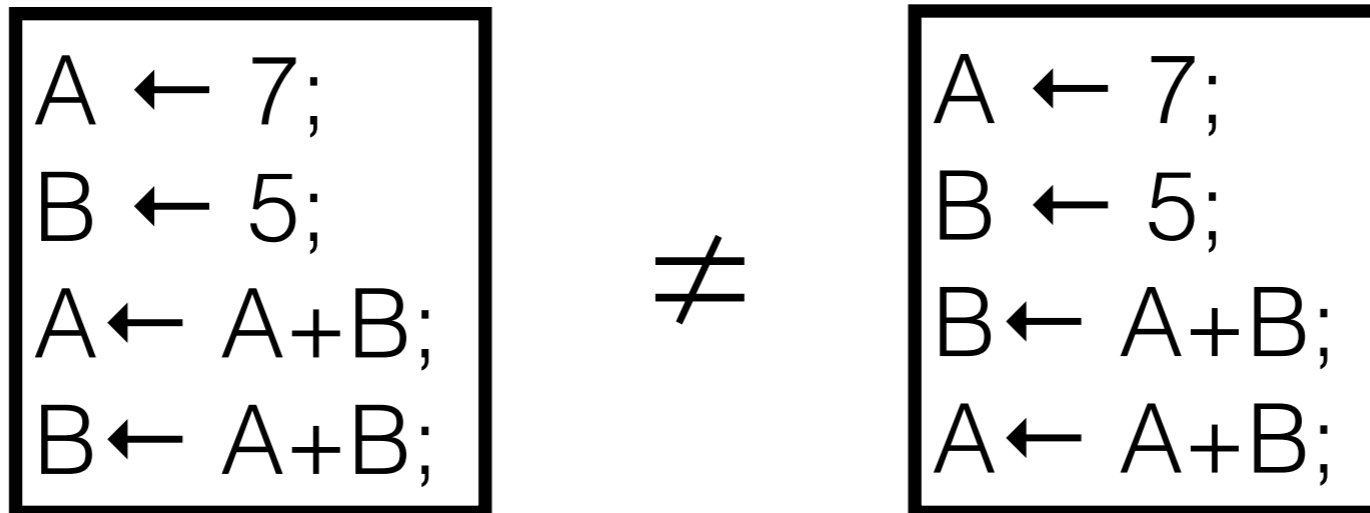


# Cours d'introduction à l'informatique

Partie 3 : Structures de contrôle  
Comment répéter une instruction ou l'éviter ?

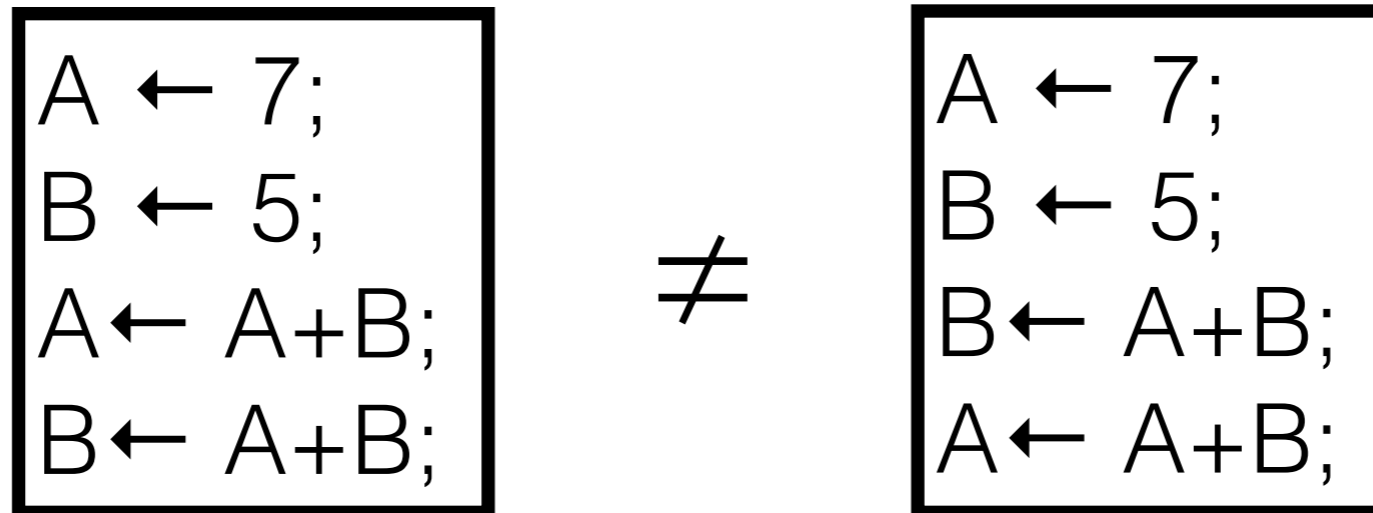
# Petit rappel

- L'organisation basique d'une suite d'instructions est le séquençement : Inst\_1 ; Inst\_2
- Effet : Exécuter l'instruction Inst\_1, puis l'instruction Inst\_2



# Petit rappel

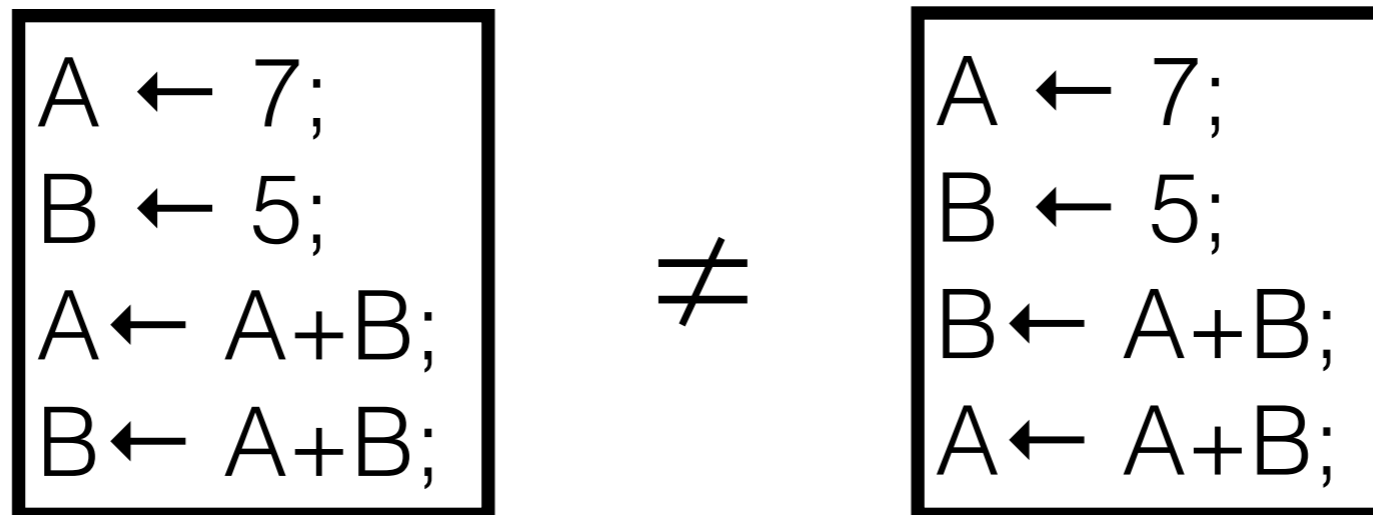
- L'organisation basique d'une suite d'instructions est le séquençement : Inst\_1 ; Inst\_2
- Effet : Exécuter l'instruction Inst\_1, puis l'instruction Inst\_2



Instructions	A	B
Avant 1	?	?
Après 1	7	?
Après 2	7	5
Après 3	12	5
Après 4	12	17

# Petit rappel

- L'organisation basique d'une suite d'instructions est le séquençement : Inst\_1 ; Inst\_2
- Effet : Exécuter l'instruction Inst\_1, puis l'instruction Inst\_2



Instructions	A	B
Avant 1	?	?
Après 1	7	?
Après 2	7	5
Après 3	12	5
Après 4	12	17

Instructions	A	B
Avant 1	?	?
Après 1	7	?
Après 2	7	5
Après 3	7	12
Après 4	19	12

# Insuffisances du séquençement

- Mais c'est insuffisant !
- Plusieurs cas possibles :
  - on ne veut pas toujours faire la même chose
  - on a besoin de répéter, d'itérer plusieurs fois la même chose.

# Insuffisances du séquençement

- Mais Modifier l'ordre naturel des instructions se fait à l'aide de structures de
- Plus contrôle: les structures conditionnelles et les structures répétitives....
- on ne veut pas toujours faire la même chose
- on a besoin de répéter, d'itérer plusieurs fois la même chose.

# Les conditionnelles

Syntaxe et usage, conditionnelles imbriquées et arbres de décision...

# Motivations

- Valeur absolue d'un nombre: c'est  $x$  si  $x > 0$  et  $-x$  sinon
- Affichage de l'inverse d'un nombre: que fait-on de  $1/0$  ?
- Jouer au jeu du plus ou moins
- Tous les cas où il y a des traitements alternatifs



# La syntaxe des alternatives

**si** (condition)

**Alors** Instructions\_alors;

**Sinon** Instructions\_sinon;

**fin si**

exemple:

**si** (age < 18)

**Alors** Ecrire('Vous êtes mineur(e)');

**Sinon** Ecrire('Vous êtes majeur(e)');

**fin si**

# La syntaxe des alternatives

n'importe quelle expression dont la valeur est booléenne

**si** (condition)

**Alors** Instructions\_alors;

**Sinon** Instructions\_sinon;

**fin si**

exemple:

**si** (age < 18)

**Alors** Ecrire('Vous êtes mineur(e)');

**Sinon** Ecrire('Vous êtes majeur(e)');

**fin si**

# La conditionnelle simple

- Cas particulier de l'alternative  
**si** (condition)  
    **Alors** Instructions\_alors;  
**fin si**

exemple:

```
si (x<0)  
    Alors x ← -x;  
fin si  
Ecrire(x);
```

# En javascript

**si** (age < 18)

**Alors** Ecrire('Vous êtes mineur(e)');

**Sinon** Ecrire('Vous êtes majeur(e)');

**fin si**

devient

**if** (age < 18) {

Ecrire('Vous êtes mineur(e)');

**} else {**

Ecrire('Vous êtes majeur(e)');

**}**

# En javascript

**si** (age < 18)

**Alors** Ecrire('Vous êtes mineur(e)');

**Sinon** Ecrire('Vous êtes majeur(e)').

**fin si**

devient

**if** (age < 18) {

Ecrire('Vous êtes

**} else {**

Ecrire('Vous êtes majeur(e)');

**}**

```
si (x < 0)
```

```
Alors x ← -x;
```

```
fin si
```

devient

```
if (x < 0) {
```

```
  x = -x;
```

```
}
```

# Les historiques d'exécution pour les conditionnelles

- Dans le cas d'une conditionnelle, l'ordre dans lequel sont évaluées les instructions n'est pas fixe.

Algorithme Majorité

Variables:

age: entier;

statut: chaîne de caractères;

Début

age ← Saisie();

si (age < 18)

Alors statut ← 'mineur';

Sinon statut ← 'majeur';

fin si

Ecrire(statut);

Fin

# Les historiques d'exécution pour les conditionnelles

- Dans le cas d'une conditionnelle, l'ordre dans lequel sont évaluées les instructions n'est pas fixe.

Algorithme Majorité

Variables:

age: entier;

statut: chaîne de caractères;

Début

1 age ← Saisie();

2 si (age < 18)

3     Alors statut ← 'mineur';

4     Sinon statut ← 'majeur';

fin si

5 Ecrire(statut);

Fin

# Les historiques d'exécution pour les conditionnelles

- Dans le cas d'une conditionnelle, l'ordre dans lequel sont évaluées les instructions n'est pas fixe.

Algorithme Majorité

Variables:

age: entier;

statut: chaîne de caractères;

Début

1 age ← Saisie();

2 si (age < 18)

3     Alors statut ← 'mineur';

4     Sinon statut ← 'majeur';

fin si

5 Ecrire(statut);

Fin



# Les historiques d'exécution pour les conditionnelles

- Dans le cas d'une conditionnelle, l'ordre dans lequel sont évaluées les instructions n'est pas fixe.

## Algorithme Majorité

### Variables:

age: entier;  
statut: chaîne de caractères;

### Début

```
1 age ← Saisie();  
2 si (age < 18)  
3     Alors statut ← 'mineur';  
4     Sinon statut ← 'majeur';  
   fin si  
5 Ecrire(statut);
```

### Fin

Instructions	age	statut
Avant 1	?	?
Après 1		?

# Les historiques d'exécution pour les conditionnelles

- Dans le cas d'une conditionnelle, l'ordre dans lequel sont évaluées les instructions n'est pas fixe.

## Algorithme Majorité

### Variables:

age: entier;  
statut: chaîne de caractères;

### Début

```
1 age ← Saisie();  
2 si (age < 18)  
3     Alors statut ← 'mineur';  
4     Sinon statut ← 'majeur';  
   fin si  
5 Ecrire(statut);
```

### Fin

Instructions	age	statut
Avant 1	?	?
Après 1	15	?
Après 3	15	'mineur'
Après 2	15	'mineur'
Après 5	15	'mineur'

# Les historiques d'exécution pour les conditionnelles

- Dans le cas d'une conditionnelle, l'ordre dans lequel sont évaluées les instructions n'est pas fixe.

## Algorithme Majorité

### Variables:

age: entier;  
statut: chaîne de caractères;

### Début

```
1 age ← Saisie();  
2 si (age < 18)  
3     Alors statut ← 'mineur';  
4     Sinon statut ← 'majeur';  
   fin si  
5 Ecrire(statut);
```

### Fin

Instructions	age	statut
Avant 1	?	?
Après 1		?

# Les historiques d'exécution pour les conditionnelles

- Dans le cas d'une conditionnelle, l'ordre dans lequel sont évaluées les instructions n'est pas fixe.

## Algorithme Majorité

### Variables:

age: entier;  
statut: chaîne de caractères;

### Début

```
1 age ← Saisie();  
2 si (age < 18)  
3     Alors statut ← 'mineur';  
4     Sinon statut ← 'majeur';  
   fin si  
5 Ecrire(statut);
```

### Fin

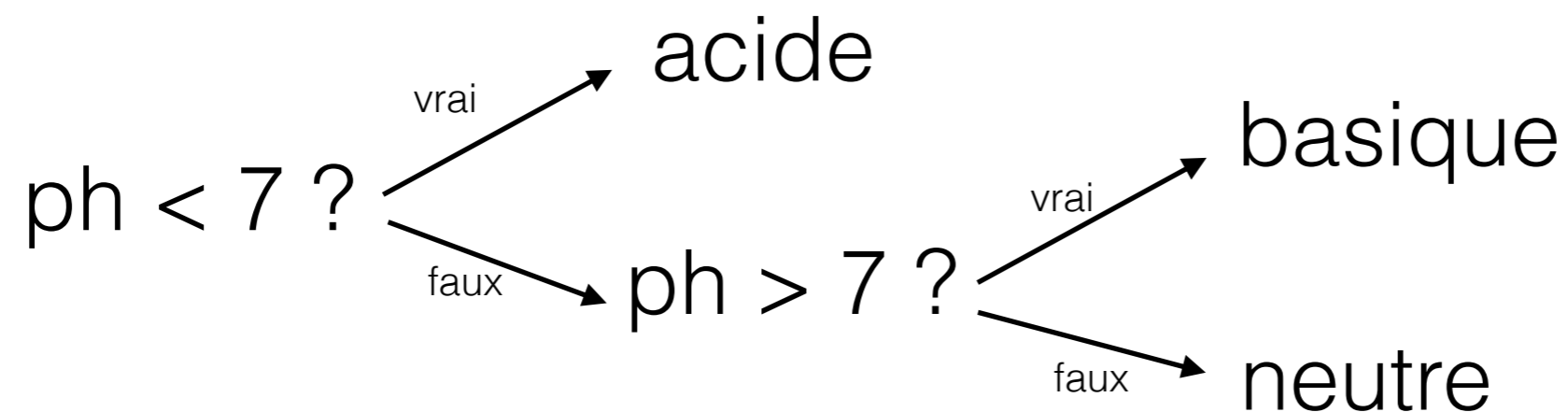
Instructions	age	statut
Avant 1	?	?
Après 1	21	?
Après 4	21	'majeur'
Après 2	21	'majeur'
Après 5	21	'majeur'

# Les conditionnelles imbriquées

- Les instructions que l'on peut mettre le bloc d'instructions du Alors, tout comme celle que l'on peut mettre dans le bloc du Sinon sont quelconques.
- On peut très bien mettre une conditionnelle !
- On parle alors de conditionnelles imbriquées.
- Cela sert notamment à implémenter des arbres de décision.

# Un cas pratique

- Un ph peut être basique, neutre ou acide. Il y a trois possibilités.
- Une alternative ne permet pas de les distinguer.
- Le problème se résout en utilisant l'arbre de décision suivant:



# Un cas pratique

- Un ph peut être classé en trois possibilités
- Une alternative
- Le problème de décision

Algorithme test de ph

Variables:

ph: entier;  
statut: chaîne de caractères;

Début

ph ← Saisie();

**si** (ph < 7)

**Alors** statut ← 'acide';

**Sinon** **si** (ph > 7)

**Alors** statut ← 'basique';

**Sinon** statut ← 'neutre';

**fin si**

**fin si**

Ecrire(statut);

Fin

ph < 7 ?

faux

ph > 7 ?

faux

neutre

# Un cas pratique

- Un ph peut être classé en trois possibilités
- Une alternative
- Le problème de décision

Avoir une bonne indentation de l'algorithme est primordiale pour en simplifier la compréhension et la lecture !!!

```
pn ← saisie();  
si (ph < 7)  
    Alors statut ← 'acide';  
    Sinon si (ph > 7)  
        Alors statut ← 'basique';  
        Sinon statut ← 'neutre';  
    fin si  
fin si  
Ecrire(statut);
```

ph < 7 ?

Fin  
faux

ph > 7 ?

faux

neutre



# Un cas pratique

- Un ph peut être classé en trois possibilités
- Une alternative
- Le problème de décision

Avoir une bonne indentation de l'algorithme est primordiale pour en simplifier la compréhension et la lecture !!!

```
ph ← saisie();  
si (ph < 7)  
  Alors statut ← 'acide';  
  Sinon si (ph > 7)  
    Alors statut ← 'basique';  
    Sinon statut ← 'neutre';  
  fin si  
fin si
```

ph < 7 ?

Fin  
faux

ph > 7 ?

faux

neutre

# Les arbres de décision

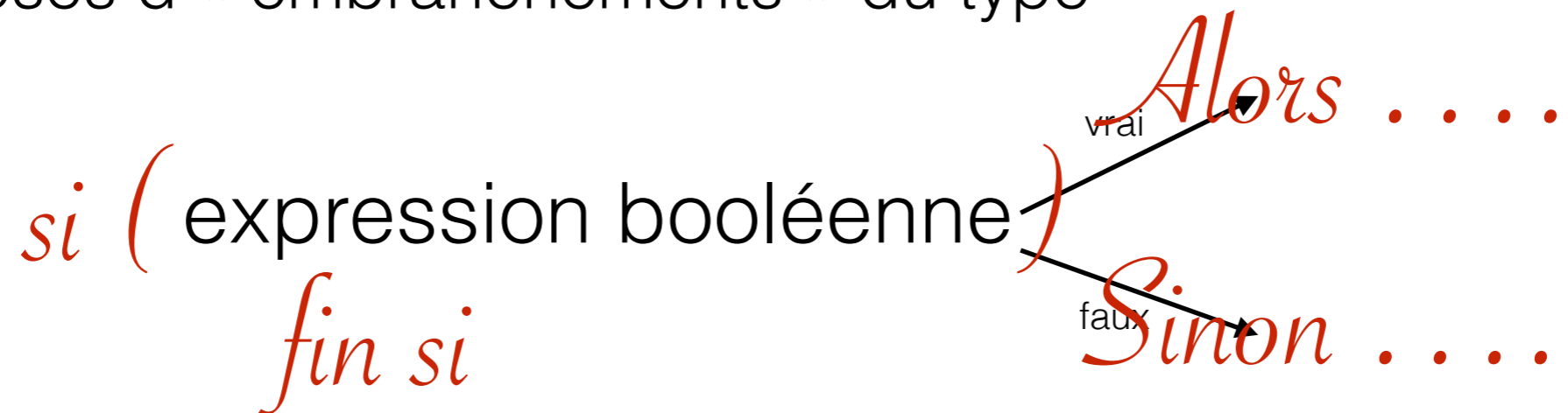
- Composés d'« embranchements » du type



- Décrivent la succession de tests à réaliser pour prendre une décision (complexe) finale.
- Souvent, il existe plusieurs arbres permettant de prendre la même décision.
- On peut passer de manière « quasi-automatique » de l'arbre de décision à son écriture dans le langage algorithmique.

# Les arbres de décision

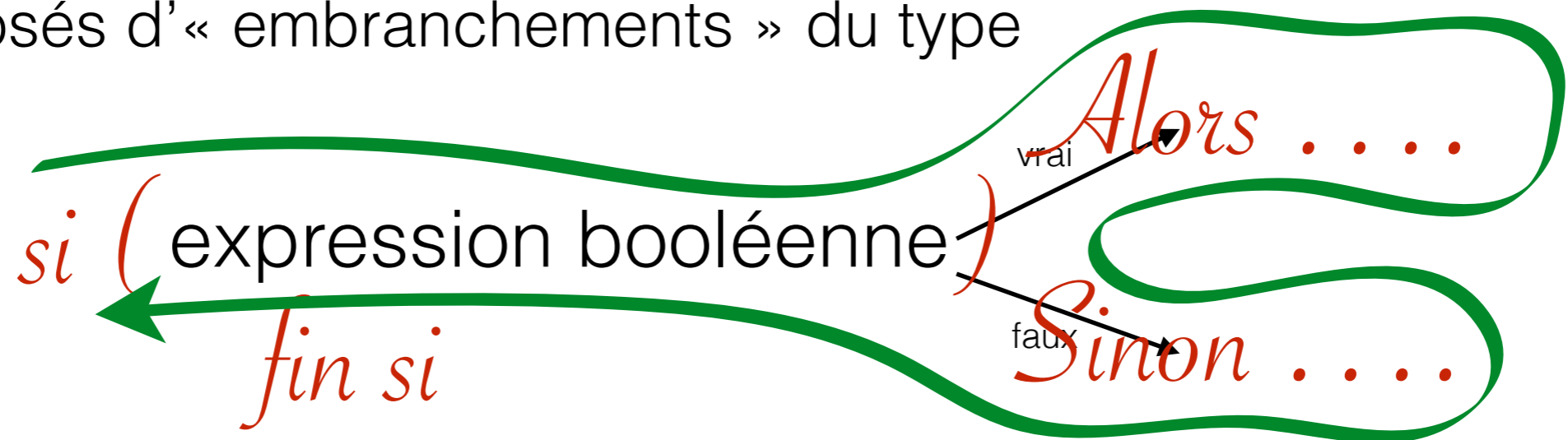
- Composés d'« embranchements » du type



- Décrivent la succession de tests à réaliser pour prendre une décision (complexe) finale.
- Souvent, il existe plusieurs arbres permettant de prendre la même décision.
- On peut passer de manière « quasi-automatique » de l'arbre de décision à son écriture dans le langage algorithmique.

# Les arbres de décision

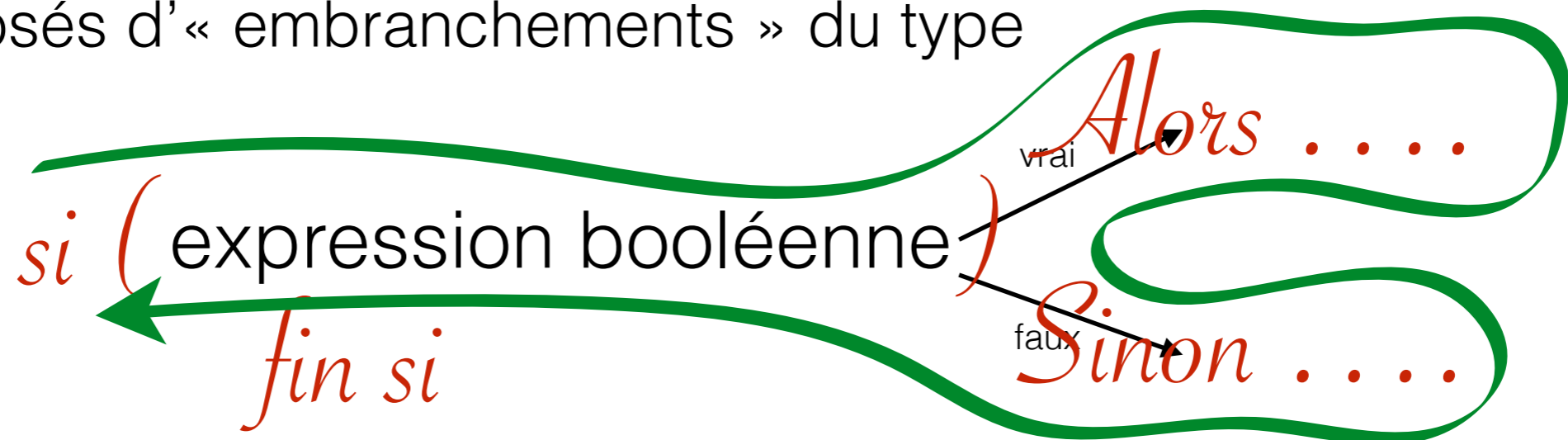
- Composés d'« embranchements » du type



- Décrivent la succession de tests à réaliser pour prendre une décision (complexe) finale.
- Souvent, il existe plusieurs arbres permettant de prendre la même décision.
- On peut passer de manière « quasi-automatique » de l'arbre de décision à son écriture dans le langage algorithmique.

# Les arbres de décision

- Composés d'« embranchements » du type



- Décrivent la succession de tests à réaliser pour prendre une décision (complexe) finale.
- Souvent, il existe plusieurs arbres pour la même décision.
- On peut passer de manière « quasi » de la structure de décision à son écriture dans le langage algorithmique.

*si (expression booléenne)*

*Alors ...*

*Sinon ...*

*fin si*

# Les répétitives

Syntaxes et usage, doubles boucles...

# Motivations

- On a très souvent envie de répéter une ou plusieurs instructions.
- Parfois on sait combien de fois on doit répéter, parfois on l'ignore.
- On ne veut pas écrire des choses du type.....

# Motivations

Calcul de  $3^{10}$

```
x ← 1;  
x ← 3 * x;  
x ← 3 * x;  
x ← 3 * x;  
x ← 3 * x;  
x ← 3 * x;  
x ← 3 * x;  
x ← 3 * x;  
x ← 3 * x;  
x ← 3 * x;  
x ← 3 * x;
```

ès souvent envie de répéter une ou  
irs instructions.

on sait combien de fois on doit répéter,  
on l'ignore.

veut pas écrire des choses du type.....









# La boucle « pour »

- C'est une boucle qui intègre un compteur de boucle.
- C'est la boucle à utiliser lorsqu'on connaît à l'avance le nombre de tours de boucle qui doivent être réalisés.

**pour** variable **allant de** val\_début **à** val\_fin **faire**  
instructions à répéter  
**fin pour**

# La boucle « pour »

- C'est une boucle qui intègre un compteur de

Algorithme Calcul de  $3^{10}$

Variables:

x,i: entiers;

- Début

x ← 1;

pour i allant de 1 à 10 faire

    x ← 3 \* x;

fin pour

Ecrire(x);

Fin

er lorsqu'on connaît à  
tours de boucle qui doivent

**pour** variable **allant de** val\_début **à** val\_fin **faire**  
instructions à répéter  
**fin pour**

# La boucle « pour »

- C'est une boucle qui intègre un compteur de

Algorithme Calcul de  $3^{100}$

Variables:

x,i: entiers;

- Début

x ← 1;

pour i allant de 1 à 100 faire

    x ← 3 \* x;

fin pour

Ecrire(x);

Fin

lorsqu'on connaît à  
l'avance le nombre de tours de boucle qui doivent

**pour** variable **allant de** val\_début **à** val\_fin **faire**  
instructions à répéter  
**fin pour**

# La boucle « pour »

- C'est une boucle qui intègre un compteur de

Algorithmes Calcul de 20 !

Variables:

x,i: entiers;

- Début

x ← 1;

pour i allant de 1 à 20 faire

    x ← i \* x;

fin pour

Ecrire(x);

Fin

lorsqu'on connaît à  
l'avance le nombre de tours de boucle qui doivent

**pour** variable **allant de** val\_début **à** val\_fin **faire**  
instructions à répéter  
**fin pour**

# La boucle « pour »

- C'est une boucle qui intègre un compteur de

Algorithmes Calcul de 20 !

Variables:

x, i: entiers

Début

x ← 1;

pour i allant de 1 à 20 faire

x ← i \* x;

fin pour

Ecrire(x);

Fin

i est une variable entière déclarée dans l'algorithme  
Elle est incrémentée de 1 à chaque tour de boucle  
Elle peut être utilisée à l'intérieur de la boucle

- C'est une boucle qui intègre un compteur de tours de boucle qui doivent

**pour** variable **allant de** val\_début **à** val\_fin **faire**  
instructions à répéter  
**fin pour**



# en Javascript

- La boucle « pour » s'écrit:

```
for(i=1; i <= 10; i=i+1) {  
    instructions  
}
```

# en Javascript

- La boucle « pour » s'écrit:

```
for(i=1; i <= 10; i=i+1) {  
    instructions  
}
```

```
// Algorithme polygone  
var n,i;  
n = Saisie();  
for(i=1; i <= n; i=i+1) {  
    Avancer(10);  
    Droite(360/n);  
}
```

# en Javascript

- La boucle « pour » s'écrit:

```
for(i=1; i <= 10; i=i+1) {  
  instructions  
}
```

```
// Algorithme polygone  
var n,i;  
n = Saisie();  
for(i=1; i <= n; i=i+1) {  
  Avancer(10);  
  Droite(360/n);  
}
```

Affectation de la valeur de départ à la variable de boucle

# en Javascript

Une expression booléenne de test de continuation

- La boucle « pour » s'écrit:

```
for(i=1; i <= 10; i=i+1) {  
  instructions  
}
```

```
// Algorithme polygone  
var n,i;  
n = Saisie();  
for(i=1; i <= n; i=i+1) {  
  Avancer(10);  
  Droite(360/n);  
}
```

Affectation de la valeur de départ à la variable de boucle

# en Javascript

Une expression booléenne de test de continuation

- La boucle « pour » s'écrit:

```
for(i=1; i <= 10; i=i+1) {  
  instructions  
}
```

```
// Algorithme polygone  
var n,i;  
n = Saisie();  
for(i=1; i <= n; i=i+1) {  
  Avancer(10);  
  Droite(360/n);  
}
```

Affectation de la valeur de départ à la variable de boucle

Affectation: comment varie la variable de boucle à chaque tour de boucle

# La boucle « tant que »

- C'est une boucle « universelle ».
- C'est la boucle à utiliser lorsqu'on **ne** connaît **pas** à l'avance le nombre de tours de boucle qui doivent être réalisés.

**Tant que** (expression booléenne) **faire**  
instructions à répéter  
**fin tant que**

# La boucle « tant que »

- C'est une boucle
- C'est la boucle l'avance le non être réalisés.

Algorithme Calcul de  $\lceil \log_3(100) \rceil$

Variables:

x, i: entiers;

Début

x ← 1;

i ← 0;

Tant que (x < 100) faire

    x ← 3 \* x;

    i ← i+1;

fin tant que

Ecrire(i-1);

Fin

connaît **pas** à  
e qui doivent

**Tant que** (expres  
instructions à répéter  
**fin tant que**

# Les dangers

**Tant que** (expression booléenne) **faire**  
instructions à répéter  
**fin tant que**

- Si le test est faux dès le début, aucune instruction n'est exécutée. Le nombre de tours de boucle peut donc être 0, 1, 2, .....
- Si le test est toujours vrai, on tombe dans le cas d'une boucle infinie. **C'est un cas à éviter !!!**
- Les instructions doivent donc modifier la valeur calculée de l'expression booléenne.



# en Javascript


- La boucle « tant que » s'écrit:

```
while(condition) {  
    instructions  
}
```

# en Javascript

Une expression booléenne de test de continuation

- La boucle « tant que » s'écrit:



```
while(condition) {  
    instructions  
}
```

# en Javascript

Une expression booléenne de test de continuation

- La boucle « tant que » s'écrit :

`while(condition) {  
 instructions  
}`

```
// Algorithme mystère  
var x;  
x = 1;  
while (x < 720) {  
  Avancer(10);  
  Droite(x);  
  x=x+1;  
}
```

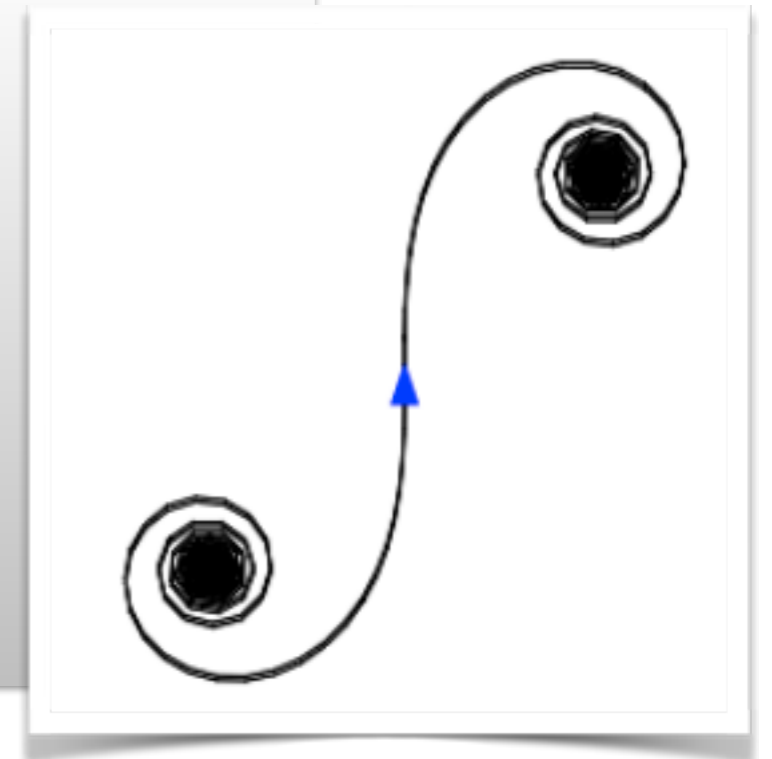
# en Javascript

Une expression booléenne de test de continuation

- La boucle « tant que » s'écrit:

```
while(condition) {  
  instructions  
}
```

```
// Algorithme mystère  
var x;  
x = 1;  
while (x < 720) {  
  Avancer(10);  
  Droite(x);  
  x=x+1;  
}
```



# Les historiques d'exécution pour les répétitives

- Dans le cas d'une répétitive, les instructions répétées se retrouvent plusieurs fois (évaluées) dans l'historique d'exécution.

Algorithme Toto

Variables:

nb1,nb2: entiers;

Début

nb1  $\leftarrow$  1;

nb2  $\leftarrow$  10;

Tant que (nb1 < nb2) faire

    nb1  $\leftarrow$  2\*nb1;

    nb2  $\leftarrow$  nb2-1;

fin tant que

Fin

# Les historiques d'exécution pour les répétitives

- Dans le cas d'une répétitive, les instructions répétées se retrouvent plusieurs fois (évaluées) dans l'historique d'exécution.

Algorithme Toto

Variables:

nb1,nb2: entiers;

Début

1 nb1 ← 1;

2 nb2 ← 10;

3 Tant que (nb1 < nb2) faire

4     nb1 ← 2\*nb1;

5     nb2 ← nb2-1;

fin tant que

Fin

# Les historiques d'exécution pour les répétitives

- Dans le cas d'une répétitive, les instructions répétées se retrouvent plusieurs fois (évaluées) dans l'historique d'exécution.

Algorithme Toto

Variables:

nb1,nb2: entiers;

Début

```
1 nb1 ← 1;
2 nb2 ← 10;
3 Tant que (nb1 < nb2) faire
4     nb1 ← 2*nb1;
5     nb2 ← nb2-1;
   fin tant que
```

Fin

Instructions	nb1	nb2
Avant 1	?	?
Après 1	1	?
Après 2	1	10

1 < 10 vrai

# Les historiques d'exécution pour les répétitives

- Dans le cas d'une répétitive, les instructions répétées se retrouvent plusieurs fois (évaluées) dans l'historique d'exécution.

Algorithme Toto

Variables:

nb1,nb2: entiers;

Début

1 nb1 ← 1;

2 nb2 ← 10;

3 Tant que (nb1<nb2) faire

4     nb1 ← 2\*nb1;

5     nb2←nb2-1;

fin tant que

Fin

Instructions	nb1	nb2	
Avant 1	?	?	
Après 1	1	?	
Après 2	1	10	1<10 vrai
Après 4	2	10	
Après 5	2	9	2<9 vrai



# Les historiques d'exécution pour les répétitives

- Dans le cas d'une répétitive, les instructions répétées se retrouvent plusieurs fois (évaluées) dans l'historique d'exécution.

Algorithme Toto

Variables:

nb1,nb2: entiers;

Début

1 nb1 ← 1;

2 nb2 ← 10;

3 Tant que (nb1<nb2) faire

4     nb1 ← 2\*nb1;

5     nb2←nb2-1;

fin tant que

Fin

Instructions	nb1	nb2	
Avant 1	?	?	
Après 1	1	?	
Après 2	1	10	1<10 vrai
Après 4	2	10	
Après 5	2	9	2<9 vrai
Après 4	4	9	
Après 5	4	8	4<8 vrai

# Les historiques d'exécution pour les répétitives

- Dans le cas d'une répétitive, les instructions répétées se retrouvent plusieurs fois (évaluées) dans l'historique d'exécution.

Algorithme Toto

Variables:

nb1,nb2: entiers;

Début

1 nb1 ← 1;

2 nb2 ← 10;

3 Tant que (nb1<nb2) faire

4     nb1 ← 2\*nb1;

5     nb2←nb2-1;

fin tant que

Fin

Instructions	nb1	nb2	
Avant 1	?	?	
Après 1	1	?	
Après 2	1	10	1<10 vrai
Après 4	2	10	
Après 5	2	9	2<9 vrai
Après 4	4	9	
Après 5	4	8	4<8 vrai
Après 4	8	8	
Après 5	8	7	8<7 faux
Après 3	8	7	

# Boucles doubles

- Les instructions à l'intérieur d'une boucle sont quelconques. Cela peut être une autre boucle (on parle de boucle double ou boucle imbriquée dans ce cas).
- Elle sont notamment utiles pour parcourir des espaces à deux dimensions (comme par exemple tous les pixels de l'écran).

# Boucles doubles

```
// Algorithme couleurs  
var x,y;  
  
for(x=0; x<800; x=x+1) {  
    for(y=0; y<600; y=y+1) {  
        Point(x,y,rgb((x+y) % 256,x%256,y%256));  
    }  
}
```

Les deux boucles sont  
l'une à l'intérieur de l'autre (on  
dit qu'elles sont imbriquées dans

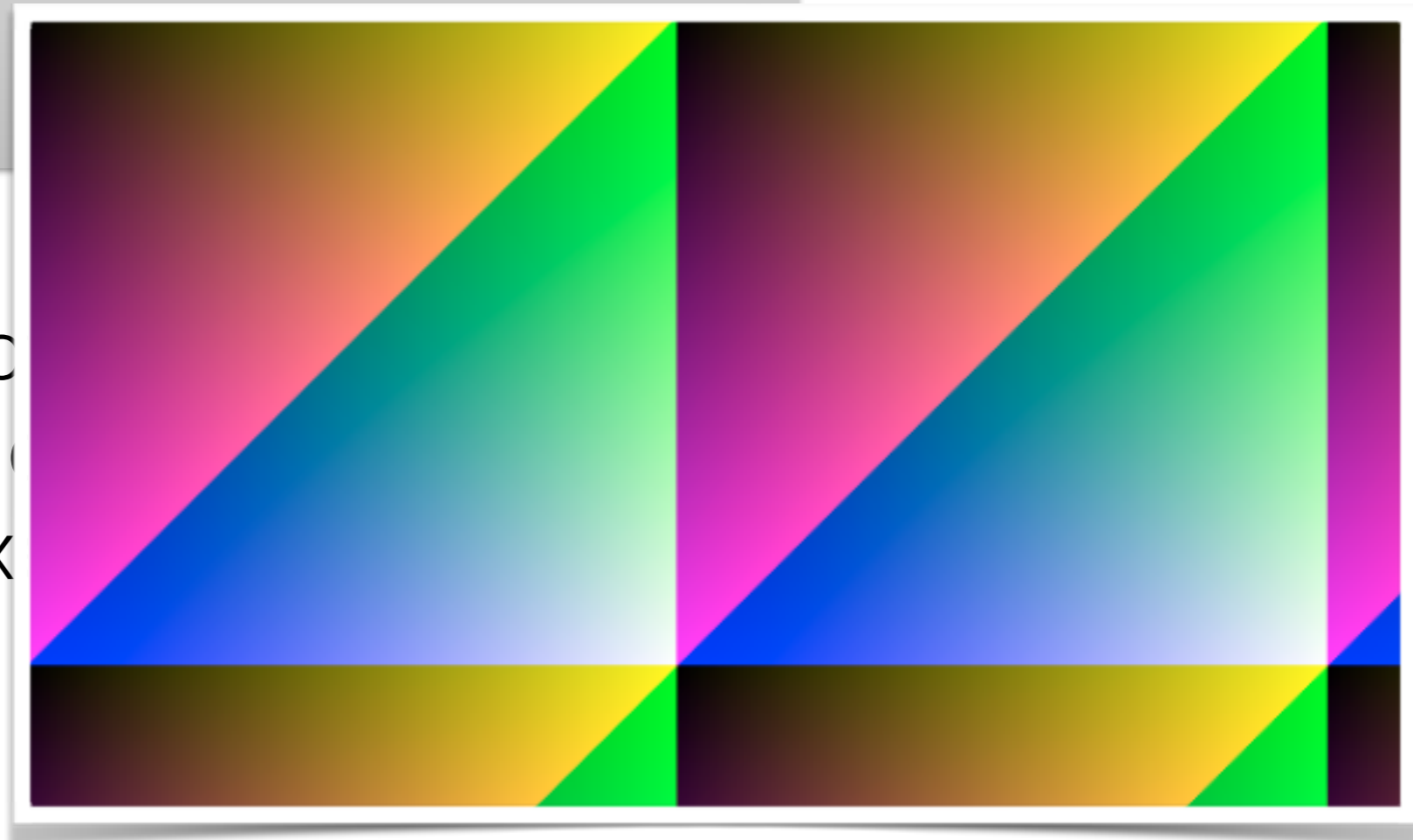
- Elles sont notamment utiles pour parcourir des espaces à deux dimensions (comme par exemple tous les pixels de l'écran).

# Boucles doubles

```
// Algorithme couleurs  
var x,y;  
  
for(x=0; x<800; x=x+1) {  
    for(y=0; y<600; y=y+1) {  
        Point(x,y,rgb((x+y) % 256,x%256,y%256));  
    }  
}
```

de la boucle sont  
de l'autre boucle (on

- Elle sont no  
espaces à  
tous les pix



IS

e

# Schémas standards

Vérification de saisie, compteurs, accumulateurs  
IL FAUT SAVOIR LES RECONNAITRE !

# Vérification de saisie

- On parle de vérification de saisie lorsqu'une boucle contient une saisie et que la valeur saisie est utilisée par le test de continuation de la boucle
- On utilise une pour « tant que » car dans ce cas, on ne connaît pas le nombre de tour de boucle à l'avance

# Vérification de saisie

- On parle de vérification de saisie lorsqu'une boucle contient une saisie et que la valeur saisie est utilisée par le test de continuation de la boucle

- On utilise dans ce cas, on ne colle pas de boucle à l'avance

Algorithme Saisie d'un nombre positif

Variables:

x: entier;

Début

x ← Saisie();

Tant que (x < 0) faire

    x ← Saisie();

fin tant que

Ecrire(x);

Fin

dans ce cas,  
on ne colle pas de boucle à l'avance



# Vérification de saisie

- On parle de vérification de saisie lorsqu'une boucle contient une saisie et que la valeur saisie est utilisée par le test de continuation de la boucle

- On utilise dans ce cas, on ne colle pas de boucle à l'avance

Algorithme Saisie d'un nombre positif

Variables:

x: entier;

Début

x ← Saisie();

Tant que (x < 0) faire

x ← Saisie();

fin tant que

Ecrire(x);

Fin

dans ce cas,  
on ne colle pas de boucle à l'avance

# Compteurs

- On parle de compteur dès lors que l'on a ajouté les instructions permettant de compter le nombre de tours de boucles (i.e., une variable et deux instructions, initialisation et incrémentation du compteur).
- On peut ajouter un compteur à tout type de boucle.
- Mais la boucle « pour » intègre naturellement un compteur.

# Compteurs

- On parle de compteur dès lors que l'on a ajouté les instructions permettant de compter le nombre de tours de boucles (i.e., une variable et deux instructions, initialisation et incrémentation du compteur).
- On peut ajouter un compteur à une boucle.
- Mais la boucle « pour » ne permet pas d'ajouter un compteur.

Algorithme Calcul de  $\lceil \log_3(100) \rceil$

Variables:

x, i: entiers;

Début

x ← 1;

i ← 0;

Tant que (x < 100) faire

    x ← 3 \* x;

    i ← i+1;

fin tant que

Ecrire(i);

Fin

# Compteurs

- On parle de compteur dès lors que l'on a ajouté les instructions permettant de compter le nombre de tours de boucles (i.e., une variable et deux instructions, initialisation et incrémentation du compteur).

- On peut ajouter un compteur à une boucle.
- Mais la boucle « pour » ne permet pas d'ajouter un compteur.

```
Algorithme Calcul de  $\lceil \log_3(100) \rceil$   
Variables:  
x, i: entiers;  
Début  
x ← 1;  
i ← 0;  
Tant que (x < 100) faire  
  x ← 3 * x;  
  i ← i + 1;  
fin tant que  
Ecrire(i);  
Fin
```

boucle.

un

# Accumulateurs

- On parle d'accumulateur dès lors que l'on a ajouté les instructions permettant de cumuler des valeurs à chaque tour de boucle (i.e., une variable et deux instructions, initialisation et modifications de l'accumulateur).
- Un accumulateur peut être additif, multiplicatif ou de concaténation.
- Il peut s'ajouter à tout type de boucle.
- Un accumulateur permet par exemple de calculer des formules de type  $\sum_{i=0}^n a_i$  ou  $\prod_{i=0}^n a_i$

# Algorithme Calcul de 20 !

Variables:

x,i: entiers;

Début

x ← 1;

pour i allant de 1 à 20 faire

    x ← x \* i;

fin pour

Ecrire(x);

Fin

l'on a ajouté les valeurs à chaque tour de boucle (instructions, simulateur).

- On parle d'accumulateur pour désigner un registre qui accumule les résultats des instructions pour un tour de boucle (initialisation et écriture).
- Un accumulateur peut être additif, multiplicatif ou de concaténation.
- Il peut s'ajouter à tout type de boucle.
- Un accumulateur permet par exemple de calculer des formules de type  $\sum_{i=0}^n a_i$  ou  $\prod_{i=0}^n a_i$

# Accumulateurs

Algorithme Calcul de 20 !

Variables:

x,i: entiers;

Début

x ← 1;

pour i allant de 1 à 20 faire

    x ← x \* i;

fin pour

Ecrire(x);

Fin

l'on a ajouté les  
s valeurs à chaque  
ux instructions,  
mult pour)

- On parle d'accumulateur pour désigner un ensemble d'instructions pour effectuer un tour de boucle. L'initialisation est faite avant le début de la boucle.

- Un accumulateur peut être additif ou multiplicatif, ou encore de concaténation.

- Il peut s'ajouter à tout type de boucle.

- Un accumulateur permet par exemple de calculer des formules de type

$$\sum_{i=0}^n a_i$$

ou

$$\prod_{i=0}^n a_i$$

Algorithme Répétition de caractères

Variables:

x: chaîne de caractères

i: entiers;

Début

x ← "";

pour i allant de 1 à 100 faire

    x ← x + '\*';

fin pour

Ecrire(x);

Fin

# Algorithme Calcul de 20 !

Algorithme Calcul de 20 !

Variables:

x,i: entiers;

Début

x ← 1;

pour i allant de 1 à 20 faire

x ← x \* i;

fin pour

Ecrire(x);

Fin

- On parle d'algorithmes pour décrire des instructions pour un ordinateur. On utilise des structures de boucles (pour, tant que) pour la répétition, l'initialisation et la fin.

l'on a ajouté les instructions pour écrire les valeurs à chaque tour de boucle. On a ajouté les instructions pour l'initialisation et la fin.

Algorithme Moyenne de 36 notes saisies

Variables:

note,i: entiers;

x: réel;

Début

x ← 0;

pour i allant de 1 à 36 faire

note ← Saisie();

x ← x + note;

fin pour

Ecrire(x/36);

Fin

Algorithme Répétition de caractères

Variables:

x: chaîne de caractères

i: entiers;

Début

x ← "";

pour i allant de 1 à 100 faire

x ← x + "\*";

fin pour

Ecrire(x);

l'addition

de boucles

par exemple

ou

$$\prod_{i=0}^{Fin} a_i$$



# Algorithme Calcul de 20 !

Variables:

x,i: entiers;

Début

x ← 1;

pour i allant de 1 à 20 faire

x ← x \* i;

fin pour

Ecrire(x);

Fin

« élément neutre » de l'opération

- On parle d'ac instructions p tour de boucl initialisation e

l'on a ajouté les s valeurs à chaque ux instructions, (calculer)

Algorithme Moyenne de 36 notes saisies

Variables:

note,i: entiers;

x: réel;

Début

x ← 0;

pour i allant de 1 à 36 faire

note ← Saisie();

x ← x + note;

fin pour

Ecrire(x/36);

Fin

e additi

e de bo

par exe

ou

$$\prod_{i=0}^{\text{Fin}} a_i$$

Algorithme Répétition de caractères

Variables:

x: chaîne de caractères

i: entiers;

Début

x ← "";

pour i allant de 1 à 100 faire

x ← x + "\*";

fin pour

Ecrire(x);

# Algorithme Calcul de 20 !

Variables:

x,i: entiers;

Début

x ← 1;

pour i allant de 1 à 20 faire

x ← x \* i;

fin pour

Ecrire(x);

Fin

« élément neutre » de l'opération

- On parle d'ac instructions p tour de boucl initialisation e

l'on a ajouté les s valeurs à chaque ux instructions, (calculer)

Algorithme Moyenne de 36 notes saisies

Variables:

note,i: entiers;

x: réel;

Début

« élément neutre » de l'opération

x ← 0;

pour i allant de 1 à 36 faire

note ← Saisie();

x ← x + note;

fin pour

Ecrire(x/36);

Fin

Algorithme Répétition de caractères

Variables:

x: chaîne de caractères

i: entiers;

Début

x ← "";

pour i allant de 1 à 100 faire

x ← x + "\*";

fin pour

Ecrire(x);

e additi

e de bo

par exe

ou

$$\prod_{i=0}^{\text{Fin}} a_i$$

# Algorithme Calcul de 20 !

Variables:

x,i: entiers;

Début

x ← 1;

pour i allant de 1 à 20 faire

x ← x \* i;

fin pour

Ecrire(x);

Fin

« élément neutre » de l'opération

- On parle d'ac instructions p tour de boucl initialisation e

l'on a ajouté les s valeurs à chaque ux instructions, (calculer)

Algorithme Moyenne de 36 notes saisies

Variables:

note,i: entiers;

x: réel;

Début

x ← 0;

pour i allant de 1 à 36 faire

note ← Saisie();

x ← x + note;

fin pour

Ecrire(x/36);

Fin

« élément neutre » de l'opération

e additi

e de bo

par exe

ou

$$\prod_{i=0}^{\text{Fin}} a_i$$

Algorithme Répétition de caractères

Variables:

x: chaîne de caractères

i: entiers;

Début

x ← ";

pour i allant de 1 à 100 faire

x ← x + '\*';

fin pour

Ecrire(x);

« élément neutre » de l'opération

# Un exemple complet

## Le jeu du plus ou moins

# Un exemple complet

## Le jeu du plus ou moins

Au départ, c'est une simple vérification de saisie...

# Un exemple complet

## Le jeu du plus ou moins

Algorithme Le jeu du plus ou moins

Variables:

cache, proposition: entiers

Début

cache ← Hasard(10); // tire un nombre au hasard entre 0 et 9

proposition ← Saisie();

Tant que (proposition ≠ cache) faire

    proposition ← Saisie();

fin tant que

Ecrire('Gagné !');

Fin

# Un exemple complet

## Le jeu du plus ou moins

On ajoute une indication à l'utilisateur, le fameux plus ou moins....

Algorithme Le jeu du plus ou moins

Variables:

cache, proposition: entiers

Début

cache ← Hasard(10); // tire un nombre au hasard entre 0 et 9

proposition ← Saisie();

Tant que (proposition ≠ cache) faire

    proposition ← Saisie();

fin tant que

Ecrire('Gagné !');

Fin

# Un exemple complet

## Le jeu du plus ou moins

Algorithme Le jeu du plus ou moins

Variables:

cache, proposition: entiers

Début

cache ← Hasard(10); // tire un nombre au hasard entre 0 et 9

proposition ← Saisie();

Tant que (proposition ≠ cache) faire

*si (proposition < cache)*

*Alors Ecrire('C est plus !');*

*Sinon Ecrire('C est moins !');*

*fin si*

proposition ← Saisie();

fin tant que

Ecrire('Gagné !');

Fin



# Un exemple complet

## Le jeu du plus ou moins

Maintenant, on va compter le nombre de propositions, ajout d'un compteur !

Algorithme Le jeu du plus ou moins

Variables:

cache, proposition: entiers

Début

cache ← Hasard(10); // tire un nombre au hasard entre 0 et 9

proposition ← Saisie();

Tant que (proposition ≠ cache) faire

*si (proposition < cache)*

*Alors Ecrire('C est plus !');*

*Sinon Ecrire('C est moins !');*

*fin si*

proposition ← Saisie();

fin tant que

Ecrire('Gagné !');

Fin

# Un exemple complet

## Le jeu du plus ou moins

Maintenant, on va compter le nombre de propositions, ajout d'un compteur !

Algorithme Le jeu du plus ou moins

Variables:

cache, proposition, nbpropositions: entiers

Début

cache ← Hasard(10); // tire un nombre au hasard entre 0 et 9

proposition ← Saisie();

*nbpropositions* ← 1;

Tant que (proposition ≠ cache) faire

*nbpropositions* ← *nbpropositions* + 1;

si (proposition < cache)

Alors Ecrire('C est plus !');

Sinon Ecrire('C est moins !');

fin si

proposition ← Saisie();

fin tant que

Ecrire('Gagné ! *Vous avez mis* ' + *nbpropositions* + ' coups !');

Fin

# Un exemple complet

## Le jeu du plus ou moins

Maintenant, on va compter le nombre de propositions, ajout d'un compteur !

Algorithme Le jeu du plus ou moins

Variables:

cache, proposition, nbpropositions: entiers

Début

cache ← Hasard(10); // tire un nombre au hasard entre 0 et 9

proposition ← Saisie();

*nbpropositions* ← 1;

Tant que **Moralité:** on a combiné une vérification de saisie et un compteur sur une boucle tant que

On a ajouté en plus une conditionnelle.

Enfin, on a créé notre jeu complet !

proposition ← Saisie(),

fin tant que

Ecrire('Gagné ! *Vous avez mis* ' + *nbpropositions* + ' coups !');

Fin

# En javascript

Algorithme Le jeu du plus ou moins

Variables:

cache, proposition, nbpropositions: entiers

Début

cache ← Hasard(10); // tire un nombre au hasard entre 0 et 9

proposition ← Saisie();

nbpropositions ← 1;

Tant que (proposition ≠ cache) faire

    nbpropositions ← nbpropositions + 1;

    si (proposition < cache)

        Alors Ecrire('C est plus !');

        Sinon Ecrire('C est moins !');

    fin si

    proposition ← Saisie();

fin tant que

Ecrire('Gagné ! Vous avez mis '+ nbpropositions+ ' coups !');

Fin

```
// Algorithme le jeu du plus ou moins

var proposition, cache, nbpropositions;

cache = Hasard(10);
proposition = Saisie();
nbpropositions = 1;
while (proposition != cache) {
    nbpropositions = nbpropositions + 1;
    if (proposition < cache) {
        Ecrire('C\'est plus !');
    } else {
        Ecrire('C\'est moins !');
    }
    proposition = Saisie();
}
Ecrire('Gagné ! Vous avez mis '+nbpropositions
+' coups');
```

# Bilan

- Un algorithme est une suite séquentielle d'instructions qui modifient l'état de la mémoire.
- La syntaxe est enrichie de structures de contrôle: conditionnelles et répétitives pour simplifier/ optimiser l'écriture de l'algorithme.
- Des schémas standards qui s'appliquent et se combinent dans énormément de cas de résolution de problèmes.