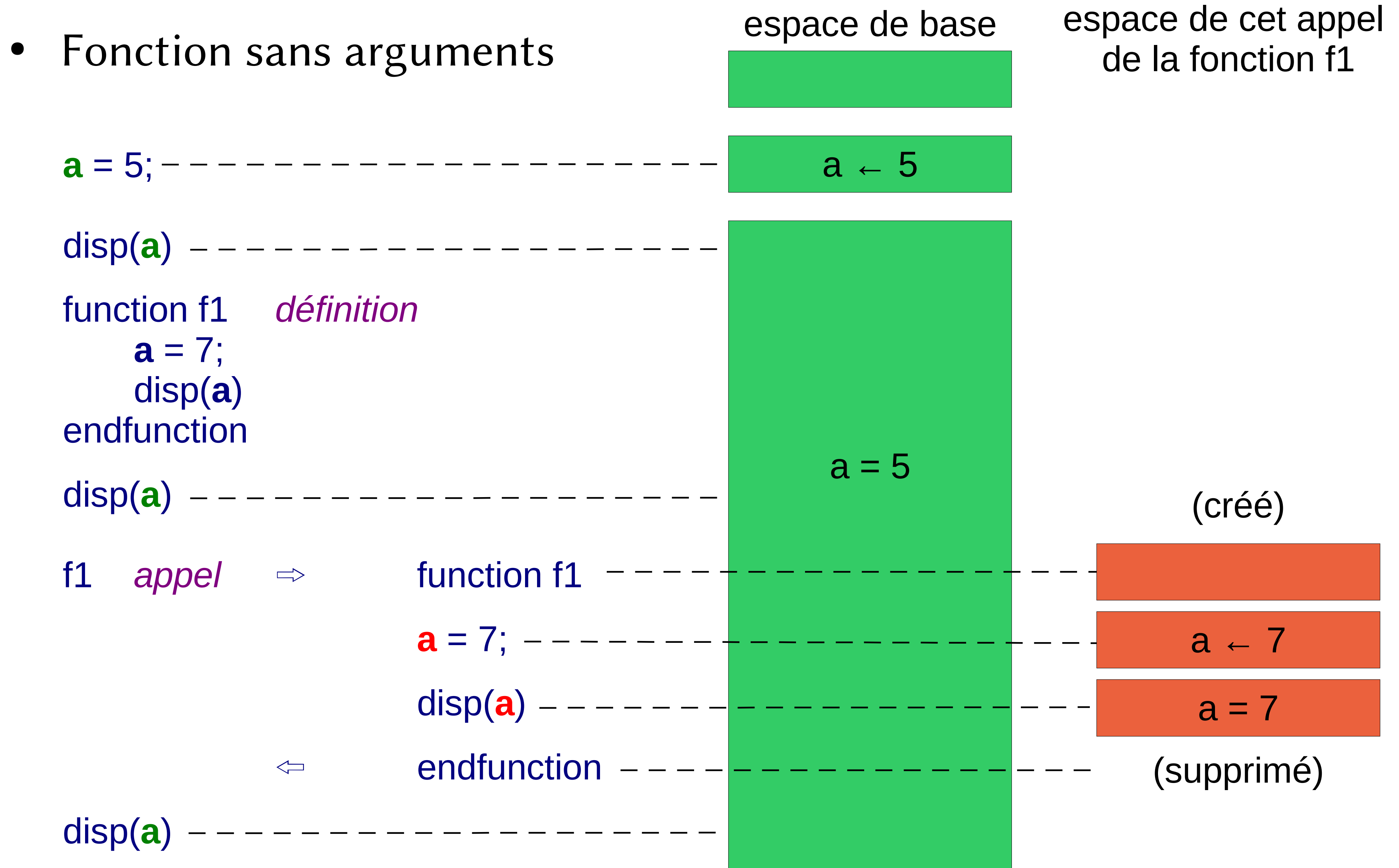
Portée et espaces de variables (suite)

- Chaque appel d'une fonction provoque la création d'un nouveau espace de variables qui sera utilisé lors de l'exécution cette fois-ci
  - À la fin de chaque exécution d'une fonction, l'espace de variables associé est supprimé et les valeurs des variables qui y étaient définies sont oubliées
  - Si la fonction est appelée pour une deuxième fois, ses variables n'ont rien à voir avec celles de l'appel précédent ; ce seront toujours des variables différentes même si portant le même nom
  - Dans certains langages de programmation, il existe des mécanismes qui permettent de conserver, d'un appel à l'autre, les valeurs de variables explicitement désignées à cet effet, appelées **variables statiques**
- Il peut alors exister plusieurs variables avec le même nom, se trouvant dans de différents espaces de variables ; chaque d'entre elles possédera sa propre valeur
  - espace de base – espace de la fonction A – espace de la fonction B – ...
  - appel no. 1 de la fonction A – appel no. 2 de la fonction A – ...
- En général, il n'y a aucun rapport ni lien permanent entre les variables définies dans des espaces de variables différents

Portée des variables – exemple

- Fonction sans arguments



Portée des variables et arguments des fonctions

- On revient à l'exemple précédent
  - **Définition** :

```
function afficherbienvenue(prenom)
    disp(["Bienvenue ", prenom, " dans le système Infores !"])
endfunction
```

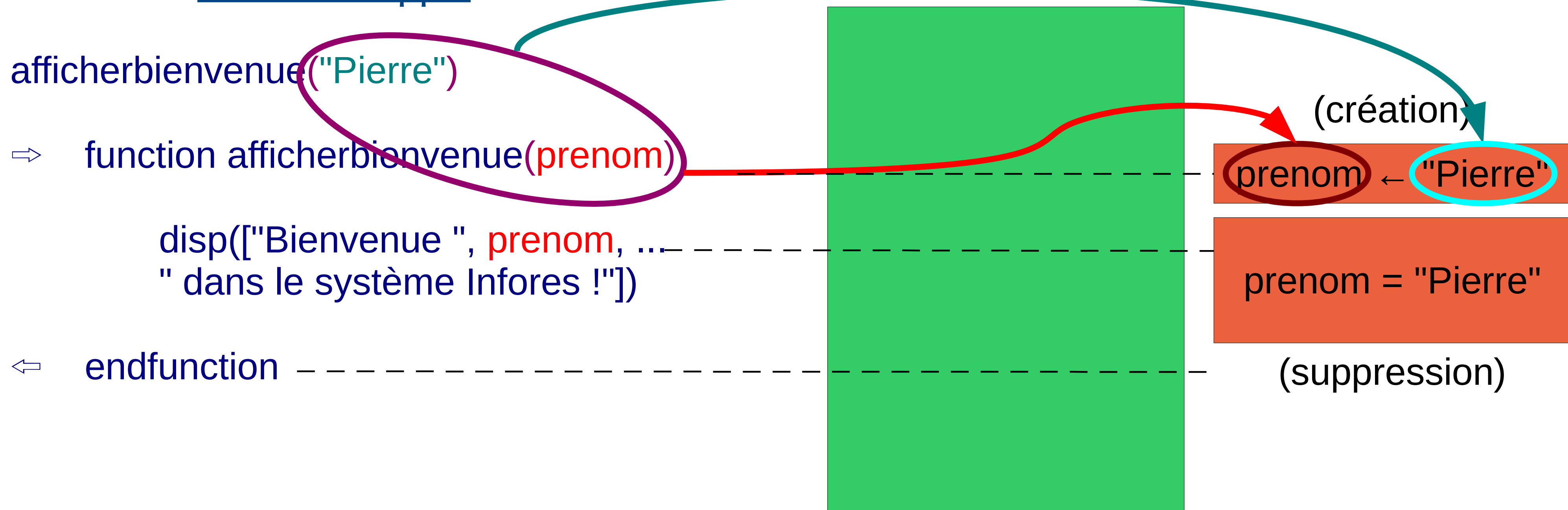
- Analyse
  - prenom est une variable déclarée pour la fonction *afficherbienvenue* comme son argument
  - La variable prenom sera créée dans l'espace de variables de la fonction *afficherbienvenue* comme la première action après que cette fonction est appelée (exécutée)
  - La valeur de prenom sera telle que passée à l'**appel** (et non à la **définition**) de cette fonction

```
afficherbienvenue("Pierre");
```

  - Donc pour chaque appel la valeur de prenom peut être différente

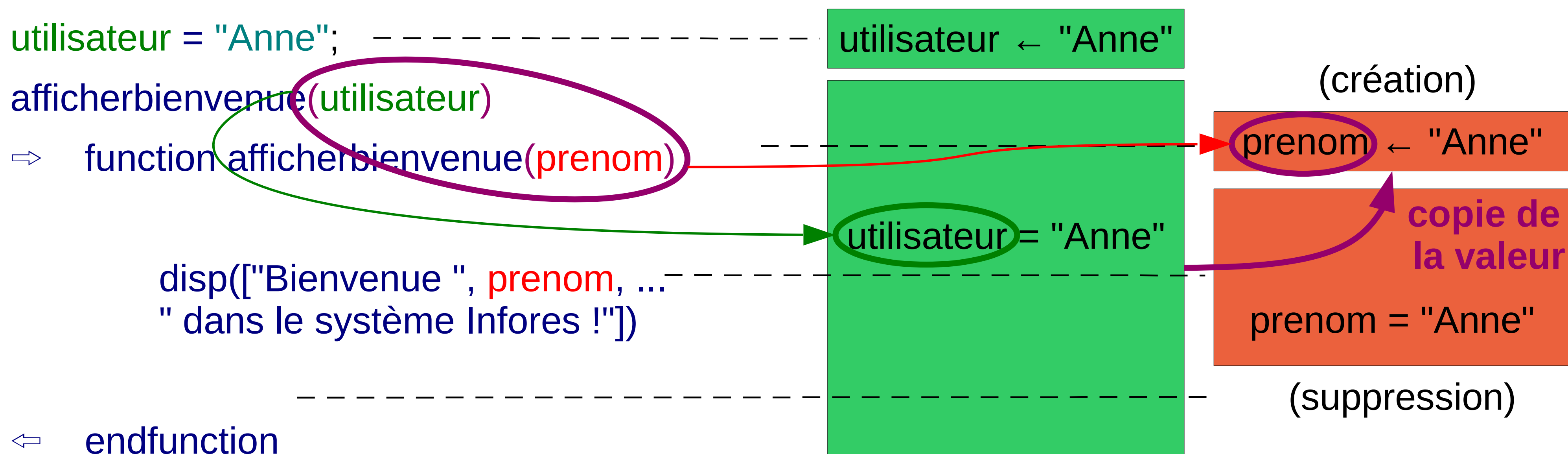
Appel avec passage de l'argument direct (par valeur) explicite (visible)

- Exemple
  - **Appel (invocation)** :
    > afficherbienvenue("Pierre")
    Bienvenue Pierre dans le système Infores !
  - Lors de cet appel (exécution) de la fonction *afficherbienvenue*, la première action, c'est l'affectation prenom ⟵ "Pierre"
  - La valeur de l'argument passée est indiquée entre parenthèses dans la formule d'appel



Appel avec passage de l'argument direct (par valeur) implicite (caché dans une variable)

- Exemple
  - > utilisateur = "Anne";
    > afficherbienvenue(utilisateur)
    Bienvenue Anne dans le système Infores !
  - Lors de cet appel de la fonction *afficherbienvenue*,
    prenom ⟵ valeur de utilisateur (= "Anne") est la première action
  - La variable qui garde la valeur de l'argument à passer est indiquée (par son nom) entre parenthèses dans la formule d'appel
  - utilisateur : espace de base ; prenom : espace de cet appel de la fonction



Mêmes noms dans de différents espaces de variables

- Et si on utilise « prenom » ?
  - Ça ne va pas marcher :
    > afficherbienvenue(prenom)
    car « prenom » n'est pas défini dans l'espace de base : on n'a pas d'où copier la valeur de l'argument !
  - Ça va marcher :
    > prenom = "Guillaume";
    > afficherbienvenue(prenom)
    mais c'est une variable différente de celle qui apparaît dans la fonction !
    - On appelle la fonction *afficherbienvenue*
    - L'interpréteur constate que l'argument de cette fonction s'appelle (à l'intérieur de cette fonction) prenom
    - L'interpréteur vérifie ce qui a été passé comme argument et il effectue l'affectation du prenom :
      prenom ← valeur du prenom qui est "Guillaume"
    - Dès ce moment, prenom devient caché de la fonction ; seulement prenom sera visible et reconnu comme variable là-dedans
- En TP, ce sera <u>interdit</u> pour ne pas comprendre mal ce qui se passe

Variables globales

- Une variable peut être définie de façon explicite comme **globale** ou **à portée globale** ; elle sera alors reconnue (comme la même) dans tous les espaces (Matlab : où elle aura été déclarée comme telle)
  - Une variable globale est placée dans l'espace de base
  - global *nom_de_variable*
  - ```
    function afficherbienvenue
        global prenom
        disp(["Bienvenue ", prenom, " dans le système Infores !"])
    endfunction
    ```
  - ```
    > global prenom
    > prenom = "Helene";
    > afficherbienvenue
    ```
- L'abus des variables globales est une mauvaise pratique
  - Il est très facile d'oublier qu'un nom de variable est déjà occupé
  - Si on veut que la valeur d'une variable soit accessible à l'intérieur d'une fonction, il faut la passer comme argument
  - Si on veut que la valeur d'une variable soit accessible au dehors d'une fonction, il faut la renvoyer comme résultat

Fonctions, arguments, résultats (3)

- Avec des arguments et un résultat renvoyé
  - function *résultat* = *nom*(*argument_1*, *argument_2*, …)
    instructions (où on se sert des argument_x)
    résultat = *expression définissant la valeur du résultat*
    endfunction
  - Cas standard, en fait l'unique appelé « fonction » dans toutes les approches
  - Il est aussi le plus fréquent car normalement
    - les programmes produisent des informations
    - les **produits de l'exécution d'une fonction** sont gardés dans des variables locales dans la mémoire (en vue de leur utilisation ultérieure)
    - ils sont **communiqués = renvoyés** au dehors d'une fonction en tant que ses **résultats = données de sortie**
  - Exemple (un argument, un résultat, une formule)
    - ```
      function ni = inverse(n)
          ni = 1/n;
      endfunction
      ```
    - avec *function*, *n* (nombre) est déclarée comme une variable argument
    - *ni* (nombre inversé) est déclarée comme une variable résultat
      ```

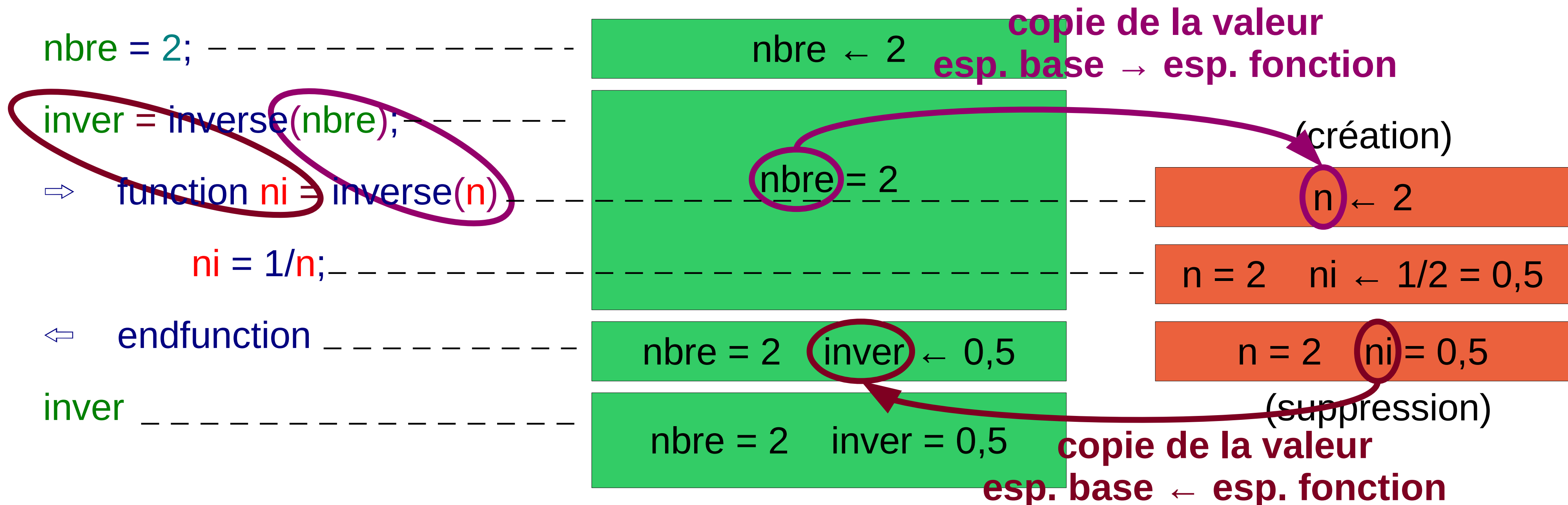
Résultats des fonctions

- function *résultat* = *nom*(*argument_1*, *argument_2*, ...)
  instructions (on se sert des argument_x)
  résultat = *expression définissant la valeur du résultat*
  endfunction

- Le *résultat*
  - Tout comme les *argument_x*, *résultat* est un nom d'une variable librement choisi
  - En plaçant ce nom avant =, on déclare que la valeur de cette variable doit être renvoyée à l'extérieur de la fonction
  - L'affectation du *résultat* peut avoir lieu dans n'importe quelle ligne de la fonction et plusieurs fois mais elle doit avoir lieu au moins une fois
  - La variable *résultat* sera une variable locale définie dans l'espace de variables de la fonction tout comme les variables *argument_x*
- Comment sauvegarder le résultat
  - Normalement on affectera une variable externe (existant au dehors de la fonction) avec la valeur renvoyée pour la garder en mémoire jusqu'au moment où elle on en a besoin

Renvoi du résultat

- Exemple
  - function ni = inverse(n)   *définition*
    ni = 1/n;
    endfunction
  - > nbre = 2;
    > inver = inverse(nbre);   *appel*
    > inver
    inver = 0.5000
  - La variable à garder la valeur renvoyée est indiquée (par son nom) devant l'opérateur de l'affectation dans la formule d'appel



Renvoi du résultat – analyse

- Exemple
  - function ni = inverse(n) *définition*
    ni = 1/n;
    endfunction
  - > nbre = 2;
    > inver = inverse(nbre); *appel*
    > inver
    inver = 0.5000
  - La première action (ligne 1) :
    n ← nbre = 2
    affectation de la **variable argument** « n » : **entre ()** en *définition*
    avec la **valeur passée** de la *variable de source* « nbre » : **entre ()** en *appel*
  - Actions décrites au sein de la fonction (ici ligne 2 seulement) :
    ni ← 1/n = 0.5
  - La dernière action (ici ligne 3) :
    inver ← ni = 0.5
    affectation de la *variable de destination* « inver » : **avant =** en *appel*
    avec la **valeur renvoyée** de la **variable résultat** « ni » : **avant =** en *définition*