Faculté des Sciences
كلية العلوم
Université Ibn Tofaïl
جامعة ابن طفيل

Département d'Informatique
Filière MIP - Semestre 2
2024-2025

Module : Informatique II (Algorithmique II / Python)

Chapitre 2

Fonctions et Procédures

Pr. Khadija Louzaoui

- Pour l'instant, un algorithme est une séquence d'instructions mais sans **partage** des **parties importantes** ou **utilisées plusieurs fois**.
- **Bonne pratique** : Ne jamais dupliquer un bloc de code !
- Ce qu'on veut **recopier** doit être mis dans une **fonction** ou **procédure**.

Exercice

Ecrire un algorithme qui permet de calculer la combinaisons C_n^p

$$C_n^p = \frac{n!}{p!\,(n-p)!}$$

Exercice

```
Algorithme Combinaisons
Variables  n,p,r1,r2,r3,i: entier
Début
    Ecrire("Donner les valeurs de n et p (p<n):")
    Lire(n,p)
     r1 ← 1
    Pour i de 2 à n pas 1 faire
    r1 ← r1*i
    Finpour
     r2 ← 1
    Pour i de 2 à p pas 1 faire
    r2 ← r2*i
    Finpour
    r3 ← 1
    Pour i de 2 à n-p pas 1 faire
    r3 ← r3*i
    Finpour
    Ecrire("Résultat = ", r1 div (r2 * r3))
fin
```

- Le traitement "Calcul de la factorielle d'un nombre est répété 3 fois.

```
r1 ← 1
Pour i de 2 à n pas 1 faire
r1 ← r1*i
Finpour
```

- L'idée est ***d'écrire*** un "**sous-algorithme**" (*Factoriel*) spécialisé dans le calcul de la factorielle d'un nombre.

- Puis ***appeler*** se sous algorithme chaque fois que c'est nécessaire dans "**l'algorithme principal**" (Algorithme calculant C_n^p)

❑ La **fiabilité**, la **lisibilité** et la **réutilisabilité** des algorithmes (programmes), reposent sur l'utilisation des sous-algorithme (sous-programmes). Ces derniers permettent :

- **La réduction de la taille des** algorithmes (programmes) : il est possible de déterminer les blocs analogues, les substituer par un sous-algorithme (sous-programme), ensuite l'appeler dans des points déterminés au niveau de l'algorithme (programme) principal.
- **L'organisation du code** : le problème initial peut être découpé en sous-problèmes (modules) où chacun sera résolu par un sous-algorithme (sous-programme).

Pour améliorer la réutilisabilité de l'algorithme la solution est **d'encapsuler** le code à **répéter** au sein d'un **sous-algorithme**.

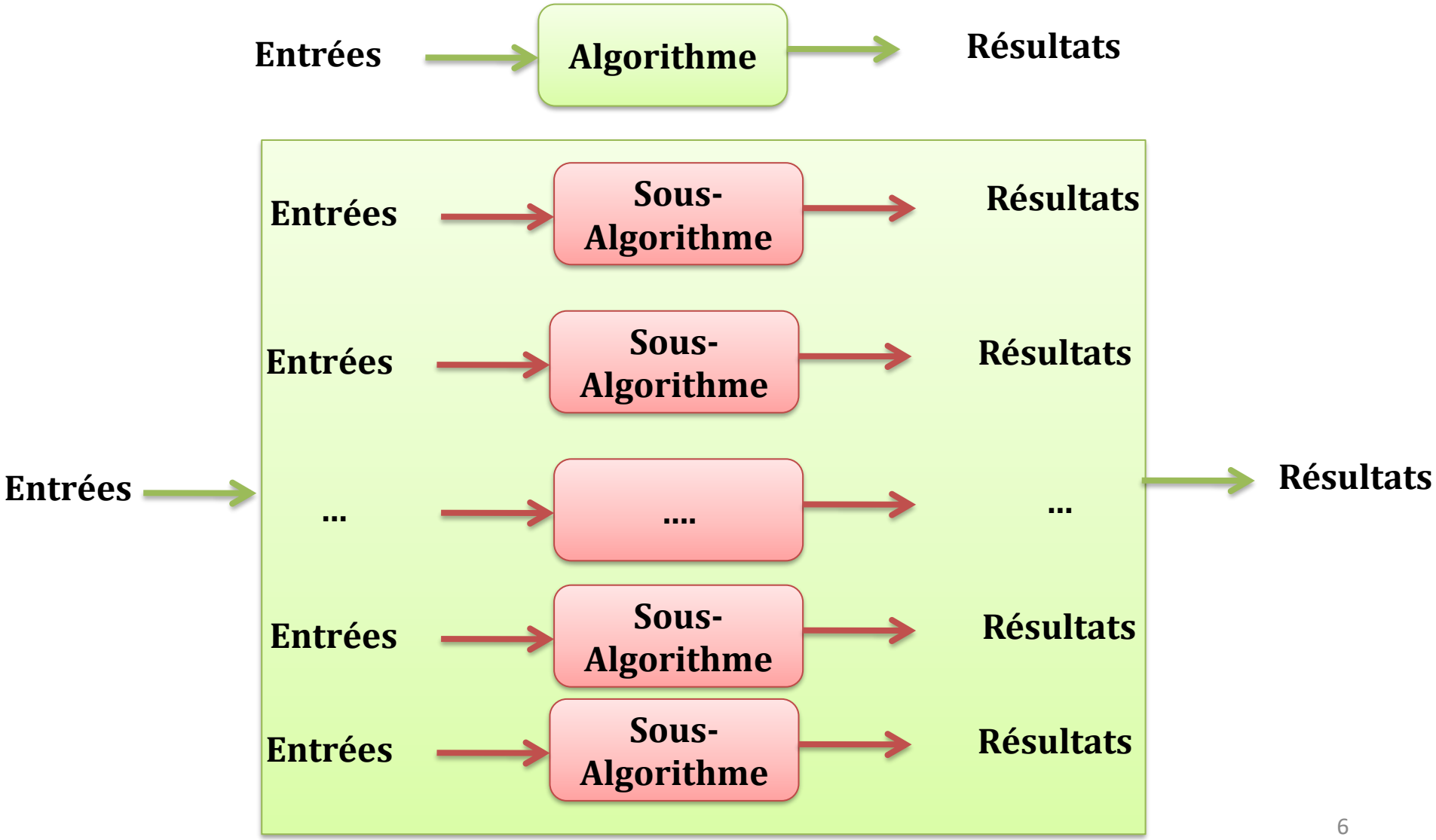
Sous-Algorithme

Pour résoudre un problème:

- L'analyse d'un problème (démarche descendante) consiste à découper un problème complexes en tâches (sous-problème) que l'on est capable de définir.
- Un algorithme **long** est souvent difficile à écrire et à comprendre.
- C'est pourquoi, il est préférable de le décomposer en des parties appelées sous-algorithme ou modules.
- Chacun de ces **sous-algorithmes** devient un **nouveau problème** à résoudre.
- Si on considère que l'on sait résoudre ces **sous-problèmes**, alors on sait "quasiment" résoudre **le problème initial**.

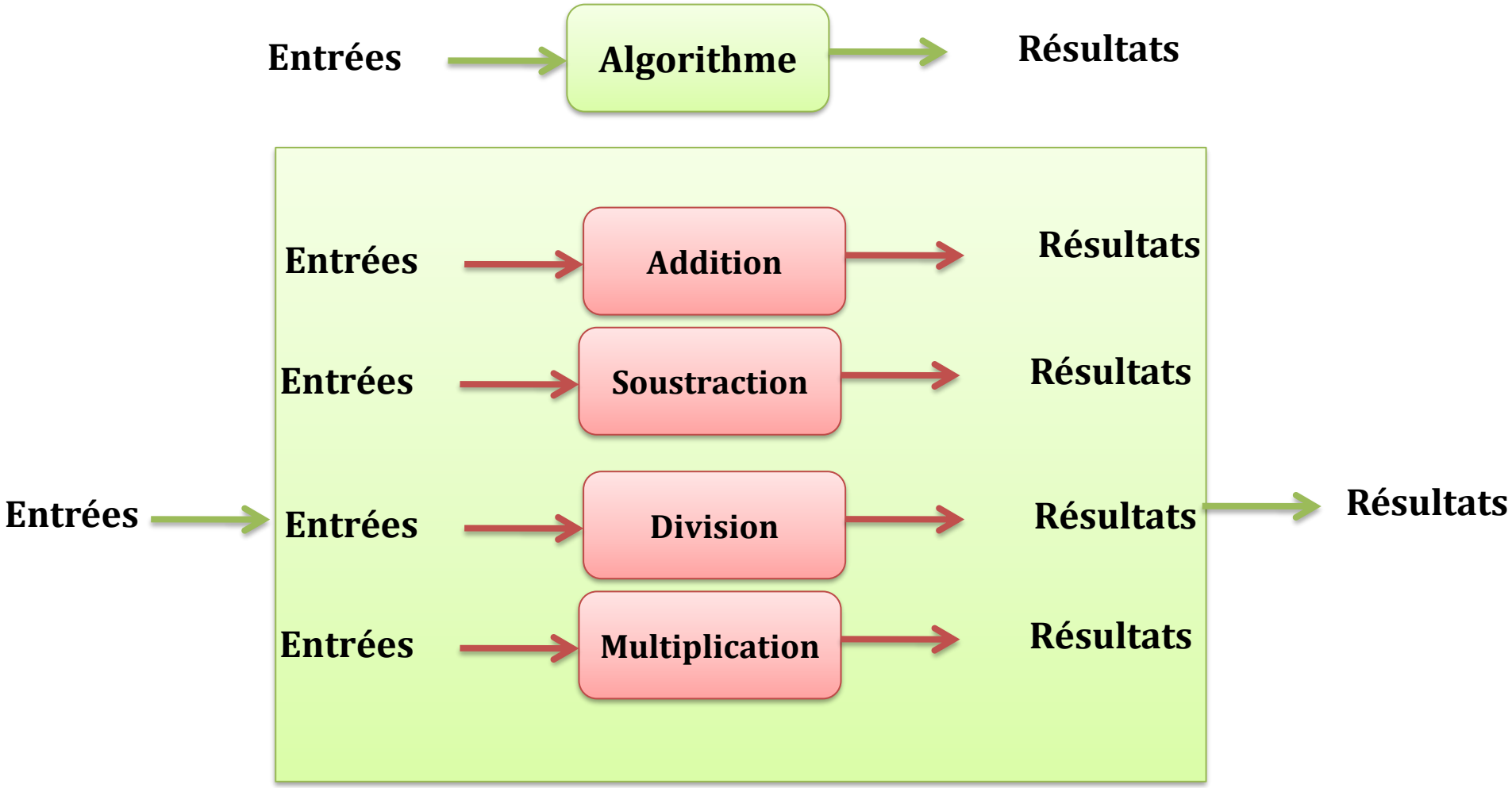
Sous-Algorithme

Pour résoudre un problème:



Sous-Algorithme

Exemple: Calculatrice



Sous-Algorithme

❑ Un **sous-algorithme** est une portion de code **analogue** à un algorithme, destiné à réaliser une certaine tâche à l'intérieur d'un autre algorithme.

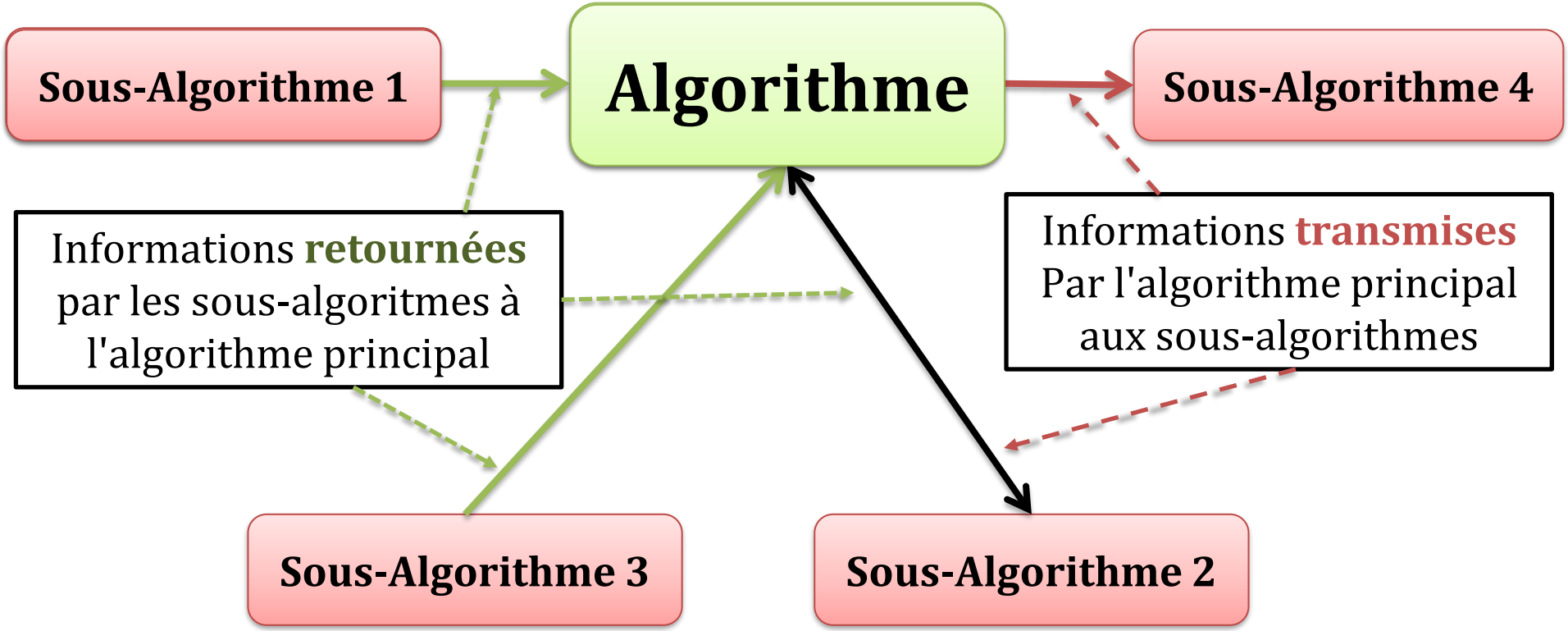
❑ Il est identifié par un nom unique et un bloc d'instructions qui peut être exécuté plusieurs fois par des **appels**.

❑ Un **appel** est une instruction qui fait partie d'un autre algorithme ou sous-algorithme appelé le (**sous- algorithme appelant**).

❑ Remarque : un sous-algorithme peut en appeler un autre.

Sous-Algorithme

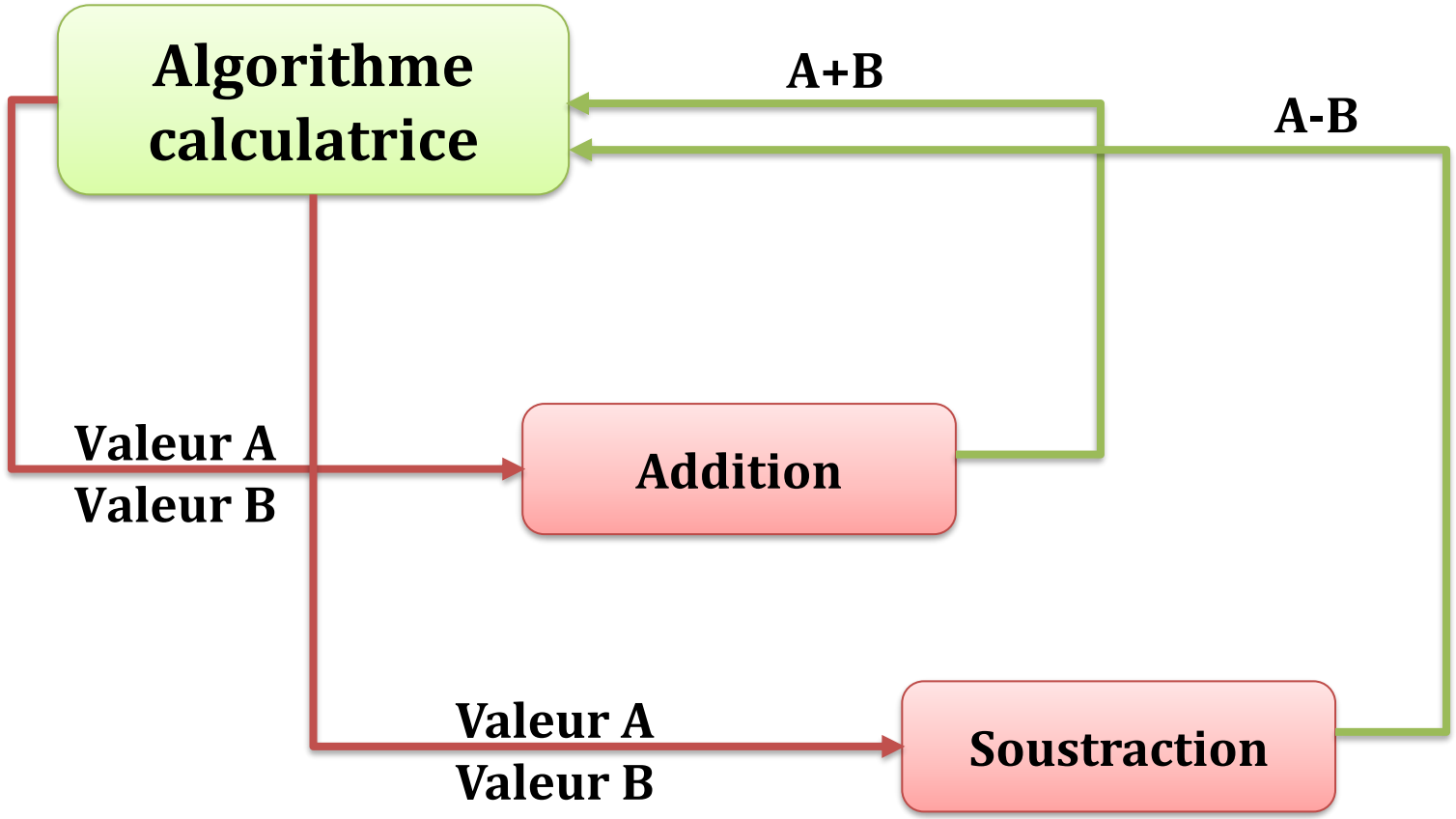
Comment les informations circulent ?



Sous-Algorithme

Comment les informations circulent ?

Exemple:



Sous-Algorithme

❑ un **sous-algorithme** est **utilisé** pour deux raisons essentielles :

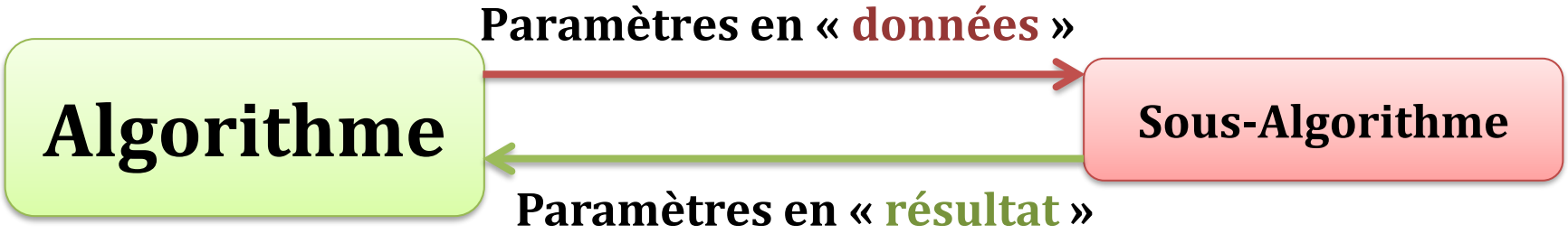
- Lorsqu'une tâche est **répétée** plusieurs fois : on écrit un sous-algorithme pour cette tâche et l'on appelle à chaque endroit où l'on en a besoin c.à.d. on évite de réécrire le même code à plusieurs endroits dans le même algorithme.

- **Pour réaliser la structuration d'un problème en sous-problèmes** : on divise le problème en sous-problèmes pour mieux le contrôler (diviser pour régner).

Sous-Algorithme

Communication des informations

Informations **transmises** Par l'algorithme principal aux sous-algorithmes



Informations **retournées** par les sous-algoritmes à l'algorithme principal

Fonctions et Procédures

- **Ecrire un algorithme qui résout un problème revient toujours à écrire des sous-algorithmes qui résolvent des sous parties du problème initial.**

- En algorithmique, il existe deux types de sous-algorithmes :
  - **Les fonctions**
  - **Les procédures**

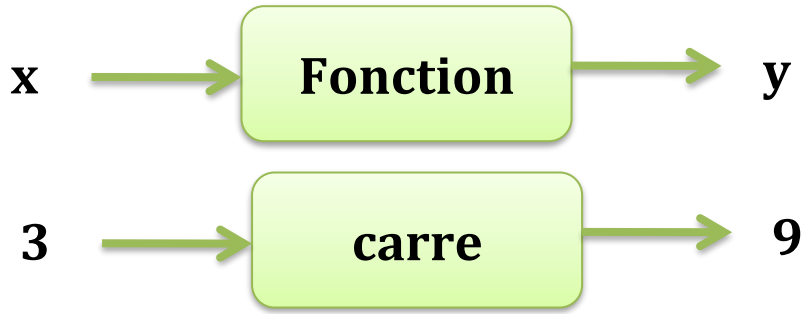
Fonctions et Procédures

- Les fonctions et les procédures sont des **modules** (groupe d'instructions) **indépendants désignés** par un **nom**.
- Elles ont plusieurs **avantages** :
  - Permettent d'éviter de **réécrire** un **même traitement** plusieurs fois (**factoriser**). En effet, on fait **appel** à la procédure ou à la fonction aux **endroits spécifiés**.
- Permettent d'**organiser le code** et améliorent la **lisibilité** des algorithmes.
- **Facilitent la maintenance** du code (il suffit de modifier une seule fois).
- Ces procédures et fonctions peuvent éventuellement être **réutilisées** dans d'autres algorithme.

Définition de la fonction

- Une **fonction** est une suite d'instructions regroupées sous un **nom** , elle prend en entrée des **paramètres** et **retourne** un **résultat**.
- Une fonction est **écrite séparément** du corps de l'algorithme principal et sera **appelé** par celui-ci lorsque cela sera nécessaire.
- Le rôle d'une fonction en algorithmique est similaire à celui d'une **fonction en mathématique** : elle **retourne un résultat** au algorithme appelant.

$$f(x) = y$$



Utilisation des fonctions

- Le **corps** de la **fonction** est la portion de l'algorithme à **réutiliser** ou à mettre en évidence.
- Les **paramètres** de la fonction sont un ensemble de variables locales (**paramètres formels**) **associées** à un ensemble de variables ou constantes de la fonction appelant (**paramètres effectifs**).
- Pour utiliser une fonction dans l'algorithme il faut :

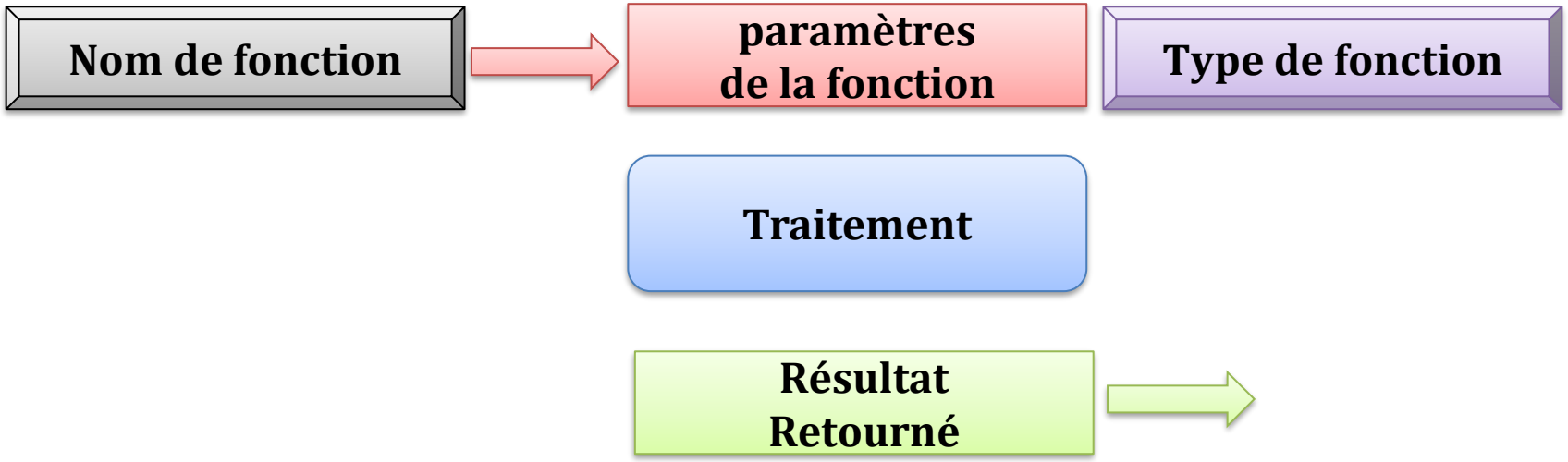
Déclarer La fonction

1

Appeler la fonction

2

Déclaration de la fonction



Déclaration de la fonction

Une fonction s'écrit en dehors de l'algorithme principal sous la forme:

```
Fonction  NomFonction (liste des paramètres :type):  type_fonction
Variables
                Liste des variables locales : type
Début
      Instructions
      Retourner Résulat
Fin
```

Corps de la fonction

Le **nom_fonction** : désignant le nom de la fonction. Pour le choix d'un nom de fonction il faut respecter les mêmes règles que celles pour les noms de variables.

type_fonction est le type du résultat retourné.

L'instruction **retourner** sert à retourner la valeur du résultat trouvé par la fonction.

Déclaration de la fonction

Exemple : Fonction carre

```
Fonction carre ( nb : entier) : entier
Variables
                    p : entier
Début
    p ← nb^2
    Retourner p
Fin
```

Appel d'une fonction

L'appel d'une fonction se fera par simple écriture de son nom dans l'algorithme principal.
Le résultat étant une valeur, devra être **affecté** ou être **utilisé dans une expression**, une **écriture**, ..

Affectation:

```
Nom_var ← NomFonction ( arg1, arg2,....)
```

Expression :

```
Nom_var1 ← Nom_var2 /NomFonction (arg1, arg2,....)
```

Ecriture :

```
Ecrire ( NomFonction (arg1, arg2,....) )
```

Appel d'une fonction

Exemple : Fonction carre

```
Algorithme exemple_fonction
Variables  x : entier
   Fonction  carre ( nb: entier) :  entier
    Variables
                    p : entier
    Début
          p ← nb^2
          Retourner p
   Fin
Début
     Ecrire("Donner un nombre:")
     Lire(x)
     Ecrire("Le carré de ", x , " est: ", carre(x) )
fin
```

Déclaration de la fonction

L'appel de la fonction

Appel d'une fonction

Exemple : Fonction carre

```
Algorithme exemple_fonction
Variables  x: entier
Fonction carre ( nb: entier) :  entier
     Variables
                    p : entier
     Début
          p ← nb^2
          Retourner p
    Fin
Début
    Ecrire("Donner un nombre":)
    Lire(x)
    Ecrire("Le carré de ", x , "est:", carre(x) )
fin
```

Variables globales

Variables Locales

Variables locales et globales

- On peut manipuler deux types de variables dans un sous-algorithme : **des variables locales** et **des variables globales**. Elles se distinguent par ce qu'on appelle leur **portée** (leur "champ de définition", leur "durée de vie").

- Une **variable locale** n'est connue qu'à l'intérieur du module où elle a été définie. Elle est créée à l'appel du module et détruite à la fin de son exécution.

- Une **variable globale** est connue par l'ensemble des modules et l'algorithme principal. Elle est définie durant toute l'algorithme et peut être utilisée et modifiée par les différents modules de l'algorithme.

Variables locales et globales

- La manière de distinguer la déclaration des variables **locales** et **globales** diffère selon le langage.
  - En général, les variables déclarées à l'**intérieur** d'un sous-programme sont considérées comme variables **locales**.
- En pseudo-code, on va adopter cette règle pour les variables locales et on déclarera les variables globales dans l'algorithme principal.
- Conseil : Il faut utiliser autant que possible des variables locales plutôt que des variables globales. Ceci permet **d'économiser la mémoire** et d'assurer **l'indépendance des sous-programmes**. (à la sortie des modules, les variables locales sont détruites et leur valeur perdues).

Appel d'une fonction

Exemple : Fonction carre

```
Algorithme exemple_fonction
Variables  x: entier
Fonction carre ( nb: entier) :  entier
     Variables
                    p : entier
     Début
          p ← nb^2
          Retourner p
    Fin
Début
    Ecrire("Donner un nombre":)
    Lire(x)
    Ecrire("Le carré de ", x , "est:", carre(x) )
fin
```

Paramètres fictifs

Paramètres effectifs (arguments)

Paramètres et Arguments

- Les paramètres (**formels** ou **fictifs**) d'un sous-algorithme sont simplement des variables locales qui sont initialisées par les valeurs obtenues lors de l'appel (paramètres **effectifs** ou **arguments**)
- Au moment de l'appel du sous-algorithme , les valeurs des **arguments** sont affectés aux paramètres **formels** .
- Le **nombre** de paramètres doit **correspondre** au **nombre** d'arguments.
- Le **type** du $k^{ième}$ argument doit **correspondre** au **type** du $k^{ième}$ paramètre.
- L'appel d'un sous-algorithme avec des **arguments** (paramètres effectifs) provoque l'exécution complète d'un sous-algorithme.

Evaluation d'un appel de fonction

L'évaluation de **l'appel** : $f(arg_1, arg_2, \dots, arg_n)$ d'une fonction définie par :

$$Fonction\ f(x_1 : type1, x_2 : type2, \dots, x_n : type\ n)\{\dots\}$$

s'effectue de la façon suivante :

1. Les expressions $\mathbf{arg_1, arg_2, \dots, arg_n}$ passées en argument sont évaluées.
2. les valeurs correspondantes sont affectées aux paramètres $\mathbf{x_1, x_2, \dots, x_n}$ de la fonction f (variables locales à f).

   Concrètement, ces deux premières étapes reviennent à faire :

   $$\mathbf{x_1 \leftarrow arg_1;\ x_2 \leftarrow arg_2;\ \dots; x_n \leftarrow arg_n}$$

3. les instructions correspondantes au corps de la fonction f sont exécutées.
4. l'expression suivant la première commande **return** rencontrée est évaluée...
5. ...et **retournée** comme **résultat** de l'appel :

   cette valeur remplace l'expression de l'appel, c-à-d l'expression

   $$f(arg_1, arg_2, \dots, arg_n)$$

Evaluation d'un appel de fonction

```
Algorithme exemple_fonction
Déclaration de la fonction carre
    Variables p, x: entier
        ……..
        p← carre(x)
fin
```

nb: Paramètre formel

x : Argument

```
Fonction carre ( nb: entier) : entier
    Variables
        p : entier
    Début
        p ← nb^2
        Retourner p
    Fin
```

Lors de l'exécution de la fonction carre( ), il y a une association entre le paramètre effectif (argument) **x** et le paramètre formel **nb** d'où la valeur de **x** est copiée dans **nb**. Ce type d'association s'appelle passage de paramètre par valeur.

Evaluation d'un appel de fonction

Exemple : Fonction carre

x=5

```
Algorithme exemple_fonction
Variables  x: entier
    Fonction  carre ( nb: entier) :  entier
     Variables
                        p : entier
     Début
          p ← nb^2
          Retourner p
    Fin
Début
    Ecrire("Donner un nombre:")
    Lire(x)
    Ecrire("Le carre de ", x , "est:", carre(x) )
fin
```

```
Début
    x ← 5
    Ecrire("Le carre de ", 5 ,
    "est:", carre(5) )
fin
```

```
Fonction  carre ( 5) :  entier
Variables
    p : entier
Début
     p ← 5^2
     Retourner 25
    Fin
```

```
Début
    Ecrire("Le carre de ", 5 ,
    "est:", 25)
fin
```

Exemple

Ecrire un algorithme qui permet de définir et d'appeler une fonction **Minimum** qui renvoie le plus petit de deux nombre différents

Exemple

Ecrire un algorithme qui permet de définir et d'appeler une fonction **Minimum** qui renvoie le plus petit de deux nombre différents

```
Algorithme Afficher_Minimum
Variables  a,b,min: entier
Fonction  Minimum ( n,m: entier) :  entier
      Variables
                        min : entier
      Début
            min <- n
            Si n>m  alors
                  min <- m
            FinSi
            Retourner min
     Fin
Début
     Ecrire("Donner deux nombres:")
     Lire(a,b)
     min <- Minimum(a,b)
     Ecrire("Le minimum de ", a , " et  ", b, " est : ",min)
fin
```

Procédure

- Dans le cas où **une tâche se répète** dans **plusieurs endroits** du **programme** et elle ne calcule pas de résultats ou qu'elle calcule plusieurs résultats à la fois alors on utilise une **procédure** au lieu d'une fonction.

- Une procédure est un sous-algorithme semblable à une fonction mais **qui ne retourne rien**.

Qu'est ce qu'une procédure?

- Une procédure est **une suite d'instructions** regroupées sous un nom , elle prend en entrée des **paramètres** mais qui **ne retourne rien**.
- Une procédure est **écrite séparément** du corps de l'algorithme principal et sera **appelé** par celui-ci lorsque cela sera nécessaire.
- Pour utilise une procédure dans l'algorithme il faut :

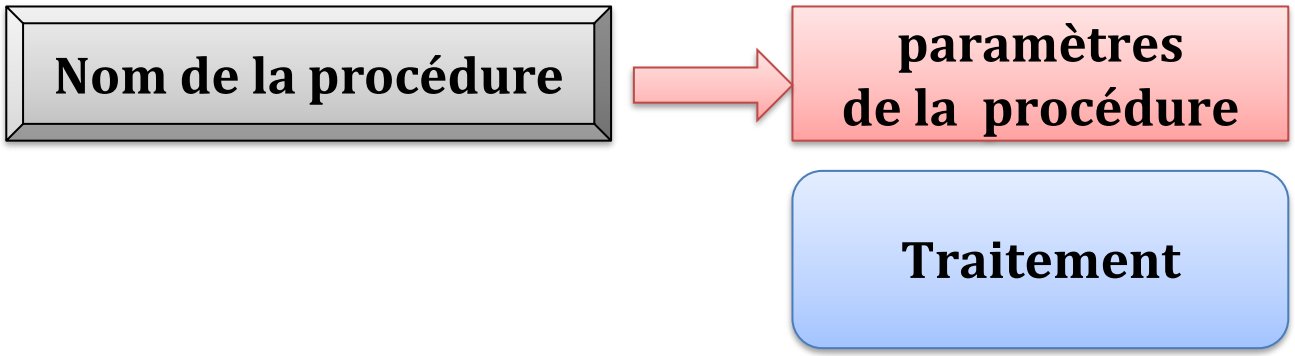
Déclarer La procédure

1

Appeler la procédure

2

Définition de la procédure



Définition de la procédure

Une procédure s'écrit en dehors de l'algorithme principal sous la forme:

Nom de la procédure

paramètres de la procédure

```
Procédure  NomProcédure ( liste des paramètres :type)
Variables
                Liste des variables locales : type
Début
      Instructions
Fin
```

Corps de la procédure

Déclaration de la procédure

Exemple : Procédure carre

```
Procédure  carre ( nb : entier)
Variables
                    p : entier
Début
    p ← nb^2
    Ecrire(" le carre du nombre" , nb, " est :" ,p)
Fin
```

Appel d'une procédure

- Pour appeler une procédure dans un algorithme principal ou dans une autre procédure, il suffit d'écrire une instruction indiquant le nom de la procédure :

Méthode:

```
NomProcédure ( arg1, arg2,....)
```

- contrairement à l'appel d'une fonction, on **ne peut pas affecter** la procédure appelée ou l'utiliser dans une expression.
- L'appel d'une procédure est une instruction **autonome**.

Appel d'une procédure

Exemple : Procédure carre

```
Algorithme exemple_procédure
    Variables  x: entier
    Procédure  carre ( nb: entier)
     Variables
                    p : entier
     Début
         p ← nb^2
         Ecrire(" le carre du nombre" , nb, " est :" ,p)
    Fin
Début
    Ecrire("Donner un nombre":)
    Lire(x)
    carre(x)
fin
```

Déclaration de La procédure

L'appel de la procédure

Appel d'une procédure

Exemple : Procédure carre

```
Algorithme exemple_procédure
    Variables  x: entier
    Procédure  carre ( nb: entier)
     Variables
                    p : entier
     Début
         p ← nb^2
         Ecrire(" le carre du nombre" , nb, " est :" ,p)
    Fin
Début
    Ecrire("Donner un nombre":)
    Lire(x)
    carre(x)
fin
```

Paramètre formel

Paramètre effectif (argument)

Appel d'une procédure

Exemple : procédure carre

x=5

```
Algorithme exemple_procédure
Variables  x: entier
     Procédure  carre ( nb: entier)
     Variables
                         p : entier
      Début
            p ← nb^2
      Ecrire("Le carre de ", nb , "est:", p)

     Fin
Début
     Ecrire("Donner un nombre":)
     Lire(x)
     carre(x)
fin
```

```
Début
     carre(5)
fin
```

```
Procédure  carre ( 5)
Variables
     p : entier
Début
      p ← 5^2
      Ecrire("Le carre de ", 5 ,
           "est:", 25)
     Fin
```

Le carre de 5 est: 25

Le carre de 5 est : 25

Exemple

Ecrire un algorithme qui permet de définir et d'appeler une procédure **Minimum** qui affiche le plus petit de deux nombre différents

```
Algorithme exemple_procédure
Variables  a,b: entier
Procédure  Minimum ( n,m: entier)
        Variables
                        min : entier
        Début
                min ← n
                Si n>m  alors
                     min ← m
                Fin si
                Ecrire("Le minimum de ", n , "et:", m, "est:",min)
        Fin
Début
     Ecrire("Donner deux nombres":)
     Lire(a,b)
     Minimum(a,b)
fin
```

Exercice

Supposons qu'on veut écrire un sous-algorithme MoySom qui prend en paramètres trois entiers a, b et c, et qui calcul leur somme et renvoie leur moyenne.

Quelle est la déclaration correspondante ?

- **Fonction** MoySom (a : **entier**, b: **entier**, c: **entier**) : **entier,réel**
- **Fonction** MoySom (a : **entier**, b: **entier**, c: **entier**) : **réel**
- **Procédure** MoySom (a : **entier**, b: **entier**, c: **entier**, som: **entier**, moy: **réel**)
- **Procédure** MoySom (a : **entier**, b: **entier**, c: **entier**)

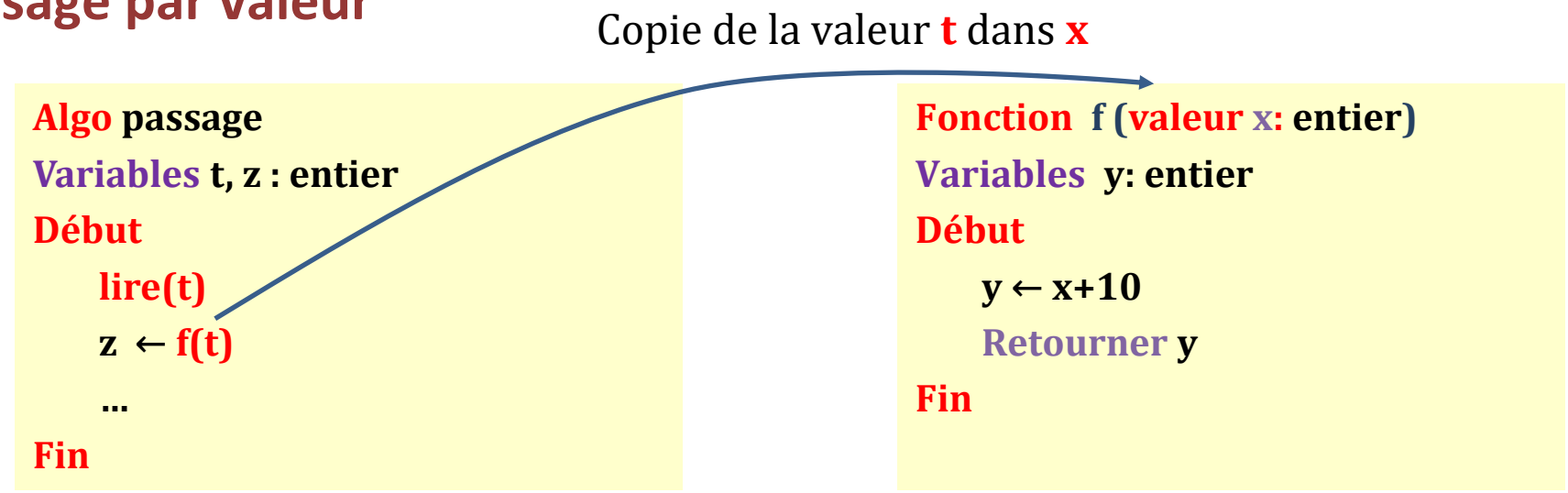
- Deux modes de transmission de données :
  - Passage par **valeur**
  - Passage par **adresse** (référence)

Passage par valeur

- On appelle **passage par valeur** quand seule la **valeur** d'un paramètre **effectif** est **connue** de la fonction
  - **Les valeurs transmises** sont **recopiées** dans la **fonction appelée**, on **travaille sur cette copie**.

- Principe :
  - la **fonction appelante** fait une **copie** de la valeur passée en argument,
  - **passe cette copie** à la **fonction appelée** à l'intérieur d'une variable **créée** dans l'espace mémoire
  - cette variable est accessible de manière **interne** par la fonction à partir du **paramètre formel** correspondant.

Passage par valeur



- Il y a **recopie** de la valeur de **l'argument** dans une **variable locale** (paramètre formel) à la fonction :
  - Toutes les modifications effectuées sur **le paramètre formel n'affectent que cette valeur locale** et **ne sont pas visibles dans l'algorithme appelant**.
- La fonction ne travaille que sur **une copie** qui va être supprimée à la fin de la fonction.

Passage par valeur

Exemple : procédure incrémenter

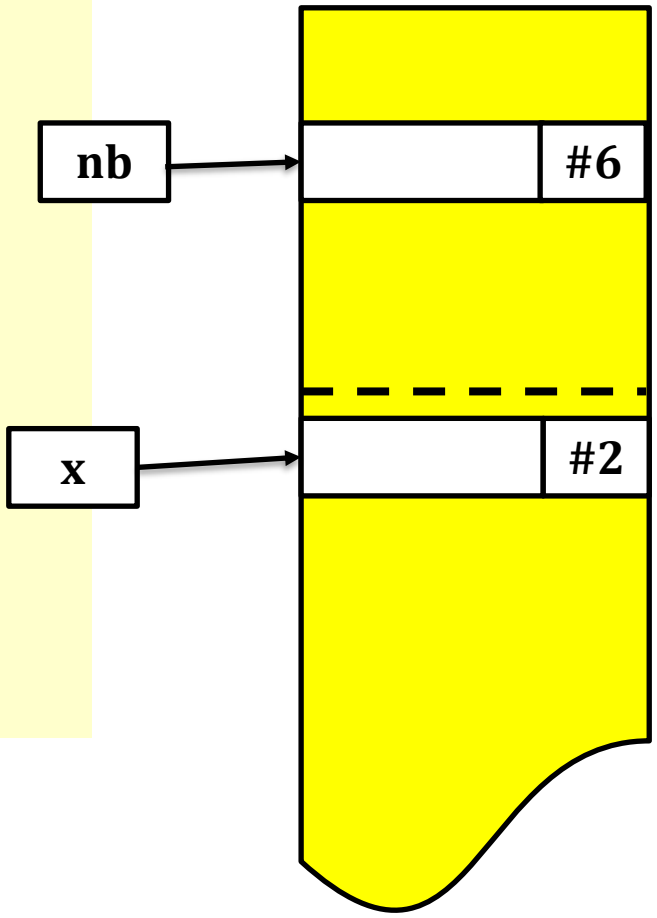
```
Algorithme test_incrementer
Variables  x: entier
     Procédure  incrementer (valeur nb: entier)
      Début
            nb ← nb+1
      Fin
Début
     Ecrire("Donner un nombre:")
     Lire(x)
     Ecrire ("la valeur avant l' appel est :", x)
     incrementer(x)
     Ecrire ("la valeur après l' appel est :", x)
fin
```

```
Donner un nombre :
5
la valeur avant l' appel est : 5
la valeur avant l' appel est : 5
```

Passage par valeur

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
      Procédure  incrementer (valeur nb: entier)
       Début
            nb ← nb+1
       Fin
Début
      Ecrire("Donner un nombre":)
      Lire(x)
      Ecrire ("la valeur avant l' appel est :", x)
      incrementer(x)
      Ecrire ("la valeur après l' appel est :", x)
fin
```

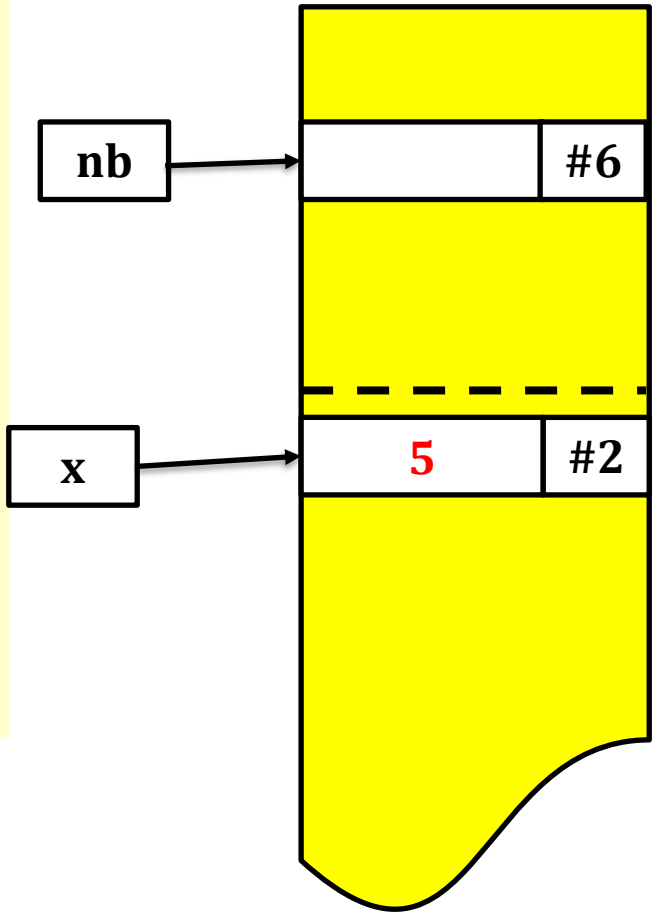


Passage par valeur

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
      Procédure  incrementer (valeur nb: entier)
       Début
            nb ← nb+1
       Fin
Début
      Ecrire("Donner un nombre":)
      Lire(x)
      Ecrire ("la valeur avant l' appel est :", x)
      incrementer(x)
      Ecrire ("la valeur après l' appel est :", x)
fin
```

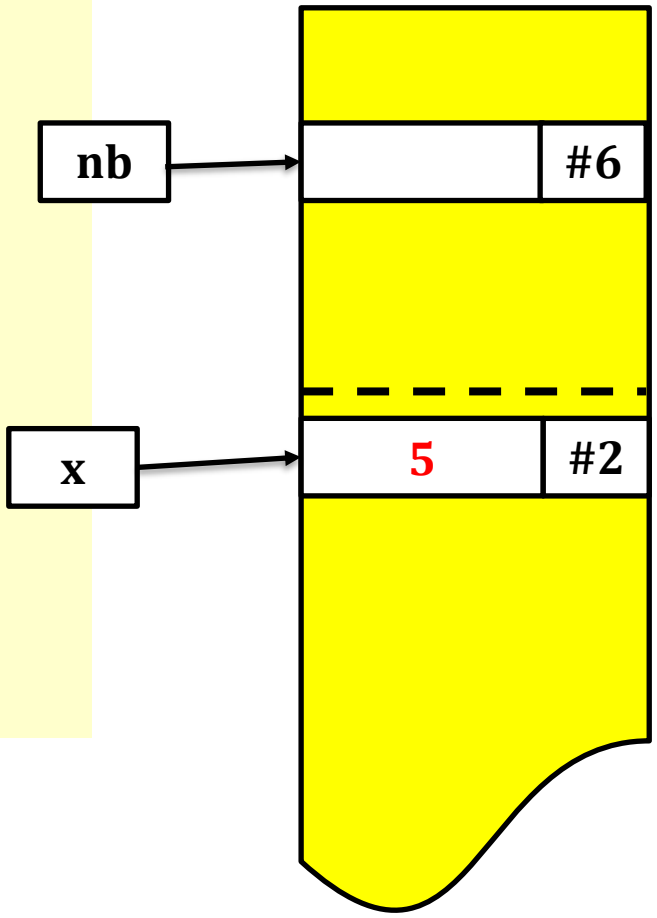
```
Donner un nombre :
5
```



Passage par valeur

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
      Procédure  incrementer (valeur nb: entier)
       Début
            nb ← nb+1
       Fin
Début
      Ecrire("Donner un nombre":)
      Lire(x)
      Ecrire ("la valeur avant l' appel est :", x)
      incrementer(x)
      Ecrire ("la valeur après l' appel est :", x)
fin
```

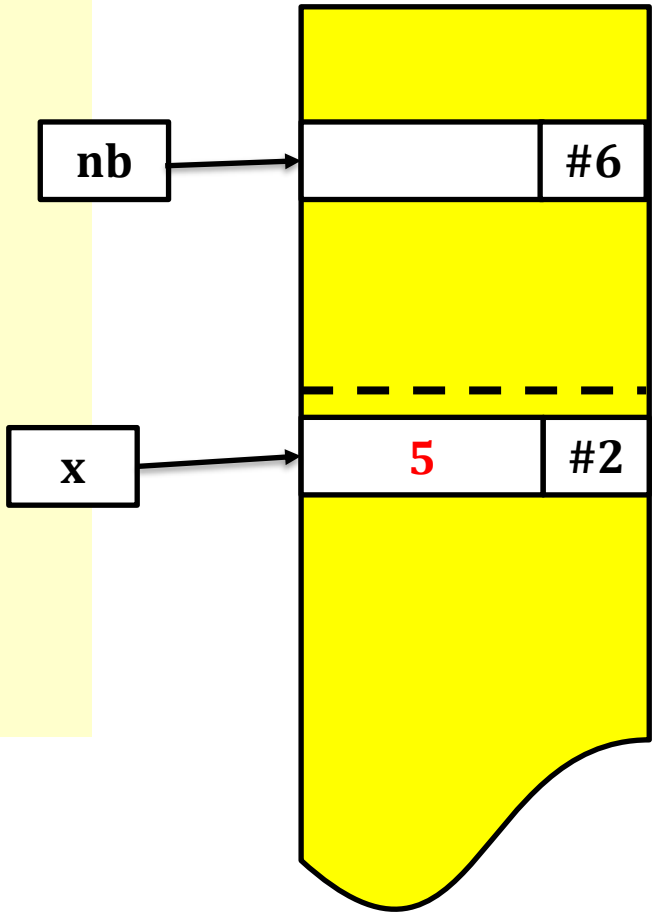



```
Donner un nombre :
5
la valeur avant l' appel est : 5
```

Passage par valeur

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
     Procédure  incrementer (valeur nb: entier)
      Début
            nb ← nb+1
      Fin
Début
     Ecrire("Donner un nombre":)
     Lire(x)
     Ecrire ("la valeur avant l' appel est :", x)
     incrementer(x)
     Ecrire ("la valeur après l' appel est :", x)
fin
```

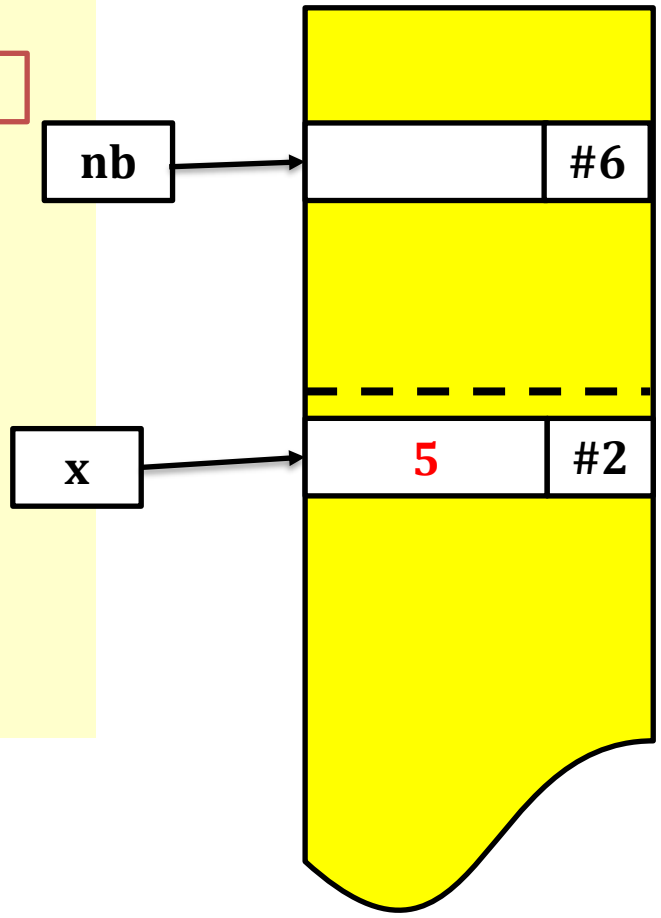



```
Donner un nombre :
5
la valeur avant l' appel est : 5
```

Passage par valeur

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
    Procédure  incrementer (valeur nb: entier)
      Début
            nb ← nb+1
      Fin
Début
     Ecrire("Donner un nombre":)
     Lire(x)
     Ecrire ("la valeur avant l' appel est :", x)
     incrementer(x)
     Ecrire ("la valeur après l' appel est :", x)
fin
```

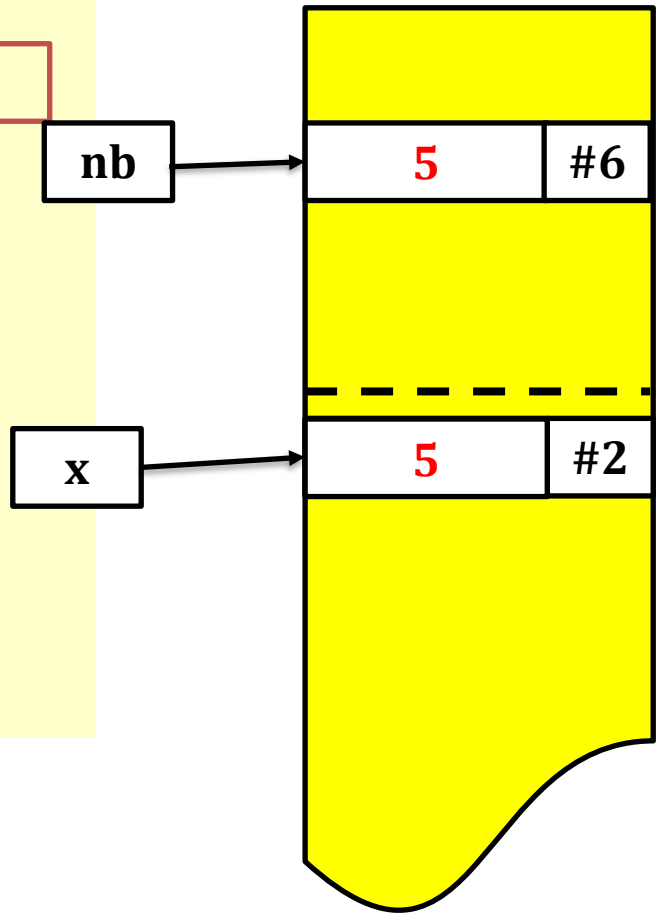



```
Donner un nombre :
5
la valeur avant l' appel est : 5
```

Passage par valeur

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
    Procédure  incrementer (valeur nb: entier)
      Début
            nb ← nb+1
      Fin
Début
     Ecrire("Donner un nombre":)
     Lire(x)
     Ecrire ("la valeur avant l' appel est :", x)
     incrementer(x)
     Ecrire ("la valeur après l' appel est :", x)
fin
```

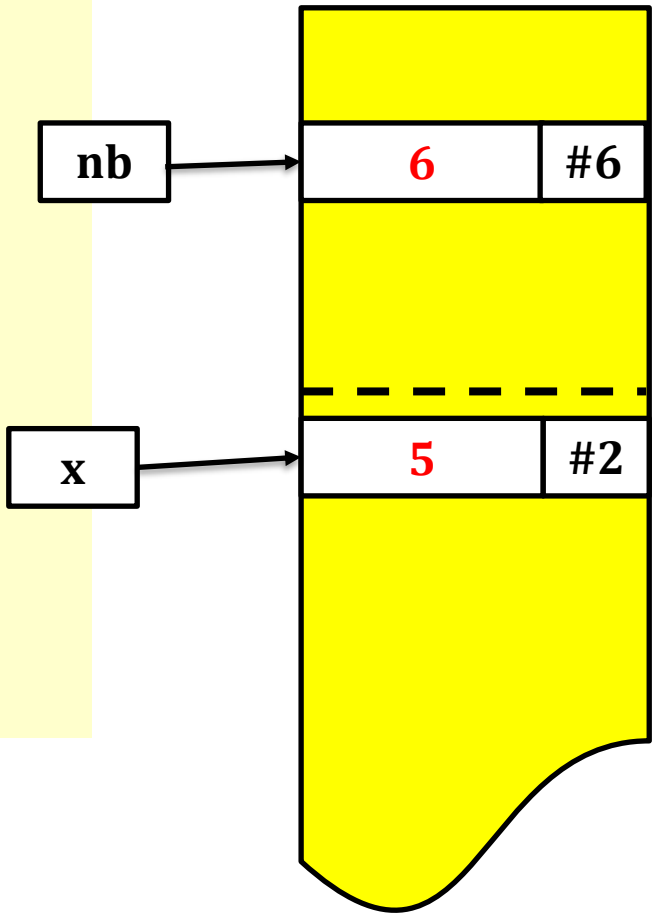



```
Donner un nombre :
5
la valeur avant l' appel est : 5
```

Passage par valeur

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
    Procédure  incrementer (valeur nb: entier)
     Début
          nb ← nb+1
     Fin
Début
    Ecrire("Donner un nombre":)
    Lire(x)
    Ecrire ("la valeur avant l' appel est :", x)
    incrementer(x)
    Ecrire ("la valeur après l' appel est :", x)
fin
```

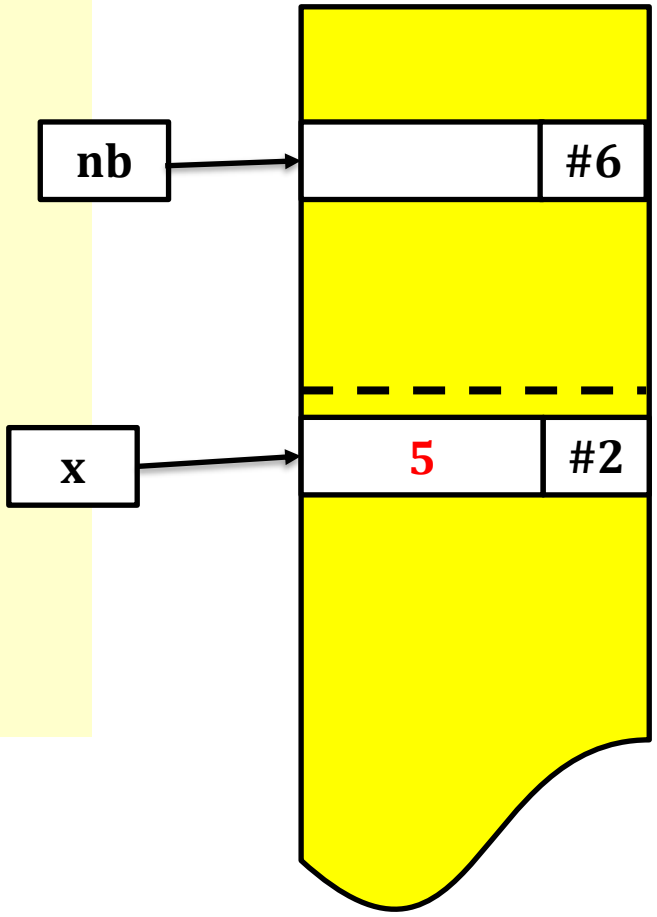



```
Donner un nombre :
5
la valeur avant l' appel est : 5
```

Passage par valeur

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
      Procédure  incrementer (valeur nb: entier)
       Début
            nb ← nb+1
       Fin
Début
      Ecrire("Donner un nombre":)
      Lire(x)
      Ecrire ("la valeur avant l' appel est :", x)
      incrementer(x)
      Ecrire ("la valeur après l' appel est :", x)
fin
```



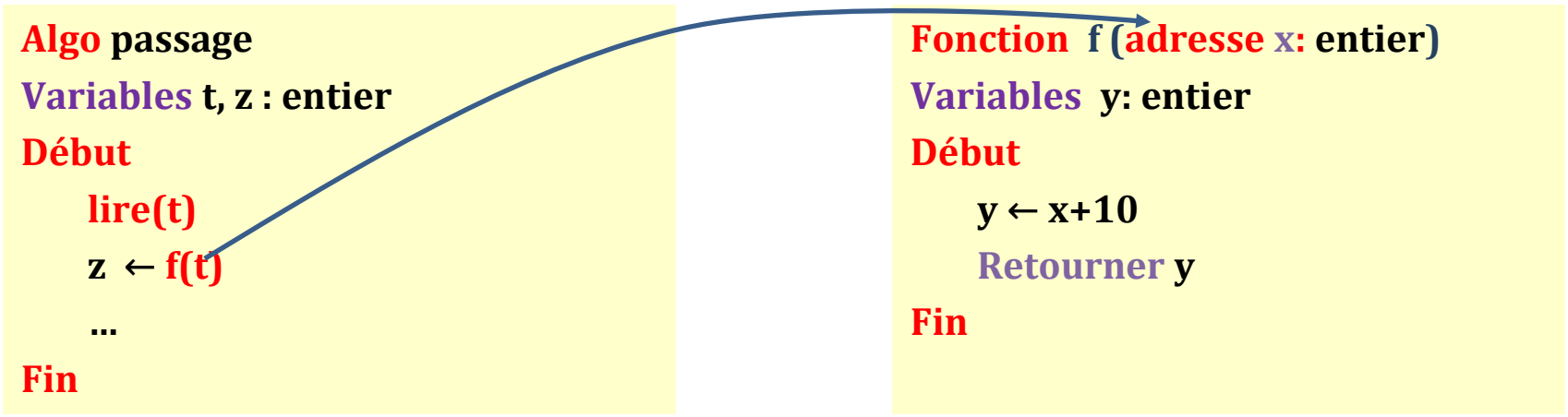

```
Donner un nombre :
5
la valeur avant l' appel est : 5
la valeur après l' appel est : 5
```

Passage par adresse

- On appelle passage par **adresse** quand le paramètre **effectif** qui est **transmis** à la fonction lors de l'appel est **l'adresse de la variable**.
  - La fonction **ne travaille plus sur une copie** de l'objet mais **sur l'objet lui-même**, puisque elle en connait l'adresse.
  - Le paramètre **effectif** est alors **l'adresse** de la variable.
- Principe :
  - la fonction **appelée** range l'adresse transmise dans un paramètre approprié (de type adresse) qui est une variable locale à la fonction appelée.
  - la fonction **appelée** a maintenant accès, via ce paramètre à la variable de la fonction **appelante**.

Passage par adresse

Connexion directe :x et t on la même adresse mémoire



```
Algo passage
Variables t, z : entier
Début
    lire(t)
    z ← f(t)
    ...
Fin
```

```
Fonction f (adresse x: entier)
Variables y: entier
Début
    y ← x+10
    Retourner y
Fin
```

- Toute **modification** sur un paramètre **transmis** par adresse (précèdé par le mot **adresse** (ou var)) entraine la **modification directe** de l'argument correspondant.

Passage par adresse

Exemple : procédure incrémenter

adresse nb: entier ≡ var nb : entier

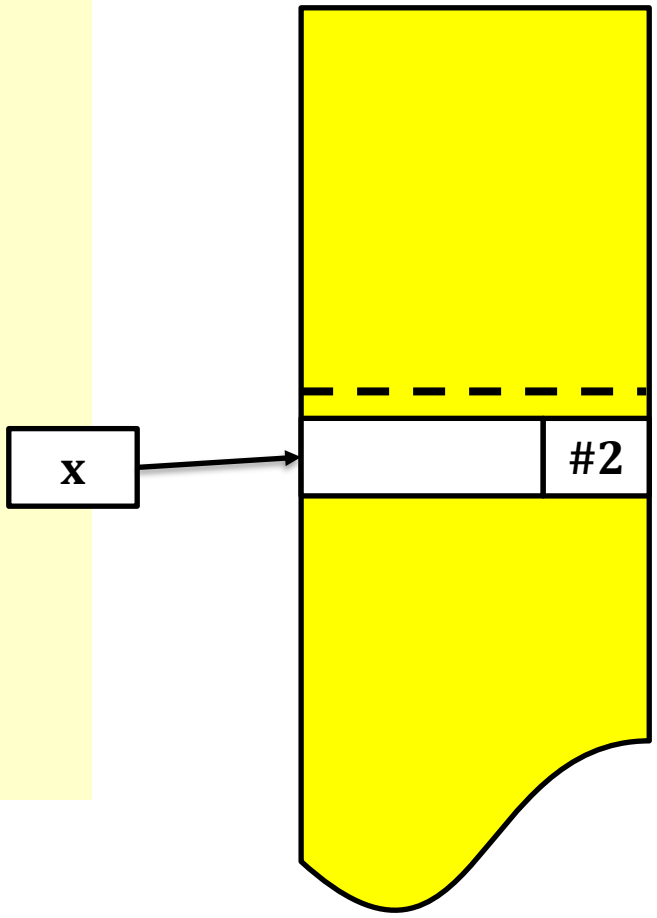
```
Algorithme test_incrementer
Variables  x: entier
     Procédure  incrementer (adresse nb: entier)
      Début
            nb ← nb+1
      Fin
Début
     Ecrire("Donner un nombre":)
     Lire(x)
     Ecrire ("la valeur avant l' appel est :", x)
     incrementer(x)
     Ecrire ("la valeur après l' appel est :", x)
fin
```

```
Donner un nombre :
5
la valeur avant l' appel est : 5
la valeur avant l' appel est : 6
```

Passage par adresse

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
      Procédure  incrementer (adresse nb: entier)
       Début
            nb ← nb+1
       Fin
Variables  x: entier
Début
      Ecrire("Donner un nombre":)
      Lire(x)
      Ecrire ("la valeur avant l' appel est :", x)
      incrementer(x)
      Ecrire ("la valeur après l' appel est :", x)
fin
```

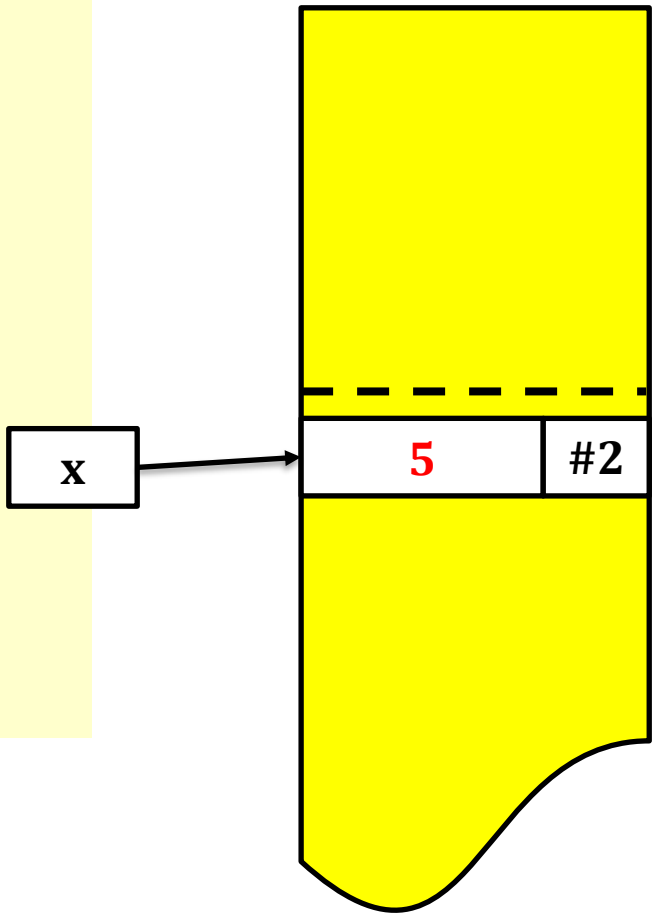


Passage par adresse

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
      Procédure  incrementer (adresse nb: entier)
       Début
            nb ← nb+1
       Fin
Début
     Ecrire("Donner un nombre":)
     Lire(x)
     Ecrire ("la valeur avant l' appel est :", x)
     incrementer(x)
     Ecrire ("la valeur après l' appel est :", x)
fin
```

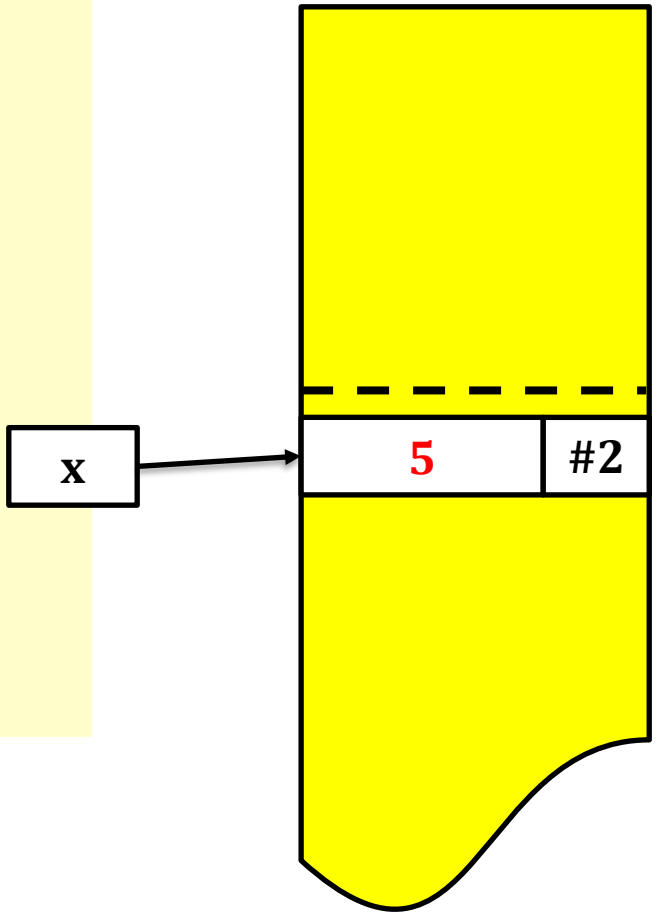
```
Donner un nombre :
5
```



Passage par adresse

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
      Procédure  incrementer (adresse nb: entier)
       Début
            nb ← nb+1
       Fin
Début
      Ecrire("Donner un nombre":)
      Lire(x)
      Ecrire ("la valeur avant l' appel est :", x)
      incrementer(x)
      Ecrire ("la valeur après l' appel est :", x)
fin
```

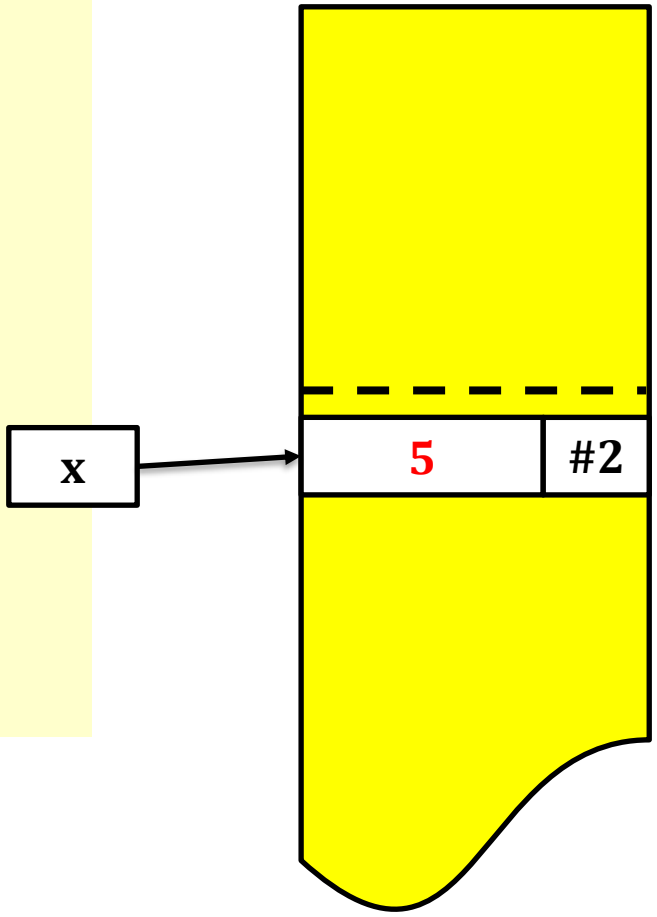



```
Donner un nombre :
5
la valeur avant l' appel est : 5
```

Passage par adresse

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
      Procédure  incrementer (adresse nb: entier)
       Début
            nb ← nb+1
       Fin
Début
      Ecrire("Donner un nombre":)
      Lire(x)
      Ecrire ("la valeur avant l' appel est :", x)
      incrementer(x)
      Ecrire ("la valeur après l' appel est :", x)
fin
```




```
Donner un nombre :
5
la valeur avant l' appel est : 5
```

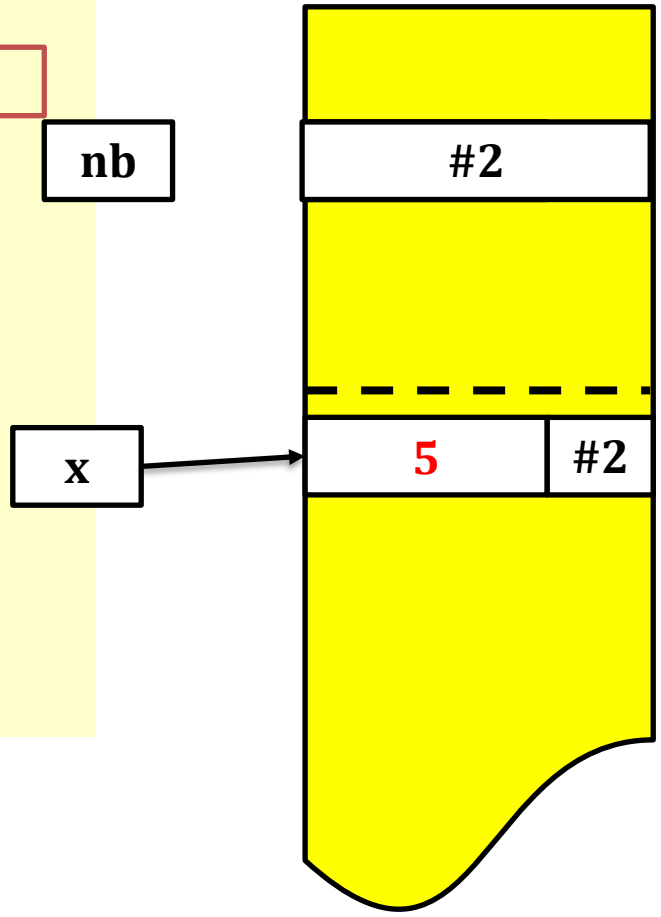
Passage par adresse

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
    Procédure  incrementer (adresse nb: entier)
     Début
          nb ← nb+1
     Fin
Début
    Ecrire("Donner un nombre":)
    Lire(x)
    Ecrire ("la valeur avant l' appel est :", x)
    incrementer(x)
    Ecrire ("la valeur après l' appel est :", x)
fin
```

```
Donner un nombre :
5
la valeur avant l' appel est : 5
```

Le paramètre **formel** nb est maintenant une variable de **type adresse (pointeur)** destinée à recevoir une adresse.

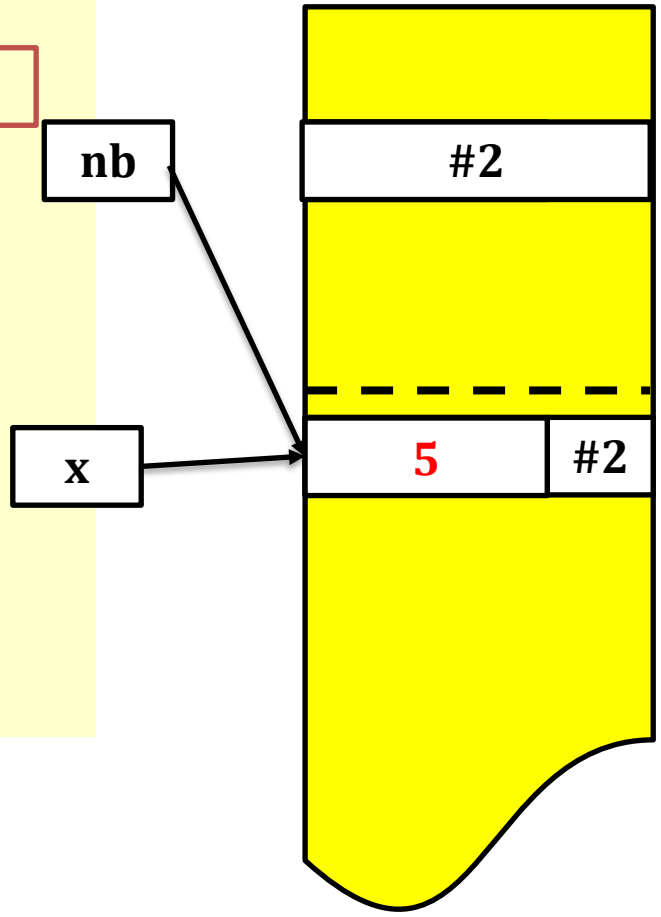


Passage par adresse

Exemple : procédure incrémenter

La fonction peut modifier directement la variable **x** à travers le pointeur **nb** contenant son adresse.

```
Algorithme test_incrementer
Variables  x: entier
    Procédure  incrementer (adresse nb: entier)
      Début
            nb ← nb+1
      Fin
Début
     Ecrire("Donner un nombre":)
     Lire(x)
     Ecrire ("la valeur avant l' appel est :", x)
     incrementer(x)
     Ecrire ("la valeur après l' appel est :", x)
fin
```



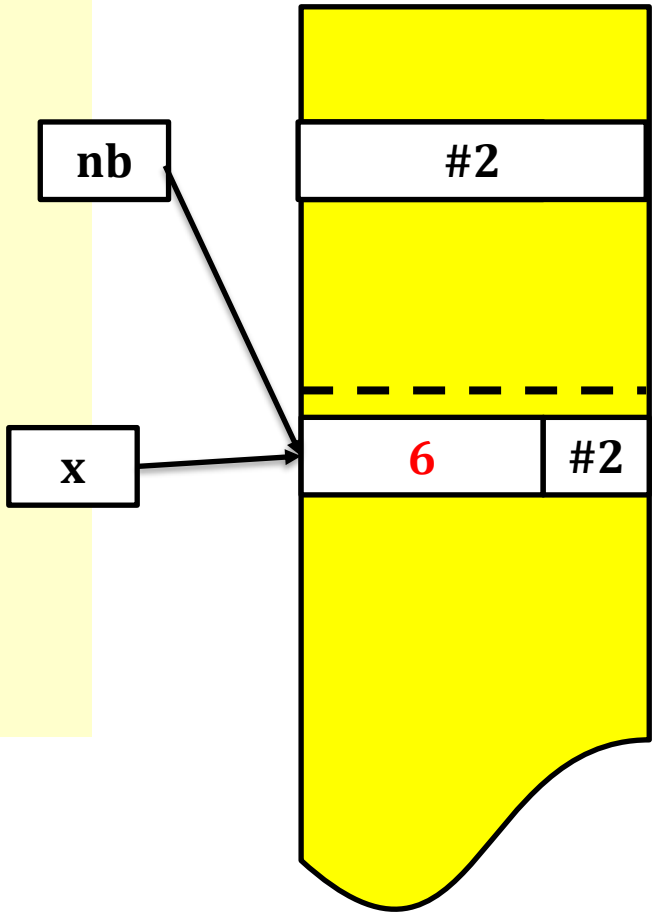

```
Donner un nombre :
5
la valeur avant l' appel est : 5
```

Passage par adresse

Exemple : procédure incrémenter

La fonction peut modifier directement la variable **x** à travers le pointeur **nb** contenant son adresse.

```
Algorithme test_incrementer
Variables  x: entier
    Procédure  incrementer (adresse nb: entier)
     Début
          nb ← nb+1
     Fin
Début
    Ecrire("Donner un nombre":)
    Lire(x)
    Ecrire ("la valeur avant l' appel est :", x)
    incrementer(x)
    Ecrire ("la valeur après l' appel est :", x)
fin
```

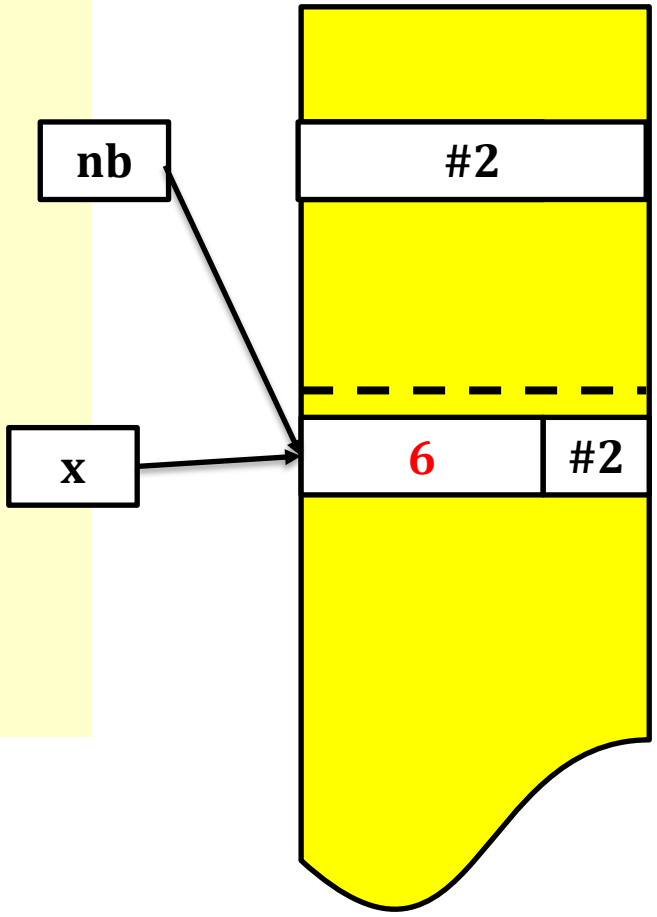



```
Donner un nombre :
5
la valeur avant l' appel est : 5
```

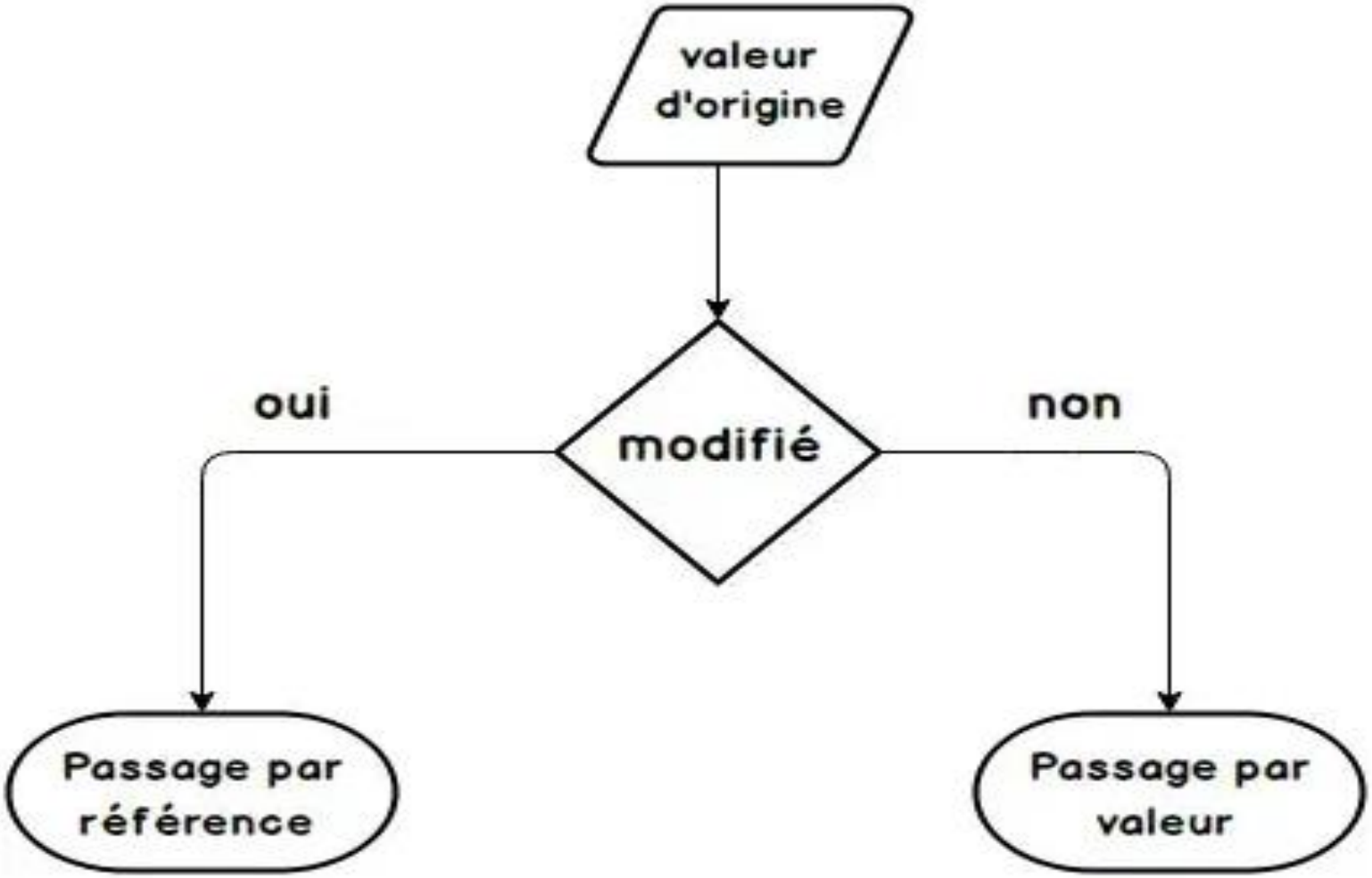
Passage par adresse

Exemple : procédure incrémenter

```
Algorithme test_incrementer
Variables  x: entier
      Procédure  incrementer (adresse nb: entier)
       Début
            nb ← nb+1
       Fin
Début
      Ecrire("Donner un nombre":)
      Lire(x)
      Ecrire ("la valeur avant l' appel est :", x)
      incrementer(x)
      Ecrire ("la valeur après l' appel est :", x)
fin
```




```
Donner un nombre :
5
la valeur avant l' appel est : 5
la valeur après l' appel est : 6
```


valeur d'origine
modifié
oui
non
Passage par référence
Passage par valeur