructions plusieurs fois
- Selon le type de boucle, le bloc d'instructions va être répété un nombre fixe de fois (n fois) ou selon un certain nombre de critères (un test de une ou plusieurs conditions)

Structure algorithmique - Tant que (While)



Tant Que - Principe

La boucle de type `tant que` permet la répétition d'un bloc d'instructions tant que la condition testée est vraie

Structure algorithmique

```
tant que condition faire bloc
```

Tant Que - Principe

La boucle de type `tant que` permet la répétition d'un bloc d'instructions tant que la condition testée est vraie

Structure algorithmique

```
tant que condition faire bloc
```

Lors de l'exécution du programme, quand celui-ci arrive sur l'instruction `tant que`

- il évalue la condition

Tant Que - Principe

La boucle de type `tant que` permet la répétition d'un bloc d'instructions tant que la condition testée est vraie

Structure algorithmique

```
tant que condition faire bloc
```

Lors de l'exécution du programme, quand celui-ci arrive sur l'instruction `tant que`

- il évalue la condition
- si l'expression retourne TRUE, le programme exécute le bloc d'instructions suivante

Tant Que - Principe

La boucle de type `tant que` permet la répétition d'un bloc d'instructions tant que la condition testée est vraie

Structure algorithmique

```
tant que condition faire bloc
```

Lors de l'exécution du programme, quand celui-ci arrive sur l'instruction `tant que`

- il évalue la condition
- si l'expression retourne TRUE, le programme exécute le bloc d'instructions suivante
- à la fin du bloc, il remonte au `tant que` et évalue de nouveau l'expression booléenne

Tant Que - Principe

La boucle de type `tant que` permet la répétition d'un bloc d'instructions tant que la condition testée est vraie

Structure algorithmique

```
tant que condition faire bloc
```

Lors de l'exécution du programme, quand celui-ci arrive sur l'instruction `tant que`

- il évalue la condition
- si l'expression retourne TRUE, le programme exécute le bloc d'instructions suivante
- à la fin du bloc, il remonte au `tant que` et évalue de nouveau l'expression booléenne
- si c'est TRUE il exécute de nouveau le bloc d'instructions, et ainsi de suite, tant que l'expression retourne TRUE

Tant Que - Principe

La boucle de type `tant que` permet la répétition d'un bloc d'instructions tant que la condition testée est vraie

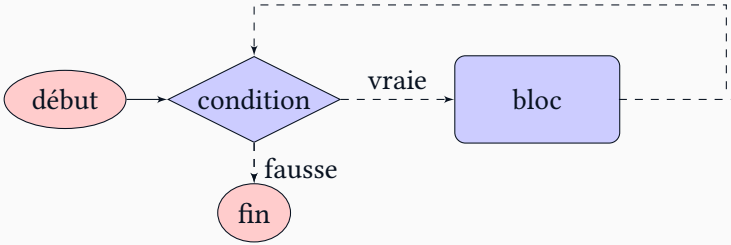
Structure algorithmique

```
tant que condition faire bloc
```

Lors de l'exécution du programme, quand celui-ci arrive sur l'instruction `tant que`

- il évalue la condition
- si l'expression retourne TRUE, le programme exécute le bloc d'instructions suivante
- à la fin du bloc, il remonte au `tant que` et évalue de nouveau l'expression booléenne
- si c'est TRUE il exécute de nouveau le bloc d'instructions, et ainsi de suite, tant que l'expression retourne TRUE
- si l'expression devient FALSE, après l'évaluation de la condition, le programme saute à l'instruction située juste après le bloc d'instruction `tant que`

Structure algorithmique - Tant que (While)



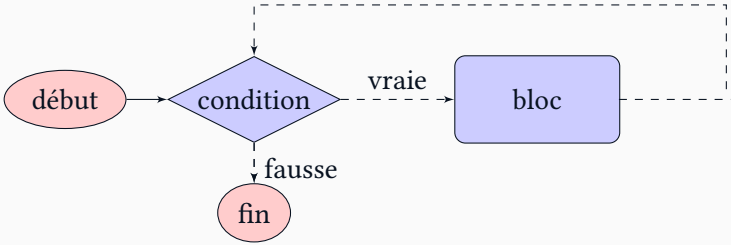
Acheter un iPhone

```
condition = avoir CHF 1500
total = CHF 0 (mois 0)
```

```
tant que total < 1500 faire
```

1. mois 1 : épargner CHF 300 (`total 300`)

Structure algorithmique - Tant que (While)



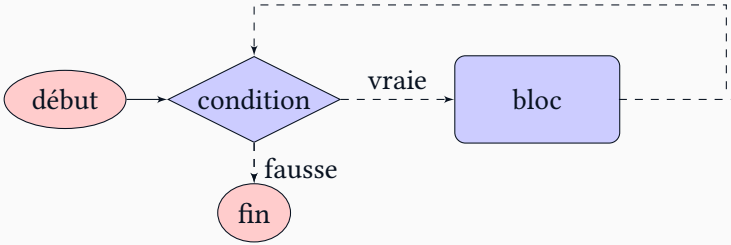
Acheter un iPhone

```
condition = avoir CHF 1500
total = CHF 0 (mois 0)
```

```
tant que total < 1500 faire
```

1. mois 1 : épargner CHF 300 (`total 300`)
2. mois 2 : épargner CHF 300 (`total 600`)

Structure algorithmique - Tant que (While)



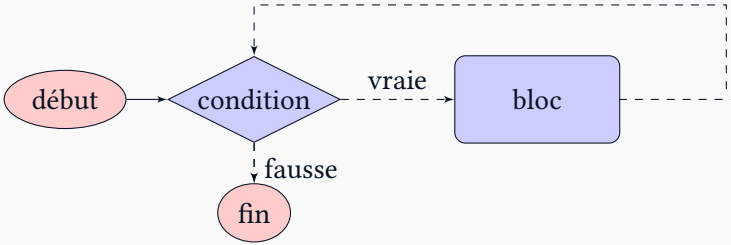
Acheter un iPhone

```
condition = avoir CHF 1500
total = CHF 0 (mois 0)
```

```
tant que total < 1500 faire
```

1. mois 1 : épargner CHF 300 (`total 300`)
2. mois 2 : épargner CHF 300 (`total 600`)
3. mois 3 : épargner CHF 300 (`total 900`)

Structure algorithmique - Tant que (While)



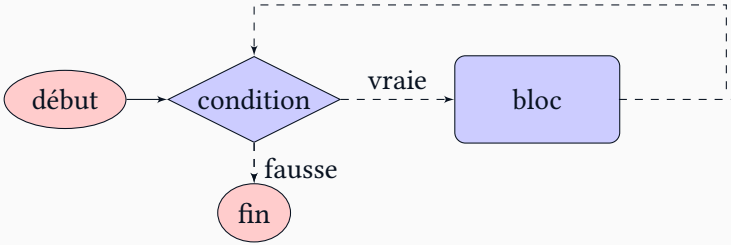
Acheter un iPhone

```
condition = avoir CHF 1500
total = CHF 0 (mois 0)
```

```
tant que total < 1500 faire
```

1. mois 1 : épargner CHF 300 (`total 300`)
2. mois 2 : épargner CHF 300 (`total 600`)
3. mois 3 : épargner CHF 300 (`total 900`)
4. mois 4 : épargner CHF 300 (`total 1200`)

Structure algorithmique - Tant que (While)



Acheter un iPhone

```
condition = avoir CHF 1500
total = CHF 0 (mois 0)
```

```
tant que total < 1500 faire
```

1. mois 1 : épargner CHF 300 (`total 300`)
2. mois 2 : épargner CHF 300 (`total 600`)
3. mois 3 : épargner CHF 300 (`total 900`)
4. mois 4 : épargner CHF 300 (`total 1200`)
5. mois 5 : épargner CHF 300 (`total 1500`)

Autre exemple

Cas où un utilisateur doit répondre à une question parmi une liste de réponses imposées comme oui ou non : si l'utilisateur répond autre chose (n'importe quoi), il faut lui reposer la question, jusqu'à ce qu'il réponde vraiment oui ou non

Autre exemple

Cas où un utilisateur doit répondre à une question parmi une liste de réponses imposées comme oui ou non : si l'utilisateur répond autre chose (n'importe quoi), il faut lui reposer la question, jusqu'à ce qu'il réponde vraiment oui ou non

Demander une réponse

```
condition = réponse oui ou non
réponse = ?
```

```
tant que réponse ≠ oui et réponse ≠ non
faire
```

- pas 1 : afficher "Voulez vous un café ?"

Autre exemple

Cas où un utilisateur doit répondre à une question parmi une liste de réponses imposées comme oui ou non : si l'utilisateur répond autre chose (n'importe quoi), il faut lui reposer la question, jusqu'à ce qu'il réponde vraiment oui ou non

Demander une réponse

```
condition = réponse oui ou non
réponse = ?
```

```
tant que réponse ≠ oui et réponse ≠ non
faire
```

- pas 1 : afficher "Voulez vous un café ?"
- pas 1 : saisir réponse (réponse "peut-être")

Autre exemple

Cas où un utilisateur doit répondre à une question parmi une liste de réponses imposées comme oui ou non : si l'utilisateur répond autre chose (n'importe quoi), il faut lui reposer la question, jusqu'à ce qu'il réponde vraiment oui ou non

Demander une réponse

```
condition = réponse oui ou non
réponse = ?
```

```
tant que réponse ≠ oui et réponse ≠ non
faire
```

- pas 1 : afficher "Voulez vous un café ?"
- pas 1 : saisir réponse (réponse "peut-être")
- pas 2 : afficher "Voulez vous un café ?"

Autre exemple

Cas où un utilisateur doit répondre à une question parmi une liste de réponses imposées comme oui ou non : si l'utilisateur répond autre chose (n'importe quoi), il faut lui reposer la question, jusqu'à ce qu'il réponde vraiment oui ou non

Demander une réponse

```
condition = réponse oui ou non
réponse = ?
```

```
tant que réponse ≠ oui et réponse ≠ non
faire
```

- pas 1 : afficher "Voulez vous un café ?"
- pas 1 : saisir réponse (réponse "peut-être")
- pas 2 : afficher "Voulez vous un café ?"
- pas 2 : saisir réponse (réponse "j'sais pas")

Autre exemple

Cas où un utilisateur doit répondre à une question parmi une liste de réponses imposées comme oui ou non : si l'utilisateur répond autre chose (n'importe quoi), il faut lui reposer la question, jusqu'à ce qu'il réponde vraiment oui ou non

Demander une réponse

```
condition = réponse oui ou non
réponse = ?
```

```
tant que réponse ≠ oui et réponse ≠ non
faire
```

- pas 1 : afficher "Voulez vous un café ?"
- pas 1 : saisir réponse (réponse "peut-être")
- pas 2 : afficher "Voulez vous un café ?"
- pas 2 : saisir réponse (réponse "j'sais pas")
- pas 3 : afficher "Voulez vous un café ?"

Autre exemple

Cas où un utilisateur doit répondre à une question parmi une liste de réponses imposées comme oui ou non : si l'utilisateur répond autre chose (n'importe quoi), il faut lui reposer la question, jusqu'à ce qu'il réponde vraiment oui ou non

Demander une réponse

```
condition = réponse oui ou non
réponse = ?
```

```
tant que réponse ≠ oui et réponse ≠ non
faire
```

- pas 1 : afficher "Voulez vous un café ?"
- pas 1 : saisir réponse (réponse "peut-être")
- pas 2 : afficher "Voulez vous un café ?"
- pas 2 : saisir réponse (réponse "j'sais pas")
- pas 3 : afficher "Voulez vous un café ?"
- pas 3 : saisir réponse (réponse "oui")

Cas particulier - à éviter !

- le bloc d'instructions d'une boucle tant que peut ne jamais être exécuté (condition non vérifiée la première fois)
  Exemple :

```
i = 0
tant que i > 0 faire bloc
```

Cas particulier - à éviter !

- le bloc d'instructions d'une boucle tant que peut ne jamais être exécuté (condition non vérifiée la première fois)
  Exemple :

```
i = 0
tant que i > 0 faire bloc
```

- on peut ne jamais sortir d'une boucle while (condition toujours vérifiée)
  Exemple :

```
tant que TRUE faire bloc
```

Tant que en Python : while

```
tant que condition faire bloc
```

Python - Tant que

```
while booleen :
    # bloc d'instructions
    ...
```

Des exemples - Compteur

Algorithme : tantque

Données : compteur:entier

```
// initialisation des variables
compteur = 1
// séquence d'opérations
tant que compteur ≤ 10 faire
    afficher compteur
    compteur = compteur + 1
```

1. compteur = 1 : condition = **TRUE** –> afficher 1

Des exemples - Compteur

Algorithme : tantque

Données : compteur:entier

```
// initialisation des variables
compteur = 1
// séquence d'opérations
tant que compteur ≤ 10 faire
    afficher compteur
    compteur = compteur + 1
```

1. compteur = 1 : condition = **TRUE** –> afficher 1
2. compteur = 2 : condition = **TRUE** –> afficher 2

Des exemples - Compteur

Algorithme : tantque

Données : compteur:entier

```
// initialisation des variables
compteur = 1
// séquence d'opérations
tant que compteur ≤ 10 faire
    afficher compteur
    compteur = compteur + 1
```

1. compteur = 1 : condition = **TRUE** –> afficher 1
2. compteur = 2 : condition = **TRUE** –> afficher 2
3. compteur = 3 : condition = **TRUE** –> afficher 3

Des exemples - Compteur

Algorithme : tantque

Données : compteur:entier

```
// initialisation des variables
compteur = 1
// séquence d'opérations
tant que compteur ≤ 10 faire
    afficher compteur
    compteur = compteur + 1
```

1. compteur = 1 : condition = **TRUE** –> afficher 1
2. compteur = 2 : condition = **TRUE** –> afficher 2
3. compteur = 3 : condition = **TRUE** –> afficher 3
4. compteur = 4 : condition = **TRUE** –> afficher 4

Des exemples - Compteur

Algorithme : tantque

Données : compteur:entier

```
// initialisation des variables
compteur = 1
// séquence d'opérations
tant que compteur ≤ 10 faire
    afficher compteur
    compteur = compteur + 1
```

1. compteur = 1 : condition = **TRUE** –> afficher 1
2. compteur = 2 : condition = **TRUE** –> afficher 2
3. compteur = 3 : condition = **TRUE** –> afficher 3
4. compteur = 4 : condition = **TRUE** –> afficher 4
5. compteur = 5 : condition = **TRUE** –> afficher 5

Des exemples - Compteur

Algorithme : tantque

Données : compteur:entier

```
// initialisation des variables
compteur = 1
// séquence d'opérations
tant que compteur ≤ 10 faire
    afficher compteur
    compteur = compteur + 1
```

1. compteur = 1 : condition = **TRUE** –> afficher 1
2. compteur = 2 : condition = **TRUE** –> afficher 2
3. compteur = 3 : condition = **TRUE** –> afficher 3
4. compteur = 4 : condition = **TRUE** –> afficher 4
5. compteur = 5 : condition = **TRUE** –> afficher 5
6. compteur = 6 : condition = **TRUE** –> afficher 6

Des exemples - Compteur

Algorithme : tantque

Données : compteur:entier

```
// initialisation des variables
compteur = 1
// séquence d'opérations
tant que compteur ≤ 10 faire
    afficher compteur
    compteur = compteur + 1
```

1. compteur = 1 : condition = **TRUE** –> afficher 1
2. compteur = 2 : condition = **TRUE** –> afficher 2
3. compteur = 3 : condition = **TRUE** –> afficher 3
4. compteur = 4 : condition = **TRUE** –> afficher 4
5. compteur = 5 : condition = **TRUE** –> afficher 5
6. compteur = 6 : condition = **TRUE** –> afficher 6
7. compteur = 7 : condition = **TRUE** –> afficher 7

Des exemples - Compteur

Algorithme : tantque

Données : compteur:entier

```
// initialisation des variables
compteur = 1
// séquence d'opérations
tant que compteur ≤ 10 faire
    afficher compteur
    compteur = compteur + 1
```

1. compteur = 1 : condition = **TRUE** –> afficher 1
2. compteur = 2 : condition = **TRUE** –> afficher 2
3. compteur = 3 : condition = **TRUE** –> afficher 3
4. compteur = 4 : condition = **TRUE** –> afficher 4
5. compteur = 5 : condition = **TRUE** –> afficher 5
6. compteur = 6 : condition = **TRUE** –> afficher 6
7. compteur = 7 : condition = **TRUE** –> afficher 7
8. compteur = 8 : condition = **TRUE** –> afficher 8

Des exemples - Compteur

Algorithme : tantque

Données : compteur:entier

```
// initialisation des variables
compteur = 1
// séquence d'opérations
tant que compteur ≤ 10 faire
    afficher compteur
    compteur = compteur + 1
```

1. compteur = 1 : condition = **TRUE** –> afficher 1
2. compteur = 2 : condition = **TRUE** –> afficher 2
3. compteur = 3 : condition = **TRUE** –> afficher 3
4. compteur = 4 : condition = **TRUE** –> afficher 4
5. compteur = 5 : condition = **TRUE** –> afficher 5
6. compteur = 6 : condition = **TRUE** –> afficher 6
7. compteur = 7 : condition = **TRUE** –> afficher 7
8. compteur = 8 : condition = **TRUE** –> afficher 8
9. compteur = 9 : condition = **TRUE** –> afficher 9

Des exemples - Compteur

Algorithme : tantque

Données : compteur:entier

```
// initialisation des variables
compteur = 1
// séquence d'opérations
tant que compteur ≤ 10 faire
    afficher compteur
    compteur = compteur + 1
```

1. compteur = 1 : condition = **TRUE** –> afficher 1
2. compteur = 2 : condition = **TRUE** –> afficher 2
3. compteur = 3 : condition = **TRUE** –> afficher 3
4. compteur = 4 : condition = **TRUE** –> afficher 4
5. compteur = 5 : condition = **TRUE** –> afficher 5
6. compteur = 6 : condition = **TRUE** –> afficher 6
7. compteur = 7 : condition = **TRUE** –> afficher 7
8. compteur = 8 : condition = **TRUE** –> afficher 8
9. compteur = 9 : condition = **TRUE** –> afficher 9
10. compteur = 10 : condition = **TRUE** –> afficher 10

Des exemples - Compteur

Algorithme : tantque

Données : compteur:entier

```
// initialisation des variables
compteur = 1
// séquence d'opérations
tant que compteur ≤ 10 faire
    afficher compteur
    compteur = compteur + 1
```

1. compteur = 1 : condition = **TRUE** –> afficher 1
2. compteur = 2 : condition = **TRUE** –> afficher 2
3. compteur = 3 : condition = **TRUE** –> afficher 3
4. compteur = 4 : condition = **TRUE** –> afficher 4
5. compteur = 5 : condition = **TRUE** –> afficher 5
6. compteur = 6 : condition = **TRUE** –> afficher 6
7. compteur = 7 : condition = **TRUE** –> afficher 7
8. compteur = 8 : condition = **TRUE** –> afficher 8
9. compteur = 9 : condition = **TRUE** –> afficher 9
10. compteur = 10 : condition = **TRUE** –> afficher 10
11. compteur = 11 : condition = **FALSE**

Des exemples - Compteur

Algorithme : tantque

Données : compteur:entier

```
// initialisation des variables
compteur = 1
// séquence d'opérations
tant que compteur ≤ 10 faire
    afficher compteur
    compteur = compteur + 1
```

Python

```python
# sequence d'operations
compteur = 1
while compteur <= 10:
    print(compteur)
    compteur = compteur + 1
```

1. compteur = 1 : condition = **TRUE** –> afficher 1
2. compteur = 2 : condition = **TRUE** –> afficher 2
3. compteur = 3 : condition = **TRUE** –> afficher 3
4. compteur = 4 : condition = **TRUE** –> afficher 4
5. compteur = 5 : condition = **TRUE** –> afficher 5
6. compteur = 6 : condition = **TRUE** –> afficher 6
7. compteur = 7 : condition = **TRUE** –> afficher 7
8. compteur = 8 : condition = **TRUE** –> afficher 8
9. compteur = 9 : condition = **TRUE** –> afficher 9
10. compteur = 10 : condition = **TRUE** –> afficher 10
11. compteur = 11 : condition = **FALSE**

Des exemples - Une table de multiplication

Algorithme : table_multi

Données : table, compteur, resultat:entier

```
// séquence d'opérations
1 afficher ``Quelle table de multiplication ?''
2 saisir table
3 compteur = 1
4 tant que compteur ≤ 10 faire
5     resultat = compteur * table
6     afficher
        table,``x'',compteur,``='',resultat
7     compteur = compteur + 1
```

Des exemples - Une table de multiplication

Algorithme : table_multi

Données : table, compteur, resultat:entier

```
// séquence d'opérations
1 afficher ``Quelle table de multiplication ?''
2 saisir table
3 compteur = 1
4 tant que compteur ≤ 10 faire
5     resultat = compteur * table
6     afficher
        table,``x'',compteur,``='',resultat
7     compteur = compteur + 1
```

Saisie

```
table = 10
```

- compteur = 1 : condition = **TRUE**
  `afficher` $10 \times 1 = 10$

Des exemples - Une table de multiplication

Algorithme : table_multi

Données : table, compteur, resultat:entier

```
// séquence d'opérations
1 afficher ``Quelle table de multiplication ?''
2 saisir table
3 compteur = 1
4 tant que compteur ≤ 10 faire
5     resultat = compteur * table
6     afficher
        table,``x'',compteur,``='',resultat
7     compteur = compteur + 1
```

Saisie

```
table = 10
```

- compteur = 1 : condition = **TRUE**
  `afficher` $10 \times 1 = 10$
- compteur = 2 : condition = **TRUE**
  `afficher` $10 \times 2 = 20$

Des exemples - Une table de multiplication

Algorithme : table_multi

Données : table, compteur, resultat:entier

```
// séquence d'opérations
1 afficher ``Quelle table de multiplication ?''
2 saisir table
3 compteur = 1
4 tant que compteur ≤ 10 faire
5     resultat = compteur * table
6     afficher
        table,``x'',compteur,``='',resultat
7     compteur = compteur + 1
```

Saisie

```
table = 10
```

- compteur = 1 : condition = **TRUE**
  `afficher` $10 \times 1 = 10$
- compteur = 2 : condition = **TRUE**
  `afficher` $10 \times 2 = 20$
- compteur = 3 : condition = **TRUE**
  `afficher` $10 \times 3 = 30$

Des exemples - Une table de multiplication

Algorithme : table_multi

Données : table, compteur, resultat:entier

```
// séquence d'opérations
1 afficher ``Quelle table de multiplication ?''
2 saisir table
3 compteur = 1
4 tant que compteur ≤ 10 faire
5     resultat = compteur * table
6     afficher
        table,``x'',compteur,``='',resultat
7     compteur = compteur + 1
```

Saisie

```
table = 10
```

- compteur = 1 : condition = **TRUE**
  `afficher` $10 \times 1 = 10$
- compteur = 2 : condition = **TRUE**
  `afficher` $10 \times 2 = 20$
- compteur = 3 : condition = **TRUE**
  `afficher` $10 \times 3 = 30$
- ...

Des exemples - Une table de multiplication

Algorithme : table_multi

Données : table, compteur, resultat:entier

```
// séquence d'opérations
1 afficher ``Quelle table de multiplication ?''
2 saisir table
3 compteur = 1
4 tant que compteur ≤ 10 faire
5     resultat = compteur * table
6     afficher
        table,``x'',compteur,``='',resultat
7     compteur = compteur + 1
```

Saisie

```
table = 10
```

- compteur = 1 : condition = **TRUE**
  `afficher` $10 \times 1 = 10$
- compteur = 2 : condition = **TRUE**
  `afficher` $10 \times 2 = 20$
- compteur = 3 : condition = **TRUE**
  `afficher` $10 \times 3 = 30$
- ...
- compteur = 9 : condition = **TRUE**
  `afficher` $10 \times 9 = 90$

Des exemples - Une table de multiplication

Algorithme : table_multi

Données : table, compteur, resultat:entier

```
// séquence d'opérations
1 afficher ``Quelle table de multiplication ?''
2 saisir table
3 compteur = 1
4 tant que compteur ≤ 10 faire
5     resultat = compteur * table
6     afficher
        table,``x'',compteur,``='',resultat
7     compteur = compteur + 1
```

Saisie

```
table = 10
```

- compteur = 1 : condition = **TRUE**
  `afficher` $10 \times 1 = 10$
- compteur = 2 : condition = **TRUE**
  `afficher` $10 \times 2 = 20$
- compteur = 3 : condition = **TRUE**
  `afficher` $10 \times 3 = 30$
- ...
- compteur = 9 : condition = **TRUE**
  `afficher` $10 \times 9 = 90$
- compteur = 10 : condition = **TRUE**
  `afficher` $10 \times 10 = 100$

Des exemples - Une table de multiplication

Algorithme : table_multi

Données : table, compteur, resultat:entier

```
// séquence d'opérations
1 afficher ``Quelle table de multiplication ?''
2 saisir table
3 compteur = 1
4 tant que compteur ≤ 10 faire
5     resultat = compteur * table
6     afficher
        table,``x'',compteur,``='',resultat
7     compteur = compteur + 1
```

Saisie

```
table = 10
```

- compteur = 1 : condition = **TRUE**
  `afficher` $10 \times 1 = 10$
- compteur = 2 : condition = **TRUE**
  `afficher` $10 \times 2 = 20$
- compteur = 3 : condition = **TRUE**
  `afficher` $10 \times 3 = 30$
- ...
- compteur = 9 : condition = **TRUE**
  `afficher` $10 \times 9 = 90$
- compteur = 10 : condition = **TRUE**
  `afficher` $10 \times 10 = 100$
- compteur = 11 : condition = **FALSE**

Une table de multiplication en Python

Table de multiplication

```
## table multiplication ##

# sequence d'operations
table = int(input("Quelle table de multiplication ?\n"))

compteur = 1
while compteur <= 10:
    resultat = table * compteur
    print(table, " x ", compteur, " = ", resultat)
    compteur = compteur + 1
```

Des examples - Une table de multiplication reverse

Algorithme : table_multi_rev

Données : table, compteur, resultat:entier

```
// séquence d'opérations
1 afficher ``Quelle table de multiplication ?''
2 saisir table
3 compteur = 10
4 tant que compteur > 0 faire
5     resultat = table * compteur
6     afficher
        table,``x'',compteur,``='',resultat
7     compteur = compteur - 1
```

Des examples - Une table de multiplication reverse

Algorithme : table_multi_rev

Données : table, compteur, resultat:entier

```
// séquence d'opérations
1 afficher ``Quelle table de multiplication ?''
2 saisir table
3 compteur = 10
4 tant que compteur > 0 faire
5     resultat = table * compteur
6     afficher
        table,``x'',compteur,``='',resultat
7     compteur = compteur - 1
```

Saisie

```
table = 10
```

- compteur = 10 : condition = **TRUE**
  `afficher` $10 \times 10 = 100$
- compteur = 9 : condition = **TRUE**
  `afficher` $10 \times 9 = 90$
- compteur = 8 : condition = **TRUE**
  `afficher` $10 \times 8 = 80$
- ...
- compteur = 2 : condition = **TRUE**
  `afficher` $10 \times 2 = 20$
- compteur = 1 : condition = **TRUE**
  `afficher` $10 \times 1 = 10$
- compteur = 0 : condition = **FALSE**

Exercice - Tant que I

Question : Quel sera la valeur (`resultat`) afficher par l'algorithme suivant ?

A) 1

B) 5

C) 4

D) 120

E) 0

Algorithme : boucle_x

Données : compteur, resultat:entier

```
// initialisation des variables
resultat = 1
// séquence d'opérations
x = 5
tant que x ≥ 1 faire
    resultat = x * resultat
    n = n - 1
afficher ``resultat = '',resultat
```

Exercice - Tant que I

Question : Quel sera la valeur (`resultat`) afficher par l'algorithme suivant ?

A) 1

B) 5

C) 4

D) 120

E) 0

Algorithme : boucle_x

Données : compteur, resultat:entier

```
// initialisation des variables
resultat = 1
// séquence d'opérations
x = 5
tant que x ≥ 1 faire
    resultat = x * resultat
    n = n - 1
afficher ``resultat = '',resultat
```

Exercice - Tant que I

- x = 5 : condition = **TRUE**
  resultat $= 5 \times 1$
- x = 4 : condition = **TRUE**
  resultat $= 4 \times 5$
- x = 3 : condition = **TRUE**
  resultat $= 3 \times 20$
- x = 2 : condition = **TRUE**
  resultat $= 2 \times 60$
- x = 1 : condition = **TRUE**
  resultat $= 1 \times 120$
- x = 0 : condition = **FALSE**

Algorithme : boucle_x

Données : compteur, resultat:entier

```
// initialisation des variables
resultat = 1
// séquence d'opérations
x = 5
tant que x ≥ 1 faire
    resultat = x * resultat
    x = x - 1
afficher ``resultat = '',x
```

Factorielle

$$n! = n \times (n-1) \times \ldots \times 2 \times 1$$
$$5! = 5 \times 4 \times 3 \times 2 \times 1$$
$$5! = 120$$

Des exemples - x à la puissance y

Algorithme : puissance

Données : x, n, compteur, resultat:entier

```
// initialisation des variables
resultat = 1
// séquence d'opérations
afficher ``x,n?''
saisir x,n
compteur = 1
tant que compteur ≤ n faire
    resultat = resultat * x
    compteur = compteur + 1
afficher resultat
```

Rappel

$$x^n = \overbrace{x * x * \ldots * x * x}^{=n}$$

$$2^4 = 2 * 2 * 2 * 2$$

Des exemples - x à la puissance y

Algorithme : puissance

Données : x, n, compteur, resultat:entier

```
   // initialisation des variables
1  resultat = 1
   // séquence d'opérations
2  afficher ``x,n?''
3  saisir x,n
4  compteur = 1
5  tant que compteur ≤ n faire
6  |  resultat = resultat * x
7  |  compteur = compteur + 1
8  afficher resultat
```

Saisie

```
x = 2
n = 4
```

Rappel

$$x^n = \overbrace{x * x * \ldots * x * x}^{=n}$$

$$2^4 = 2 * 2 * 2 * 2$$

Des exemples - x à la puissance y

Algorithme : puissance

Données : x, n, compteur, resultat:entier

```
// initialisation des variables
resultat = 1
// séquence d'opérations
afficher ``x,n?''
saisir x,n
compteur = 1
tant que compteur ≤ n faire
    resultat = resultat * x
    compteur = compteur + 1
afficher resultat
```

Rappel

$$x^n = \overbrace{x * x * \ldots * x * x}^{=n}$$

$$2^4 = 2 * 2 * 2 * 2$$

Saisie

```
x = 2
n = 4
```

- compteur = 1 : condition = **TRUE**

  resultat $= 1 \times 2$

Des exemples - x à la puissance y

Algorithme : puissance

Données : x, n, compteur, resultat:entier

```
// initialisation des variables
1 resultat = 1
// séquence d'opérations
2 afficher ``x,n?''
3 saisir x,n
4 compteur = 1
5 tant que compteur ≤ n faire
6 |   resultat = resultat * x
7 |   compteur = compteur + 1
8 afficher resultat
```

Rappel

$$x^n = \overbrace{x * x * \ldots * x * x}^{=n}$$

$$2^4 = 2 * 2 * 2 * 2$$

Saisie

```
x = 2
n = 4
```

- compteur = 1 : condition = **TRUE**
  resultat $= 1 \times 2$
- compteur = 2 : condition = **TRUE**
  afficher $= 2 \times 2$

Des exemples - x à la puissance y

Algorithme : puissance

Données : x, n, compteur, resultat:entier

```
// initialisation des variables
resultat = 1
// séquence d'opérations
afficher ``x,n?''
saisir x,n
compteur = 1
tant que compteur ≤ n faire
    resultat = resultat * x
    compteur = compteur + 1
afficher resultat
```

Rappel

$$x^n = \overbrace{x * x * \ldots * x * x}^{=n}$$

$$2^4 = 2 * 2 * 2 * 2$$

Saisie

```
x = 2
n = 4
```

- compteur = 1 : condition = **TRUE**
  `resultat` $= 1 \times 2$
- compteur = 2 : condition = **TRUE**
  `afficher` $= 2 \times 2$
- compteur = 3 : condition = **TRUE**
  `afficher` $= 4 \times 2$

Des exemples - x à la puissance y

Algorithme : puissance

Données : x, n, compteur, resultat:entier

```
// initialisation des variables
resultat = 1
// séquence d'opérations
afficher ``x,n?''
saisir x,n
compteur = 1
tant que compteur ≤ n faire
    resultat = resultat * x
    compteur = compteur + 1
afficher resultat
```

Rappel

$$x^n = \overbrace{x * x * \ldots * x * x}^{=n}$$

$$2^4 = 2 * 2 * 2 * 2$$

Saisie

```
x = 2
n = 4
```

- compteur = 1 : condition = **TRUE**
  `resultat` $= 1 \times 2$
- compteur = 2 : condition = **TRUE**
  `afficher` $= 2 \times 2$
- compteur = 3 : condition = **TRUE**
  `afficher` $= 4 \times 2$
- compteur = 4 : condition = **TRUE**
  `afficher` $= 8 \times 2$

Des exemples - x à la puissance y

Algorithme : puissance

Données : x, n, compteur, resultat:entier

```
// initialisation des variables
1 resultat = 1
// séquence d'opérations
2 afficher ``x,n?''
3 saisir x,n
4 compteur = 1
5 tant que compteur ≤ n faire
6 |   resultat = resultat * x
7 |   compteur = compteur + 1
8 afficher resultat
```

Rappel

$$x^n = \overbrace{x * x * \ldots * x * x}^{=n}$$

$$2^4 = 2 * 2 * 2 * 2$$

Saisie

```
x = 2
n = 4
```

- compteur = 1 : condition = **TRUE**
  `resultat` $= 1 \times 2$
- compteur = 2 : condition = **TRUE**
  `afficher` $= 2 \times 2$
- compteur = 3 : condition = **TRUE**
  `afficher` $= 4 \times 2$
- compteur = 4 : condition = **TRUE**
  `afficher` $= 8 \times 2$
- compteur = 5 : condition = **FALSE**

Des exemples - x à la puissance n

Algorithme : puissance

Données : n:entier, x:numerique

```
1  resultat = 1:numerique                    // initialisation des variables
2  signe = 1:entier
3  afficher ``x,n?''                         // séquence d'opérations
4  saisir x,n
5  si n ≠ 0 alors                                                  // x^0 = 1
6      si n ≤ 0 alors                                  // test le signe de n
7          n = -n                                                // valeur absolue
8          signe = -1                                 // puissance negative
9      compteur = 1:entier
10     tant que compteur ≤ n faire
11         resultat = resultat * x
12         compteur = compteur + 1
13     si signe < 0 alors
14         resultat = 1/resultat
15 afficher resultat
```

Des exemples - x à la puissance n en Python

```
## puissance ##
resultat = 1            # initialisation des variables
signe = 1
print("x, n ?")         # sequence d'operations
x = float(input())
n = int(input())
if n != 0:
    if n < 0:
        n = -n
        signe = -1
    compteur = 1
    while compteur <= n:
        resultat = resultat * x
        compteur = compteur + 1
    if signe < 0:
        resultat = 1 / resultat
print(resultat)
```

Des boucles imbriquées

Boucles imbriquées - une boucle dans une autre boucle

De même qu'une structure si ... alors peut contenir d'autres structures si ... alors, une boucle peut tout à fait contenir d'autres boucles

Calcule de la moyenne par étudiant-e

« prenons tou-te-s les étudiant-e-s de la filière IG un par un »
    « pour chaque étudiant-e, prenons toutes les notes et calculons la moyenne »

Des boucles imbriquées - Exemple

Table de multiplication 10 x 10

Calculer et d'afficher toutes les tables de multiplication de 1 à 10 :

- cela deux boucles doivent être utilisées
- la première va représenter la table à calculer, de 1 à 10
- la seconde à l'intérieur de la première va multiplier la table donnée de 1 à 10

- première boucle : table des **1**
  seconde boucle : exécution de `1*1`, `1*2`, `1*3`, `...`, `1*9`, `1*10`

Des boucles imbriquées - Exemple

Table de multiplication 10 x 10

Calculer et d'afficher toutes les tables de multiplication de 1 à 10 :

- cela deux boucles doivent être utilisées
- la première va représenter la table à calculer, de 1 à 10
- la seconde à l'intérieur de la première va multiplier la table donnée de 1 à 10

- première boucle : table des **1**
  - seconde boucle : exécution de `1*1`, `1*2`, `1*3`, `...`, `1*9`, `1*10`
- première boucle : table des **2**
  - seconde boucle : exécution de `2*1`, `2*2`, `2*3`, `...2*9`, `2*10`

Des boucles imbriquées - Exemple

Table de multiplication 10 x 10

Calculer et d'afficher toutes les tables de multiplication de 1 à 10 :

- cela deux boucles doivent être utilisées
- la première va représenter la table à calculer, de 1 à 10
- la seconde à l'intérieur de la première va multiplier la table donnée de 1 à 10

- première boucle : table des **1**
  - seconde boucle : exécution de `1*1, 1*2, 1*3, ..., 1*9, 1*10`
- première boucle : table des **2**
  - seconde boucle : exécution de `2*1, 2*2, 2*3, ...2*9, 2*10`
- première boucle : table des **3**
  - seconde boucle : exécution de `3*1, 3*2, 3*3, ... ,3*9,3*10`

Des boucles imbriquées - Exemple

Table de multiplication 10 x 10

Calculer et d'afficher toutes les tables de multiplication de 1 à 10 :

- cela deux boucles doivent être utilisées
- la première va représenter la table à calculer, de 1 à 10
- la seconde à l'intérieur de la première va multiplier la table donnée de 1 à 10

- première boucle : table des **1**
  - seconde boucle : exécution de `1*1, 1*2, 1*3, ..., 1*9, 1*10`
- première boucle : table des **2**
  - seconde boucle : exécution de `2*1, 2*2, 2*3, ...2*9, 2*10`
- première boucle : table des **3**
  - seconde boucle : exécution de `3*1, 3*2, 3*3, ... ,3*9,3*10`
- ...

Des boucles imbriquées - Exemple

Table de multiplication 10 x 10

Calculer et d'afficher toutes les tables de multiplication de 1 à 10 :

- cela deux boucles doivent être utilisées
- la première va représenter la table à calculer, de 1 à 10
- la seconde à l'intérieur de la première va multiplier la table donnée de 1 à 10

- première boucle : table des **1**
  seconde boucle : exécution de `1*1`, `1*2`, `1*3`, ..., `1*9`, `1*10`
- première boucle : table des **2**
  seconde boucle : exécution de `2*1`, `2*2`, `2*3`, ...`2*9`, `2*10`
- première boucle : table des **3**
  seconde boucle : exécution de `3*1`, `3*2`, `3*3`, ... ,`3*9`,`3*10`
- ...
- première boucle : table des **10**
  seconde boucle : exécution de `10*1`, `10*2`, `10*3`, ..., `10*9`, `10*10`

Structures des boucles imbriquées

```
tant que condition_1 faire
  tant que condition_2 faire
            ...
      tant que condition_n faire
                bloc_n
```

Python - Tant que imbriqué

```
while booleen_1 :
    while booleen_2 :
        ...
            while booleen_n :
                # bloc d'instructions n
                ...
```

Toutes les tables de multiplication

Algorithme : boucle_imbriquee

Données : table, compteur, resultat :entier

```
1  table = 1                                  // initialisation des variables
   // séquence d'opérations
2  tant que table ≤ 10 faire                  // traiter la boucle extérieur
3      afficher "table des", table
4      compteur = 1
5      tant que compteur ≤ 10 faire           // traiter la boucle intérieur
6          resultat = table * compteur
7          afficher table," x ",compteur," = ",resultat
8          compteur = compteur + 1
9      table = table + 1
10 afficher resultat
```

Exercice - Tant que II

Question : Quel est le résultat du algorithme suivant :

A) 1234567

B) 123456

C) 23456

D) 234567

Algorithme : boucle_x

Données : jour:entier

```
// initialisation des variables
jour = 1
// séquence d'opérations
tant que jour < 7 faire
    jour = jour + 1
    afficher jour
```

Exercice - Tant que II

Question : Quel est le résultat du algorithme suivant :

A) 1234567

B) 123456

C) 23456

D) 234567

Algorithme : boucle_x

Données : jour:entier

```
// initialisation des variables
jour = 1
// séquence d'opérations
tant que jour < 7 faire
    jour = jour + 1
    afficher jour
```

Exercice - Tant que III

Question : Quel est le résultat du algorithme suivant :

A) 1

B) 9

C) 11

D) 13

Algorithme : boucle_x

Données : plus:entier

```
// initialisation des variables
plus = 1
// séquence d'opérations
tant que plus ≤ 10 faire
    plus = plus + 2
afficher jour
```

Exercice - Tant que III

Question : Quel est le résultat du algorithme suivant :

A) 1

B) 9

C) 11

D) 13

Algorithme : boucle_x

Données : plus:entier

```
// initialisation des variables
plus = 1
// séquence d'opérations
tant que plus ≤ 10 faire
    plus = plus + 2
afficher jour
```

Exercice - Tant que IV

Question : Quel est le résultat du algorithme suivant :

A) 15

B) 1

C) 0

D) -1

Algorithme : boucle_x

Données : x:entier

```
// initialisation des variables
x = 15
// séquence d'opérations
tant que x > 0 faire
    x = x - 2
afficher x
```

Exercice - Tant que IV

Question : Quel est le résultat du algorithme suivant :

A) 15

B) 1

C) 0

D) -1

Algorithme : boucle_x

Données : x:entier

```
// initialisation des variables
x = 15
// séquence d'opérations
tant que x > 0 faire
    x = x - 2
afficher x
```

Exercice - Tant que V

Question : Quel est le résultat du algorithme suivant :

A) rien

B) 110

C) 10

D) 100

Algorithme : boucle_x

Données : x:entier

```
// initialisation des variables
x = 10
// séquence d'opérations
tant que x > 0 faire
    x = x + 10
afficher x
```

Exercice - Tant que V

Question : Quel est le résultat du algorithme suivant :

A) rien

B) 110

C) 10

D) 100

Algorithme : boucle_x

Données : x:entier

```
// initialisation des variables
x = 10
// séquence d'opérations
tant que x > 0 faire
    x = x + 10
afficher x
```

Exercice - Tant que VI

Question : Quel est le résultat du algorithme suivant :

A) rien

B) 2345678910...

C) 12345678910...

D) TRUE

Algorithme : boucle_x

Données : x:entier

```
// initialisation des variables
x = 1
// séquence d'opérations
tant que TRUE faire
    x = x + 1
    afficher x
```

Exercice - Tant que VI

Question : Quel est le résultat du algorithme suivant :

A) rien

B) 2345678910...

C) 12345678910...

D) TRUE

Algorithme : boucle_x

Données : x:entier

```
// initialisation des variables
x = 1
// séquence d'opérations
tant que TRUE faire
    x = x + 1
    afficher x
```

Exercice - Tant que VII

Question : Quel est le résultat du algorithme suivant :

A) 0-0-0-

B) 2-3-6-

C) 2-5-11-

D) 3-3-3-

Algorithme : boucle_x

Données : x,y,res:entier

```
// séquence d'opérations
x = 3
tant que x > 0 faire
    res = 0
    y = 3
    tant que y > 0 faire
        res = res + y
        y = y - 1
    res = res / x
    afficher res,"-"
    x = x - 1
```

Exercice - Tant que VII

Question : Quel est le résultat du algorithme suivant :

A) 0-0-0-

B) 2-3-6-

C) 2-5-11-

D) 3-3-3-

Algorithme : boucle_x

Données : x,y,res:entier

```
// séquence d'opérations
x = 3
tant que x > 0 faire
    res = 0
    y = 3
    tant que y > 0 faire
        res = res + y
        y = y - 1
    res = res / x
    afficher res,"-"
    x = x - 1
```

Structure algorithmique - Pour (For)

Deuxième structure itérative de l'algorithmique : le `pour ... faire` est une boucle à l'usage des compteurs et séquences

Principe

À chaque passage dans la boucle, un compteur est incrémenté ou décrémenté, selon le cas On dit alors qu'il s'agit d'une structure incrémentale

```
pour element ⟵ debut à fin [pas] faire bloc
```

Structure algorithmique - Pour (For)

Deuxième structure itérative de l'algorithmique : le `pour ... faire` est une boucle à l'usage des compteurs et séquences

Principe

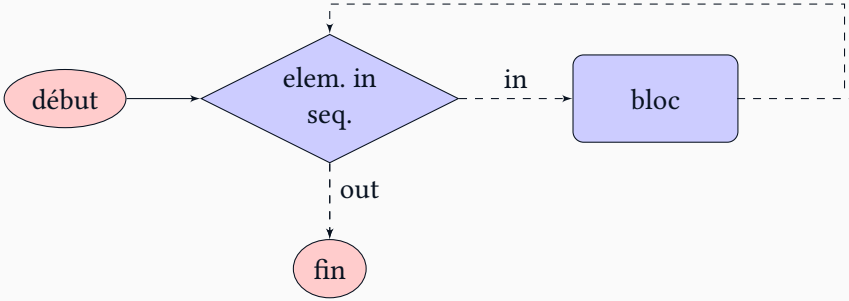
À chaque passage dans la boucle, un compteur est incrémenté ou décrémenté, selon le cas On dit alors qu'il s'agit d'une structure incrémentale

```
pour element ⟵ debut à fin [pas] faire bloc
```

Python - Pour

```
for element in sequence :
    # bloc d'instructions
    ...
```

Structure algorithmique - Pour (For)

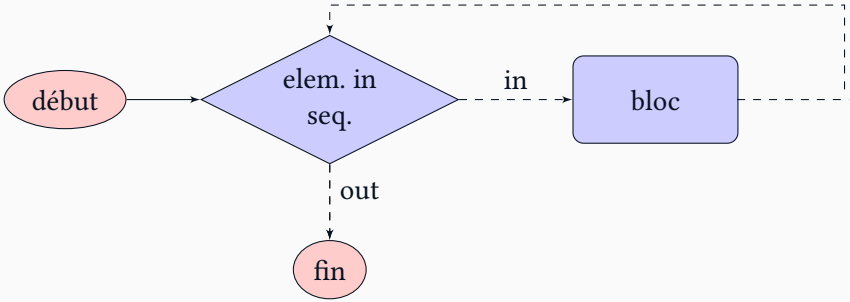


Sequence

- 1, 2, 3, ..., 9, 10

```
pour compteur ← 1 à 10 faire ...
```

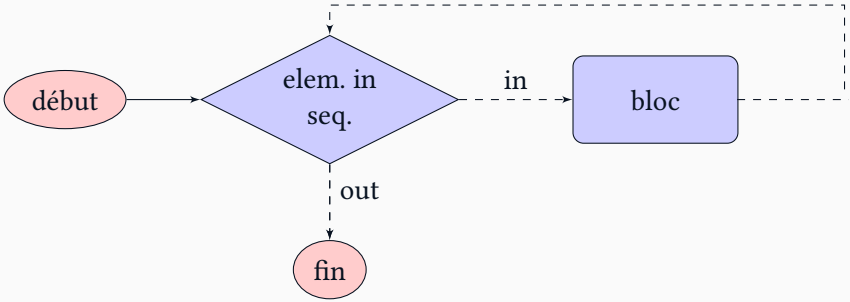
Structure algorithmique - Pour (For)



Sequence

- 1, 2, 3, ..., 9, 10 — `pour compteur ← 1 à 10 faire ...`
- 0, 2, 4, ..., 8, 10 — `pour cpt_pair ← 0 à 10 [pas 2] faire ...`

Structure algorithmique - Pour (For)



Sequence

- 1, 2, 3, ..., 9, 10 — `pour compteur ← 1 à 10 faire ...`
- 0, 2, 4, ..., 8, 10 — `pour cpt_pair ← 0 à 10 [pas 2] faire ...`
- 10, 9, 8, ..., 2, 1 — `pour cpt_rev ← 10 à 1 [pas -1] faire ...`

Un example simple

Pour créer une table de multiplication, de 3 par exemple, on peut procéder comme suivant :

Créer une table de multiplication de 3

séquénce =
compteur ≥ 1 et compteur ≤ 10

- $3 \times 1 = 3$
- $3 \times 2 = 6$
- ...
- $3 \times 9 = 27$
- $3 \times 10 = 30$

pour compteur $\leftarrow 1$ à 10 faire

- pas 1 : afficher 3×1 (compteur = 1)

Un example simple

Pour créer une table de multiplication, de 3 par exemple, on peut procéder comme suivant :

Créer une table de multiplication de 3

séquénce = compteur ≥ 1 et compteur ≤ 10

- $3 \times 1 = 3$
- $3 \times 2 = 6$
- ...
- $3 \times 9 = 27$
- $3 \times 10 = 30$

pour compteur $\leftarrow 1$ à 10 faire

- pas 1 : afficher 3×1 (compteur = 1)
- pas 2 : afficher 3×2 (compteur = 2)

Un example simple

Pour créer une table de multiplication, de 3 par exemple, on peut procéder comme suivant :

Créer une table de multiplication de 3

séquénce =
compteur ≥ 1 et compteur ≤ 10

- $3 \times 1 = 3$
- $3 \times 2 = 6$
- ...
- $3 \times 9 = 27$
- $3 \times 10 = 30$

pour compteur $\leftarrow 1$ à 10 faire

- pas 1 : afficher 3×1 (compteur = 1)
- pas 2 : afficher 3×2 (compteur = 2)
- pas 3 : afficher 3×3 (compteur = 3)

Un example simple

Pour créer une table de multiplication, de 3 par exemple, on peut procéder comme suivant :

Créer une table de multiplication de 3

séquénce = compteur ≥ 1 et compteur ≤ 10

- $3 \times 1 = 3$
- $3 \times 2 = 6$
- ...
- $3 \times 9 = 27$
- $3 \times 10 = 30$

pour compteur $\leftarrow 1$ à 10 faire

- pas 1 : afficher 3×1 (compteur = 1)
- pas 2 : afficher 3×2 (compteur = 2)
- pas 3 : afficher 3×3 (compteur = 3)
- ...

Un example simple

Pour créer une table de multiplication, de 3 par exemple, on peut procéder comme suivant :

Créer une table de multiplication de 3

séquénce = compteur ≥ 1 et compteur ≤ 10

- $3 \times 1 = 3$
- $3 \times 2 = 6$
- ...
- $3 \times 9 = 27$
- $3 \times 10 = 30$

pour compteur $\leftarrow 1$ à 10 faire

- pas 1 : afficher 3×1 (compteur = 1)
- pas 2 : afficher 3×2 (compteur = 2)
- pas 3 : afficher 3×3 (compteur = 3)
- ...
- pas 8 : afficher 3×8 (compteur = 8)

Un example simple

Pour créer une table de multiplication, de 3 par exemple, on peut procéder comme suivant :

Créer une table de multiplication de 3

séquénce = compteur ≥ 1 et compteur ≤ 10

- $3 \times 1 = 3$
- $3 \times 2 = 6$
- ...
- $3 \times 9 = 27$
- $3 \times 10 = 30$

pour compteur $\leftarrow 1$ à 10 faire

- pas 1 : afficher 3×1 (compteur = 1)
- pas 2 : afficher 3×2 (compteur = 2)
- pas 3 : afficher 3×3 (compteur = 3)
- ...
- pas 8 : afficher 3×8 (compteur = 8)
- pas 9 : afficher 3×9 (compteur = 9)

Un example simple

Pour créer une table de multiplication, de 3 par exemple, on peut procéder comme suivant :

Créer une table de multiplication de 3

séquénce = compteur ≥ 1 et compteur ≤ 10

- $3 \times 1 = 3$
- $3 \times 2 = 6$
- ...
- $3 \times 9 = 27$
- $3 \times 10 = 30$

pour compteur $\leftarrow 1$ à 10 faire

- pas 1 : afficher 3×1 (compteur = 1)
- pas 2 : afficher 3×2 (compteur = 2)
- pas 3 : afficher 3×3 (compteur = 3)
- ...
- pas 8 : afficher 3×8 (compteur = 8)
- pas 9 : afficher 3×9 (compteur = 9)
- pas 10 : afficher 3×10 (compteur = 10)

Pour en Python : for

```
pour element ←debut à fin [pas] faire bloc
```

Python - Pour

```
for element in sequence :
    # bloc d'instructions
    ...
```

Factorielle avec pour

Un classique : avec une factorielle de n, on sait à l'avance le nombre d'itérations nécessaire : n.

C'est donc une application de choix pour la structure `pour ... fair`

Algorithme : factorielle_pour

Données : compteur,i,resultat:numerique

```
// initialisation des variables
resultat = 1
afficher ``Quelle factorielle ?''
saisir compteur
pour i ⟵2 à compteur faire
    x = resultat*i
afficher resultat
```

Factorielle avec pour

Un classique : avec une factorielle de n, on sait à l'avance le nombre d'itérations nécessaire : n.

C'est donc une application de choix pour la structure `pour ... fair`

Algorithme : factorielle_pour

Données : compteur,i,resultat:numerique

```
// initialisation des variables
resultat = 1
afficher ``Quelle factorielle ?''
saisir compteur
pour i ⟵2 à compteur faire
    x = resultat*i
afficher resultat
```

factorielle - Python

```python
## factorielle ##
# initialisation des variables
resultat = 1
# sequence d'operations
print("Quelle factorielle ?")
compteur = int(input())
for i in range(2, compteur+1):
    resultat = resultat*i
print(resultat)
```

Quelle structure choisir

Pour

- on emploie une structure `pour` lorsqu'on connaît à l'avance le nombre d'itérations nécessaires au traitement
- la boucle `pour` est déterministe : son nombre d'itérations est fixé une fois pour toute et est en principe invariable

Tant que

- on emploie les structures `tant que` lorsqu'on ne connaît pas forcément à l'avance le nombre d'itérations qui seront nécessaires à l'obtention du résultat souhaité

Un piège à éviter

Tout comme il faut éviter les boucles infinies, il faut aussi éviter quelques erreurs qui peuvent se révéler très surprenantes selon le cas

Voici un exemple de ce qu'il ne faut pas faire :

Algorithme : bad_pour

Données : x:numerique

```
1 pour x ⟵ 1 à 31 faire
2     x = x*2
```

Exercice - Pour I

Question : Quel est le résultat du algorithme suivant :

A) 3-6-9-

B) 1-3-6-9

C) 11

D) 0

Algorithme : pour_x

Données : resultat:entier

```
// initialisation des variables
resultat = 0
// séquence d'opérations
pour i ⟵ 1 à 10 faire
    pour j ⟵ 1 à 10 faire
        si i == 3 * j alors
            afficher i,"-"
```

Exercice - Pour I

Question : Quel est le résultat du algorithme suivant :

A) 3-6-9-

B) 1-3-6-9

C) 11

D) 0

Algorithme : pour_x

Données : resultat:entier

```
// initialisation des variables
resultat = 0
// séquence d'opérations
pour i ⟵ 1 à 10 faire
    pour j ⟵ 1 à 10 faire
        si i == 3 * j alors
            afficher i,"-"
```

Conclusion

Contenu vu :

- précédence des opérateurs
- les boucles simple et imbriqué
  - tant que
  - pour

Référence

Algorithmique - Techniques fondamentales de programmation
Chapitre : Les boucles
Ebel et Rohaut, https://aai-logon.hes-so.ch/eni

Cyberlearn : HES-SO-GE_631-1 Fondements de la programmation
(*welcome*)
http://cyberlearn.hes-so.ch