

Algorithmique

— instructions de base —

Jacques TISSEAU

LISYC EA 3883 UBO-ENIB-ENSIETA
Centre Européen de Réalité Virtuelle
Ecole Nationale d'Ingénieurs de Brest

enib ©2007



Instructions

Instruction

Commande élémentaire interprétée et exécutée par le processeur.

Instructions

Instruction

Commande élémentaire interprétée et exécutée par le processeur.

Jeu d'instructions

Dans un processeur, ensemble des instructions que cette puce peut exécuter.

Instructions

Instruction

Commande élémentaire interprétée et exécutée par le processeur.

Jeu d'instructions

Dans un processeur, ensemble des instructions que cette puce peut exécuter.

Bloc d'instructions

Dans un algorithme, séquence d'instructions pouvant être vue comme une seule instruction.

Jeux d'instructions des microprocesseurs

Classes d'instructions μP

arithmétique : +, -, *, /

logique : not, and, or

transferts de données : load,
store, move

contrôle du flux d'instructions :
branchements,
boucles, appels de
procédure

entrée-sortie : read, write

Jeux d'instructions des microprocesseurs

Classes d'instructions μP

arithmétique : +, -, *, /

logique : not, and, or

transferts de données : load,
store, move

contrôle du flux d'instructions :
branchements,
boucles, appels de
procédure

entrée-sortie : read, write

Traitement des instructions

- ① fetch : chargement de l'instruction,
- ② decode : décodage,
- ③ load operand : chargement des données,
- ④ execute : exécution,
- ⑤ result write back : mise à jour.

Instructions de base

Instructions

`commentaire` : aide pour l'utilisateur humain.

fin de ligne ignorée ↩

Instructions de base

Instructions

commentaire : aide pour l'utilisateur humain.

```
# fin de ligne ignorée ↵
```

instruction vide : ne rien faire.

```
pass
```


Instructions de base

Instructions

commentaire : aide pour l'utilisateur humain.

```
# fin de ligne ignorée ↩
```

instruction vide : ne rien faire.

```
pass
```

bloc d'instructions : regrouper plusieurs instructions en une seule.

```
instruction1
| instruction2.1
| ...
| instruction2.n
instruction3
```

noter l'**indentation** du bloc d'instructions 2

Instructions de base

Instructions

affectation : changer la valeur d'une variable.

```
variable = expression
```

Instructions de base

Instructions

affectation : changer la valeur d'une variable.

```
variable = expression
```

conditions : exécuter une instruction sous condition.

```
if condition : bloc  
[elif condition : bloc]*  
[else : bloc]
```

Instructions de base

Instructions

affectation : changer la valeur d'une variable.

```
variable = expression
```

conditions : exécuter une instruction sous condition.

```
if condition : bloc  
[elif condition : bloc]*  
[else : bloc]
```

itérations : répéter plusieurs fois la même instruction.

```
while condition : bloc
```

```
for element in sequence : bloc
```

Variables

Définition

Une variable est un objet informatique qui associe un nom à une valeur qui peut éventuellement varier au cours du temps
(une variable dénote une valeur).

Variables

Définition

Une variable est un objet informatique qui associe un nom à une valeur qui peut éventuellement varier au cours du temps
(une variable dénote une valeur).

Nom d'une variable

Le nom d'une variable est un identificateur aussi explicite que possible (exprimer le contenu sémantique de la variable).

Variables

Définition

Une variable est un objet informatique qui associe un nom à une valeur qui peut éventuellement varier au cours du temps (**une variable dénote une valeur**).

Nom d'une variable

Le nom d'une variable est un identificateur aussi explicite que possible (**exprimer le contenu sémantique de la variable**).

Exemples :

:-(:-)
x	pression
y	angleRotation
z	altitude

Variables

Définition

Une variable est un objet informatique qui associe un nom à une valeur qui peut éventuellement varier au cours du temps (**une variable dénote une valeur**).

Nom d'une variable

Le nom d'une variable est un identificateur aussi explicite que possible (**exprimer le contenu sémantique de la variable**).

Exemples :

:- (:-)
x	pression
y	angleRotation
z	altitude

:- (:-)
t	temps
u	masse
v	vitesse

Règles lexicales

Règles lexicales

- Un nom de variable est une séquence de lettres (a...z , A...Z) et de chiffres (0...9), qui doit toujours commencer par une lettre.

a2pique, jeanMartin, ieee754

Règles lexicales

Règles lexicales

- Un nom de variable est une séquence de lettres (a...z , A...Z) et de chiffres (0...9), qui doit toujours commencer par une lettre.
- Pas de lettres accentuées, de cédilles, d'espaces, de caractères spéciaux tels que \$, #, @, etc., à l'exception du caractère _ (souligné).

a2pique, jeanMartin, ieee754

vitesse_angulaire, element, ca_marche

Règles lexicales

Règles lexicales

- Un nom de variable est une séquence de lettres (a...z , A...Z) et de chiffres (0...9), qui doit toujours commencer par une lettre.
`a2pique`, `jeanMartin`, `ieee754`
- Pas de lettres accentuées, de cédilles, d'espaces, de caractères spéciaux tels que \$, #, @, etc., à l'exception du caractère _ (souligné).
`vitesse_angulaire`, `element`, `ca_marche`
- La casse est significative : les caractères majuscules et minuscules sont distingués.
`python` \neq `Python` \neq `PYTHON`

Conventions lexicales

Conventions lexicales

- *a priori*, n'utiliser que des lettres minuscules

<code>:- (</code>	<code>:-)</code>
<code>Variable</code>	<code>variable</code>

Conventions lexicales

Conventions lexicales

- *a priori*, n'utiliser que des lettres minuscules

<code>:-(</code>	<code>:-)</code>
Variable	variable

- n'utiliser les majuscules qu'à l'intérieur du nom pour augmenter la lisibilité

<code>:-(</code>	<code>:-)</code>
programmepython	programmePython

Conventions lexicales

Conventions lexicales

- *a priori*, n'utiliser que des lettres minuscules

<code>:-(</code>	<code>:-)</code>
Variable	variable

- n'utiliser les majuscules qu'à l'intérieur du nom pour augmenter la lisibilité

<code>:-(</code>	<code>:-)</code>
programmepython	programmePython

- nom de constante tout en majuscule

<code>:-(</code>	<code>:-)</code>
rouge	ROUGE

Affectation

Définition

Opération qui attribue une valeur à une variable.

... = ...

Affectation

Définition

Opération qui attribue une valeur à une variable.

... = ...

Valeur d'une constante

variable = constante

Affectation

Définition

Opération qui attribue une valeur à une variable.

`... = ...`

Valeur d'une constante

`variable = constante`

Valeur d'une expression

`variable = expression`

Affectation

Valeur d'une constante

```
variable = constante
```

La constante doit être une constante du type de la variable.

Affectation

Valeur d'une constante

```
variable = constante
```

La constante doit être une constante du type de la variable.

Exemple : initialisations

```
booleen = False
```

Affectation

Valeur d'une constante

```
variable = constante
```

La constante doit être une constante du type de la variable.

Exemple : initialisations

```
booléen = False
```

```
entier = 3
```

Affectation

Valeur d'une constante

```
variable = constante
```

La constante doit être une constante du type de la variable.

Exemple : initialisations

```
booléen = False
```

```
entier = 3
```

```
reel = 0.0
```

Affectation

Valeur d'une constante

```
variable = constante
```

La constante doit être une constante du type de la variable.

Exemple : initialisations

```
booléen = False
```

```
entier = 3
```

```
reel = 0.0
```

```
chaine = "salut"
```

Affectation

Valeur d'une constante

```
variable = constante
```

La constante doit être une constante du type de la variable.

Exemple : initialisations

```
booléen = False
```

```
entier = 3
```

```
reel = 0.0
```

```
chaine = "salut"
```

```
autreChaine = 'bonjour, comment ça va?'
```

Affectation

Valeur d'une constante

```
variable = constante
```

La constante doit être une constante du type de la variable.

Exemple : initialisations

```
booleen = False
```

```
entier = 3
```

```
reel = 0.0
```

```
chaine = "salut"
```

```
autreChaine = 'bonjour, comment ça va?'
```

```
tableau = [5,2,9,3]
```


Affectation

Valeur d'une constante

```
variable = constante
```

La constante doit être une constante du type de la variable.

Exemple : initialisations

```
booléen = False
```

```
entier = 3
```

```
reel = 0.0
```

```
chaine = "salut"
```

```
autreChaine = 'bonjour, comment ça va?'
```

```
tableau = [5,2,9,3]
```

```
matrice = [[1,2],[6,7],[9,1]]
```

Affectation

Valeur d'une expression

`variable = expression`

On évalue d'abord l'expression puis on affecte sa valeur à la variable.

Affectation

Valeur d'une expression

`variable = expression`

On évalue d'abord l'expression puis on affecte sa valeur à la variable.

Exemple : calculs

`somme = n*(n+1)/2`

■ TD 5.12

Affectation

Valeur d'une expression

`variable = expression`

On évalue d'abord l'expression puis on affecte sa valeur à la variable.

Exemple : calculs

`somme = n*(n+1)/2`

■ TD 5.12

`delta = b*b - 4*a*c`

Affectation

Valeur d'une expression

`variable = expression`

On évalue d'abord l'expression puis on affecte sa valeur à la variable.

Exemple : calculs

`somme = n*(n+1)/2`

■ TD 5.12

`delta = b*b - 4*a*c`

Exemple : échange de valeurs entre 2 variables

`tmp = x`

`x = y`

`y = tmp`

■ TD 5.3

Affectation

Exemple : modification

```
i = i + 1      # incrémentation
```

Affectation

Exemple : modification

`i = i + 1` # incrémentation

`i = i - 1` # décrémentation

Affectation

Exemple : modification

`i = i + 1` # incrémentation

`i = i - 1` # décrémentation

`q = q/b`

■ TD 5.4

Affectation

Exemple : modification

`i = i + 1` # incrémentation

`i = i - 1` # décrémentement

`q = q/b`

■ TD 5.4

Attention !

L'affectation est une opération typiquement informatique qui se distingue de l'égalité mathématique.

Affectation

Exemple : modification

`i = i + 1` # incrémentation

`i = i - 1` # décrémentation

`q = q/b`

■ TD 5.4

Attention !

L'affectation est une opération typiquement informatique qui se distingue de l'égalité mathématique.

En mathématique une expression du type `i = i+1` se réduit en `0 = 1` !

Affectation

Exemple : modification

$i = i + 1$ # incrémentation

$i = i - 1$ # décrémentation

$q = q/b$

■ TD 5.4

Attention !

L'affectation est une opération typiquement informatique qui se distingue de l'égalité mathématique.

En mathématique une expression du type $i = i+1$ se réduit en $0 = 1$!

En informatique, l'expression $i = i+1$ conduit à ajouter 1 à la valeur de i (évaluation de l'expression $i+1$), puis à donner cette nouvelle valeur à i (affectation).

Tests

Définition

Exécuter une instruction sous condition.

```
if condition : bloc  
[elif condition : bloc]*  
[else : bloc]
```

Les instructions entre crochets ([...]) sont optionnelles.

[...]* signifie que les instructions entre crochets peuvent être répétées 0 ou plusieurs fois.

Structure de contrôle effectuant un test et permettant un choix entre diverses parties du programme. On sort ainsi de l'exécution purement séquentielle des instructions.

Test simple

```
if condition : bloc
```

Condition : comparaison

x == y

x != y

x < y

x <= y

x > y

x >= y

Test simple

```
if condition : bloc
```

Condition : comparaison

x == y

x != y

x < y

x <= y

x > y

x >= y

```
if x < 0 : y = -x
```

Test simple

```
if condition : bloc
```

Condition : comparaison

x == y

x != y

x < y

x <= y

x > y

x >= y

```
if x < 0 : y = -x
```

```
if x != y : y = x
```

Test simple

```
if condition : bloc
```

Condition : comparaison

x == y

x != y

x < y

x <= y

x > y

x >= y

```
if x < 0 : y = -x
```

```
if x != y : y = x
```

Condition : calcul booléen

not a

a and b

a or b

Test simple

```
if condition : bloc
```

Condition : comparaison

```
x == y
```

```
x != y
```

```
x < y
```

```
x <= y
```

```
x > y
```

```
x >= y
```

```
if x < 0 : y = -x
```

```
if x != y : y = x
```

Condition : calcul booléen

```
not a
```

```
a and b
```

```
a or b
```

```
if (x > 0) and (x < 2) :  
    y = 3*x
```

Test simple

```
if condition : bloc
```

Condition : comparaison

x == y

x != y

x < y

x <= y

x > y

x >= y

```
if x < 0 : y = -x
```

```
if x != y : y = x
```

Condition : calcul booléen

not a

a and b

a or b

```
if (x > 0) and (x < 2) :
```

```
    y = 3*x
```

```
if (x <= 0) or (x >= 2) :
```

```
    y = 4*x
```

■ TD 5.5

Alternative simple

```
if condition : bloc  
else : bloc
```

Exemple : valeur absolue

```
if x < 0 :  
    valeurAbsolue = -x
```

Alternative simple

```
if condition : bloc  
else : bloc
```

Exemple : valeur absolue

```
if x < 0 :  
    valeurAbsolue = -x  
else :  
    valeurAbsolue = x
```

Alternative simple

```
if condition : bloc  
else : bloc
```

Exemple : valeur absolue

```
if x < 0 :  
    valeurAbsolue = -x  
else :  
    valeurAbsolue = x
```

Exemple : maximum

```
if x > y :  
    maximum = x
```

Alternative simple

```
if condition : bloc  
else : bloc
```

Exemple : valeur absolue

```
if x < 0 :  
    valeurAbsolue = -x  
else :  
    valeurAbsolue = x
```

Exemple : maximum

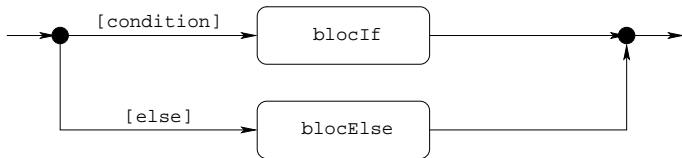
```
if x > y :  
    maximum = x  
else :  
    maximum = y
```

■ TD 5.6

■ TD 5.7

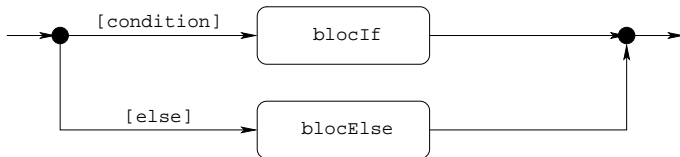
Alternatives simples

Alternative simple

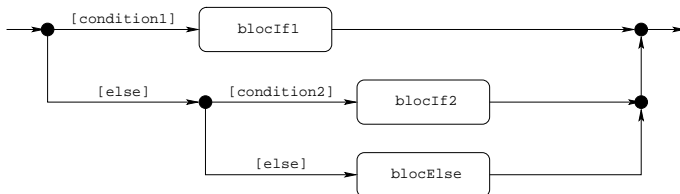


Alternatives simples

Alternative simple



Alternatives simples en cascade



Alternatives multiples

```
if condition : bloc  
elif condition : bloc  
...  
else : bloc
```

Exemple : mentions du bac

```
if note < 10 : mention = "ajourné"
```

Alternatives multiples

```
if condition : bloc  
elif condition : bloc  
...  
else : bloc
```

Exemple : mentions du bac

```
if note < 10 : mention = "ajourné"  
elif note < 12 : mention = "passable"
```

Alternatives multiples

```
if condition : bloc  
elif condition : bloc  
...  
else : bloc
```

Exemple : mentions du bac

```
if note < 10 : mention = "ajourné"  
elif note < 12 : mention = "passable"  
elif note < 14 : mention = "assez bien"
```

Alternatives multiples

```
if condition : bloc  
elif condition : bloc  
...  
else : bloc
```

Exemple : mentions du bac

```
if note < 10 : mention = "ajourné"  
elif note < 12 : mention = "passable"  
elif note < 14 : mention = "assez bien"  
elif note < 16 : mention = "bien"
```

Alternatives multiples

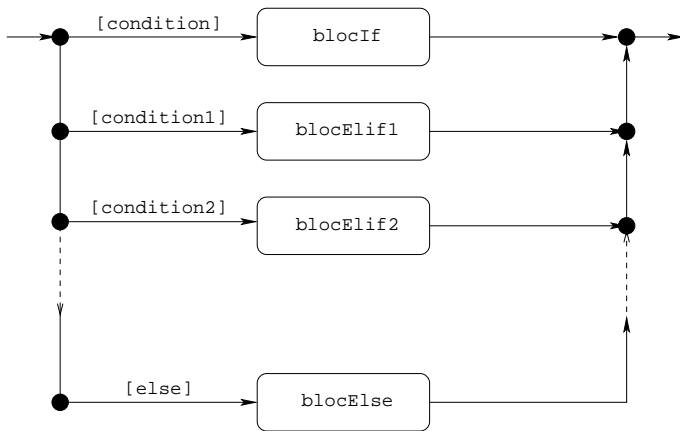
```
if condition : bloc  
elif condition : bloc  
...  
else : bloc
```

Exemple : mentions du bac

```
if note < 10 : mention = "ajourné"  
elif note < 12 : mention = "passable"  
elif note < 14 : mention = "assez bien"  
elif note < 16 : mention = "bien"  
else : mention = "très bien"
```

■ TD 5.15

Alternatives multiples



Boucles

Définition

Répétition d'un bloc d'instructions 0 ou plusieurs fois.

```
while condition : bloc
```

```
for element in sequence : bloc
```

Structures de contrôle destinées à être exécutées plusieurs fois (la structure de contrôle relançant l'exécution du bloc tant qu'une condition est remplie).

Itération conditionnelle

Boucle while

```
while condition : bloc
```


Itération conditionnelle

Boucle while

```
while condition : bloc
```

- Le bloc d'instructions d'une boucle **while** peut ne jamais être exécuté (condition non vérifiée la première fois).

Exemple : $i = 0$

```
while i > 0 : bloc
```

Itération conditionnelle

Boucle while

```
while condition : bloc
```

- Le bloc d'instructions d'une boucle **while** peut ne jamais être exécuté (condition non vérifiée la première fois).

Exemple : $i = 0$

```
while i > 0 : bloc
```

- On peut ne jamais sortir d'une boucle **while** (condition toujours vérifiée).

Exemple : **while** True : bloc

Exemple : élévation à la puissance

$$p = x^n$$

```
x = 2
n = 3
i = 0
p = 1
print x, n, p, i
while i < n :
    p = p * x
    i = i + 1
    print x, n, p, i
print x, n, p, i
```

Exemple : élévation à la puissance

$$p = x^n$$

```
x = 2
n = 3
i = 0
p = 1
print x, n, p, i
while i < n :
    p = p * x
    i = i + 1
    print x, n, p, i
print x, n, p, i
```

x	n	p	i
2	3	1	0

Exemple : élévation à la puissance

$$p = x^n$$

```
x = 2
n = 3
i = 0
p = 1
print x, n, p, i
while i < n :
    p = p * x
    i = i + 1
    print x, n, p, i
print x, n, p, i
```

x	n	p	i
2	3	1	0
2	3	2	1

Exemple : élévation à la puissance

$$p = x^n$$

```
x = 2
n = 3
i = 0
p = 1
print x, n, p, i
while i < n :
    p = p * x
    i = i + 1
    print x, n, p, i
print x, n, p, i
```

x	n	p	i
2	3	1	0
2	3	2	1
2	3	4	2

Exemple : élévation à la puissance

$$p = x^n$$

```
x = 2
n = 3
i = 0
p = 1
print x, n, p, i
while i < n :
    p = p * x
    i = i + 1
    print x, n, p, i
print x, n, p, i
```

x	n	p	i
2	3	1	0
2	3	2	1
2	3	4	2
2	3	8	3

Exemple : élévation à la puissance

$$p = x^n$$

```
x = 2
n = 3
i = 0
p = 1
print x, n, p, i
while i < n :
    p = p * x
    i = i + 1
    print x, n, p, i
print x, n, p, i
```

x	n	p	i
2	3	1	0
2	3	2	1
2	3	4	2
2	3	8	3
2	3	8	3

$$p = 8 = 2^3 = x^n$$

Exemple : division entière

$$q = a \div b, r = a \bmod b$$

```
a = 8
b = 3
q = 0
r = a
print a, b, r, q
while r >= b :
    r = r - b
    q = q + 1
    print a, b, r, q
print a, b, r, q
```

Exemple : division entière

$$q = a \div b, r = a \bmod b$$

```
a = 8
b = 3
q = 0
r = a
print a, b, r, q
while r >= b :
    r = r - b
    q = q + 1
    print a, b, r, q
print a, b, r, q
```

<i>a</i>	<i>b</i>	<i>r</i>	<i>q</i>
8	3	8	0

Exemple : division entière

$$q = a \div b, r = a \bmod b$$

```
a = 8
b = 3
q = 0
r = a
print a, b, r, q
while r >= b :
    r = r - b
    q = q + 1
    print a, b, r, q
print a, b, r, q
```

<i>a</i>	<i>b</i>	<i>r</i>	<i>q</i>
8	3	8	0
8	3	5	1

Exemple : division entière

$$q = a \div b, r = a \bmod b$$

```
a = 8
b = 3
q = 0
r = a
print a, b, r, q
while r >= b :
    r = r - b
    q = q + 1
    print a, b, r, q
print a, b, r, q
```

<i>a</i>	<i>b</i>	<i>r</i>	<i>q</i>
8	3	8	0
8	3	5	1
8	3	2	2

Exemple : division entière

$$q = a \div b, r = a \bmod b$$

```
a = 8
b = 3
q = 0
r = a
print a, b, r, q
while r >= b :
    r = r - b
    q = q + 1
    print a, b, r, q
print a, b, r, q
```

<i>a</i>	<i>b</i>	<i>r</i>	<i>q</i>
8	3	8	0
8	3	5	1
8	3	2	2
8	3	2	2

$$a = bq + r = 3 \cdot 2 + 2 = 8$$

Exemple : racine carrée entière

$$r = \sqrt{n}$$

```
n = 17
r = 0
print n, r
while (r+1)**2 <= n :
    r = r + 1
    print n, r
print n, r
```

Exemple : racine carrée entière

$$r = \sqrt{n}$$

```
n = 17
r = 0
print n, r
while (r+1)**2 <= n :
    r = r + 1
    print n, r
print n, r
```

n	r
17	0

Exemple : racine carrée entière

$$r = \sqrt{n}$$

```
n = 17
r = 0
print n, r
while (r+1)**2 <= n :
    r = r + 1
    print n, r
print n, r
```

n	r
17	0
17	1

Exemple : racine carrée entière

$$r = \sqrt{n}$$

```
n = 17
r = 0
print n, r
while (r+1)**2 <= n :
    r = r + 1
    print n, r
print n, r
```

n	r
17	0
17	1
17	2

Exemple : racine carrée entière

$$r = \sqrt{n}$$

```
n = 17
r = 0
print n, r
while (r+1)**2 <= n :
    r = r + 1
    print n, r
print n, r
```

n	r
17	0
17	1
17	2
17	3

Exemple : racine carrée entière

$$r = \sqrt{n}$$

```
n = 17
r = 0
print n, r
while (r+1)**2 <= n :
    r = r + 1
    print n, r
print n, r
```

n	r
17	0
17	1
17	2
17	3
17	4

Exemple : racine carrée entière

$$r = \sqrt{n}$$

```
n = 17
r = 0
print n, r
while (r+1)**2 <= n :
    r = r + 1
    print n, r
print n, r
```

n	r
17	0
17	1
17	2
17	3
17	4
17	4

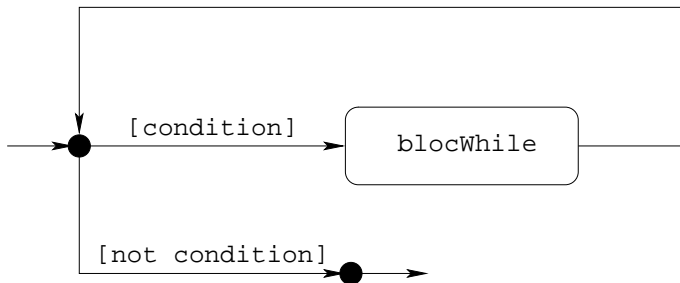
$$r^2 = 4^2 = 16 \leq 17 = n$$

$$n = 17 < (r+1)^2 = 5^2 = 25$$

$$r^2 \leq n < (r+1)^2$$

Itération conditionnelle

Boucle while



■ TD 5.16

Boucle for

Boucle for

```
for element in sequence : bloc
```

La séquence peut être

- une séquence explicite

Exemples : [5,6,7],

Boucle for

Boucle for

```
for element in sequence : bloc
```

La séquence peut être

- une séquence explicite

Exemples : [5,6,7],[3,-2,0,1],

Boucle for

Boucle for

```
for element in sequence : bloc
```

La séquence peut être

- une séquence explicite

Exemples : [5,6,7],[3,-2,0,1],["a","b"]

Boucle for

Boucle for

```
for element in sequence : bloc
```

La séquence peut être

- une séquence explicite

Exemples : `[5,6,7]`, `[3,-2,0,1]`, `["a","b"]`

- une séquence calculée (`range(min,max,pas)`)

Exemples : `range(0,5,2)` → `[0,2,4]`

Boucle for

Boucle for

```
for element in sequence : bloc
```

La séquence peut être

- une séquence explicite

Exemples : `[5,6,7]`, `[3,-2,0,1]`, `["a","b"]`

- une séquence calculée (`range(min,max,pas)`)

Exemples : `range(0,5,2) → [0,2,4]`

`range(0,3,1) → [0,1,2]`

Boucle for

Boucle for

```
for element in sequence : bloc
```

La séquence peut être

- une séquence explicite

Exemples : `[5,6,7]`, `[3,-2,0,1]`, `["a","b"]`

- une séquence calculée (`range(min,max,pas)`)

Exemples : `range(0,5,2) → [0,2,4]`

`range(0,3,1) → [0,1,2]`

`range(0,3) → [0,1,2]`

Boucle for

Boucle for

```
for element in sequence : bloc
```

La séquence peut être

- une séquence explicite

Exemples : `[5,6,7]`, `[3,-2,0,1]`, `["a","b"]`

- une séquence calculée (`range(min,max,pas)`)

Exemples : `range(0,5,2) → [0,2,4]`

`range(0,3,1) → [0,1,2]`

`range(0,3) → [0,1,2]`

`range(3) → [0,1,2]`

Exemple : somme arithmétique

$$s = \sum_{i=1}^{i=n} i$$

```
n = 4
s = 0
print n, i, s
for i in range(1,n+1) :
    s = s + i
    print n, i, s
print n, i, s
```

Exemple : somme arithmétique

$$s = \sum_{i=1}^{i=n} i$$

```
n = 4
s = 0
print n, i, s
for i in range(1,n+1) :
    s = s + i
    print n, i, s
print n, i, s
```

n	i	s
4	?	0

Exemple : somme arithmétique

$$s = \sum_{i=1}^{i=n} i$$

```
n = 4
s = 0
print n, i, s
for i in range(1,n+1) :
    s = s + i
    print n, i, s
print n, i, s
```

n	i	s
4	?	0
4	1	1

Exemple : somme arithmétique

$$s = \sum_{i=1}^{i=n} i$$

```
n = 4
s = 0
print n, i, s
for i in range(1,n+1) :
    s = s + i
    print n, i, s
print n, i, s
```

n	i	s
4	?	0
4	1	1
4	2	3

Exemple : somme arithmétique

$$s = \sum_{i=1}^{i=n} i$$

```
n = 4
s = 0
print n, i, s
for i in range(1,n+1) :
    s = s + i
    print n, i, s
print n, i, s
```

n	i	s
4	?	0
4	1	1
4	2	3
4	3	6

Exemple : somme arithmétique

$$s = \sum_{i=1}^{i=n} i$$

```
n = 4
s = 0
print n, i, s
for i in range(1,n+1) :
    s = s + i
    print n, i, s
print n, i, s
```

<i>n</i>	<i>i</i>	<i>s</i>
4	?	0
4	1	1
4	2	3
4	3	6
4	4	10

Exemple : somme arithmétique

$$s = \sum_{i=1}^{i=n} i$$

```
n = 4
s = 0
print n, i, s
for i in range(1,n+1) :
    s = s + i
    print n, i, s
print n, i, s
```

n	i	s
4	?	0
4	1	1
4	2	3
4	3	6
4	4	10
4	4	10

$$s = 1+2+3+4 = 10 = \sum_{i=1}^{i=4} i$$

Exemple : factorielle

$$f = n! = \prod_{i=1}^{i=n} i$$

```
n = 4
f = 1
print n, i, f
for i in range(1,n+1) :
    f = f * i
    print n, i, f
print n, i, f
```

Exemple : factorielle

$$f = n! = \prod_{i=1}^{i=n} i$$

```
n = 4
f = 1
print n, i, f
for i in range(1,n+1) :
    f = f * i
    print n, i, f
print n, i, f
```

n	i	f
4	?	1

Exemple : factorielle

$$f = n! = \prod_{i=1}^{i=n} i$$

```
n = 4
f = 1
print n, i, f
for i in range(1,n+1) :
    f = f * i
    print n, i, f
print n, i, f
```

n	i	f
4	?	1
4	1	1

Exemple : factorielle

$$f = n! = \prod_{i=1}^{i=n} i$$

```
n = 4
f = 1
print n, i, f
for i in range(1,n+1) :
    f = f * i
    print n, i, f
print n, i, f
```

n	i	f
4	?	1
4	1	1
4	2	2

Exemple : factorielle

$$f = n! = \prod_{i=1}^{i=n} i$$

```
n = 4
f = 1
print n, i, f
for i in range(1,n+1) :
    f = f * i
    print n, i, f
print n, i, f
```

n	i	f
4	?	1
4	1	1
4	2	2
4	3	6

Exemple : factorielle

$$f = n! = \prod_{i=1}^{i=n} i$$

```
n = 4
f = 1
print n, i, f
for i in range(1,n+1) :
    f = f * i
    print n, i, f
print n, i, f
```

n	i	f
4	?	1
4	1	1
4	2	2
4	3	6
4	4	24

Exemple : factorielle

$$f = n! = \prod_{i=1}^{i=n} i$$

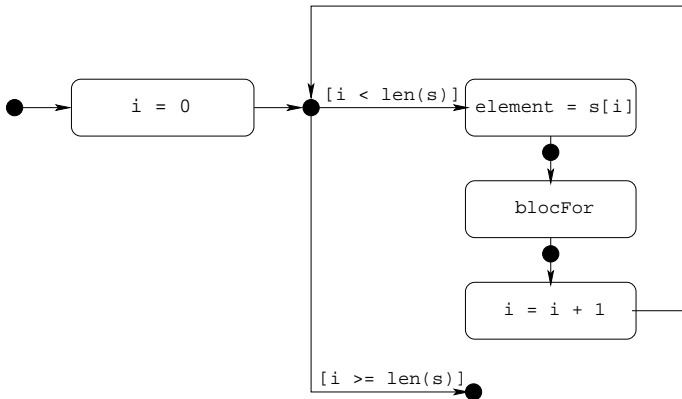
```
n = 4
f = 1
print n, i, f
for i in range(1,n+1) :
    f = f * i
    print n, i, f
print n, i, f
```

n	i	f
4	?	1
4	1	1
4	2	2
4	3	6
4	4	24
4	4	24

$$s = 1 \cdot 2 \cdot 3 \cdot 4 = 24 = \prod_{i=1}^{i=4} i$$

Parcours de séquence

Boucle for



Equivalence boucles `for` et `while`

```
for i in range(min,max,pas) :  
    bloc
```

Equivalence boucles `for` et `while`

```
for i in range(min,max,pas) :  
    bloc
```

élévation à la puissance

```
p = 1  
for i in range(n) :  
    p = p * x
```

Equivalence boucles `for` et `while`

```
for i in range(min,max,pas) :  
    bloc
```

élévation à la puissance

```
p = 1  
for i in range(n) :  
    p = p * x
```

```
i = min  
while i < max :  
    bloc  
    i = i + pas
```

Equivalence boucles `for` et `while`

```
for i in range(min,max,pas) :  
    bloc
```

élévation à la puissance

```
p = 1  
for i in range(n) :  
    p = p * x
```

```
i = min  
while i < max :  
    bloc  
    i = i + pas
```

```
p = 1  
i = 0  
while i < n :  
    p = p * x  
    i = i + 1
```

Itérations

Séquences d'itérations

Qu'affichent les séquences d'itérations suivantes ?

```
s = 0
for i in range(5) :
    x = input('entrer un nombre : ')
    s = s + x
print s
```


Itérations

Séquences d'itérations

Qu'affichent les séquences d'itérations suivantes ?

```
s = 0
for i in range(5) :
    x = input('entrer un nombre : ')
    s = s + x
print s
```

```
for i in range(10) :
    j = 10 - i
    while j > 0 :
        print '*',
        j = j - 1
    print
```