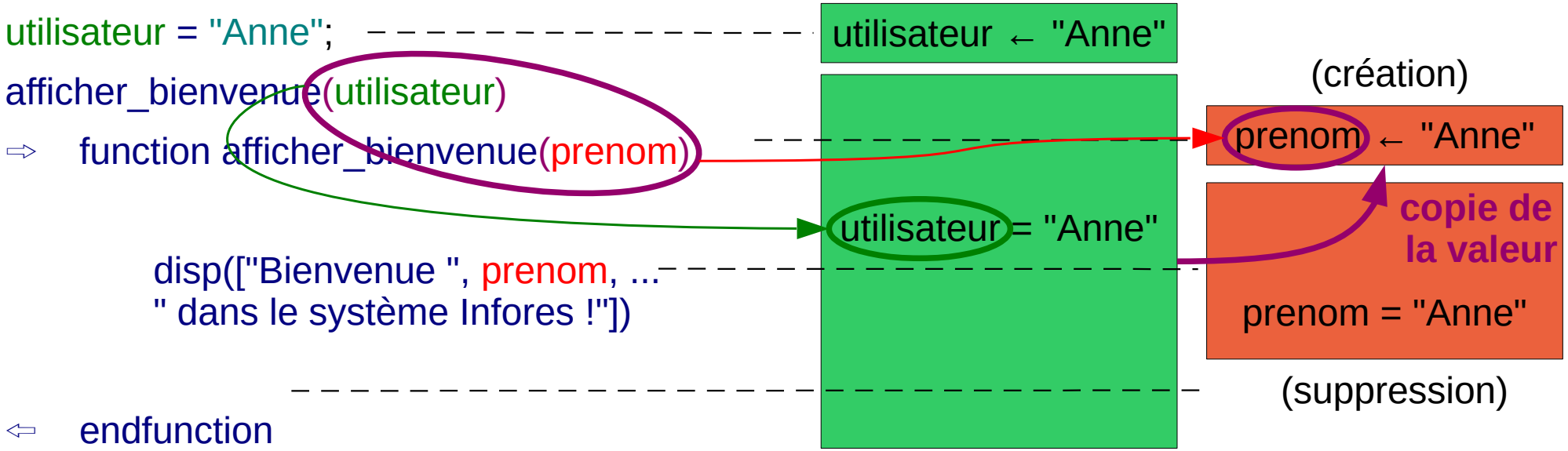
Appel avec passage de l'argument direct (par valeur) implicite (caché dans une variable)

- Exemple
  - > utilisateur = "Anne";
    > afficher_bienvenue(utilisateur)
    Bienvenue Anne dans le système Infores !
  - Lors de cet appel de la fonction *afficher_bienvenue*,
    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 :
    > afficher_bienvenue(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";
    > afficher_bienvenue(prenom)
    mais c'est une variable différente de celle qui apparaît dans la fonction !
    - On appelle la fonction *afficher_bienvenue*
    - 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 afficher_bienvenue
        global prenom
        disp(["Bienvenue ", prenom, " dans le système Infores !"])
    endfunction
    ```
  - ```
    > global prenom
    > prenom = "Helene";
    > afficher_bienvenue
    ```
- 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

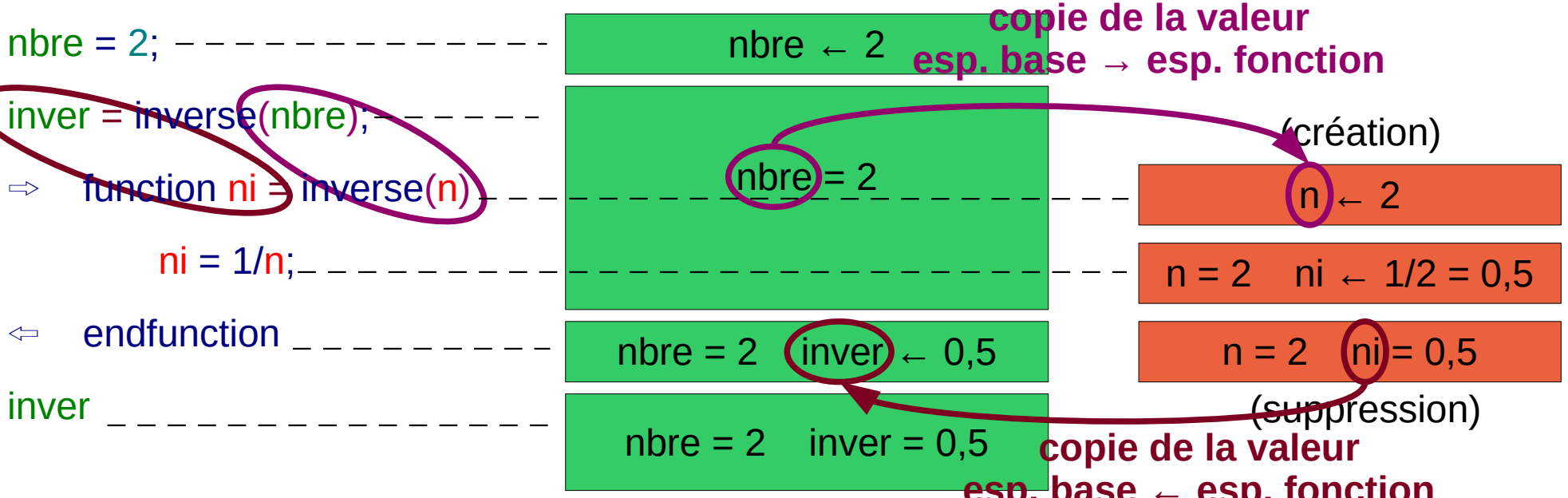
Résultats des fonctions

- 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*

Formes d'appel – arguments

- Passage **direct explicite**
  - inverse(3)
  - On fournit la **valeur** en cause
  - Justifié quand l'argument en cause est fixe à l'endroit donné du programme
    - p. ex. une fonction *rac* calcule la racine *n*-ième d'un nombre ; il faut calculer $a + \sqrt{a} + \sqrt[3]{a}$, alors on écrit a+rac(a,2)+rac(a,3)
- Passage **direct implicite**
  - inverse(nbre)
  - On fournit le nom d'une **variable** qui contient la valeur en cause
  - En général, l'argument peut être une expression : inverse(2*nbre1+nbre2)
  - Cas standard car la plupart de valeurs utilisées dans des programmes sont des paramètres qu'on garde dans des variables
- Il existe aussi passage **indirect** (par référence)
  - On fournit la *référence*, c'est-à-dire l'**adresse** de l'emplacement dans la mémoire où est gardée la valeur en cause
    - ou le nom d'une **variable** qui contient cette adresse – pass. ind. implicite
  - plus compliqué mais souvent plus efficace ; inexistant (en principe) en Matlab

Formes d'appel – résultats

- Résultat sauvegardé dans une variable
  - inver = inverse(3) ou inver = inverse(nbre)
    - la forme du passage d'arguments est indépendante de celle du renvoi de résultats
  - Cas standard car normalement les calculs sont effectués en plusieurs étapes alors les résultats antérieurs doivent être temporairement mémorisés afin de pouvoir servir d'arguments dans des opérations ultérieures
- Résultat non sauvegardé, utilisé comme argument d'une autre fonction ou opérande d'une opération
  - rapport_longueurs = rac(inverse(3),2)
  - tva = pttc * taux * inverse(1+taux)
  - Quand le résultat est utilisé juste une fois et immédiatement après obtention
- Résultat non sauvegardé, non utilisé, juste affiché
  - inverse(2) ou inverse(nbre)
  - Pratique pour tester une fonction lors de son développement – voir le résultat qu'elle produit pour une valeur d'entrée donnée
- Il existe aussi le renvoi indirect (par référence)

Valeurs par défaut

- La **valeur par défaut** d'un argument d'une fonction, c'est la valeur qui sera supposée si aucune valeur n'est passée pour cet argument
  - function *resultat* = *nom*(*argument_1*, ... *argument_i*, *argument_i+1* = *valeur_i+1*, ... *argument_n* = *valeur_n*)
  - une même fonction peut posséder des arguments qui ont et qui n'ont pas une valeur par défaut
  - l'ordinateur doit être capable de déterminer quels arguments ont été omis
  - l'ordinateur supposera l'omission à commencer du dernier argument
  - les arguments possédant des valeurs par défaut doivent être placés à la fin
- Exemple
  - function racine = rac(radicande, degre=2)
    racine = radicande^(1/degre);
    endfunction
  - > rac(8,3)
    ans = 2
    > rac(8,2)
    ans = 2.8284
    > rac(8)
    ans = 2.8284
- ... tandis que
  - function racine = rac(radicande, degre)
    racine = radicande^(1/degre);
    endfunction
  - > rac(8)
    error : degre undefined

4. Structures de contrôle

Algorithmes simples de traitement de données
Classification des structures de contrôle
Boucle incrémentale
Opérateurs de relation et logiques
Alternatives
Boucles conditionnelles
Instructions de terminaison
Comparaison des boucles

Développement d'algorithmes (exemples)

- Moyenne :
  Calculer la moyenne d'éléments d'un vecteur de nombres

- Problème déjà cité :
  Trouver le numéro du mois où les ventes ont été les plus basses.
  S'il y avait plusieurs mois avec le même taux de ventes, renvoyer le premier.
- Simplifions d'abord :
  Trouver le nombre le plus bas dans un vecteur de nombres.
- Et si on modifie ?
  S'il y avait plusieurs mois avec le même taux de ventes, renvoyer le dernier.