Formatage simple de résultats numériques

- format *format*
  - affecte les résultats d'instructions affichés automatiquement
  - et ce que produit *disp*
- *format* peut être :
  - short
    5 chiffres
    > pi
    3.1416
  - long
    15 chiffres
    > pi
    3.14159265358979
  - short e
    5 chiffres + puissance
    > pi
    3.1416e+000
  - long e
    15 chiffres + puissance
    > pi
    3.14159265358979e+000
  - bit
    binaire (base 2)
    > 3
    ans = 11
  - hex
    hexadécimal (base 16)
    > 255
    ans = FF
- Valeurs complexes sont affichées comme
  - 4 + 2i

1.234e+5
veut dire
$1{,}234 \cdot 10^5$

3. Fonctions et portée de variables

Fichiers M, programmes, fonctions
Définition de fonctions, arguments et résultats
Appel de fonctions, passage d'arguments, renvoi de résultats
Portée et espaces de variables

Fichiers M

- Tout programme et fonction Matlab doit être enregistré(e) dans un fichier portant l'extension .m
  - des fichiers programmes sont toujours plein texte, c.-à-d. sans formatage
- Fonction – on va s'y concentrer en TP :
  - commence par la commande *function* et finit par *endfunction*
  - peut posséder (et normalement possède) des arguments **passés** au moment de son **appel** ainsi que des résultats **renvoyés** à la fin de son **exécution**
- Programme :
  - un ensemble quelconque d'instructions (ne commençant par *function*)
  - ne peut pas posséder d'arguments ni de résultats explicites
  - on peut lui fournir des données d'entrée par des fichiers, par l'espace de variables de base ou par la ligne de commandes (sur l'écran)
  - par analogie, lui, il peut fournir des données de sortie
  - son exécution prend place dans l'espace de variables de base
- Pour *lancer un programme* ou *appeler (invoquer) une fonction*
  - on entre juste son nom propre (sans l'extension .m)

Noms (désignations) des fonctions

- Tout comme pour les données (variables) :
  - ne peuvent être composés que des caractères autorisés
  - on applique un modèle choisi pour des noms composés
- Exigence de Matlab
  - Le nom d'un fichier M doit être identique avec celui de la fonction qu'il contient (sauf l'extension .m qui y est ajoutée pour former le nom du fichier)
  - Du côté système d'exploitation, il est plus sûr d'utiliser seules les minuscules
- Convention générale (largement adoptée en génie informatique)
  - Le nom d'une fonction doit être composé d'un verbe (en français à l'infinitif) décrivant ce que la fonction réalise, suivi de substantifs (ou adjectifs) expliquant quelle donnée la fonction concerne (p. ex. *calculer_prix_moyen*)
  - À titre d'exception, une fonction, surtout à fonctionnalité simple ou basique, qui renvoie *un* résultat (pas plus, pas moins) peut être appelée après ce résultat donc avec un ou plusieurs substantifs (p. ex. *moyenne*)
- Différentes fonctions ou versions avec un rôle similaire (en TP)
  - S'il n'est pas possible de les distinguer à l'aide d'un substantif ou adjectif, compléter le nom avec le numéro de l'exercice / sous-point (p. ex. *moyenne3a*)

Répertoire de travail

- Tout logiciel (donc l'interpréteur Octave aussi), dans un moment donné, travaille dans un répertoire (dossier) défini
  - C'est là qu'il va chercher des fichiers (y compris fichiers programmes et fonctions Matlab), enregistrer des résultats...
  - L'adresse d'un répertoire est appelé *chemin*
    - p. ex. *c:\users\patricia\My Documents* dans le système Windows 7
    - il est affiché en haut de la fenêtre Explorateur Windows
- Afin d'être trouvable pour Octave, un fichier M doit se trouver dans le répertoire de travail en cours
  - pwd pour vérifier le répertoire de travail en cours
  - cd(*chemin*) pour changer du répertoire de travail en cours
    - cd('h:\octave');
    - Le *chemin* est un texte donc il faut utiliser les " ou ' (avec ", il faut remplacer chaque \ par \\, \ étant un caractère spécial)
- ou dans un répertoire contenu dans la variable *path*
  - addpath(*chemin*) pour ajouter un répertoire à la variable *path*
  - rmpath(*chemin*) pour effacer un répertoire de la variable *path*

Fonctions, arguments, résultats (1)

- Sans arguments ni résultat renvoyé
  - function *nom*
    instructions
    endfunction
  - Cas le plus simple ; puisqu'il manque du renvoi, dans beaucoup de langages ce n'est pas considéré une *fonction* mais un élément à part : *procédure*
  - C'est utile quand :
    - on veut raccourcir le code du programme principal (le rendre plus lisible) – on place un ensemble de commandes dans une fonction séparée
    - un ensemble d'actions est répété – modification plus simple
  - C'est possible quand :
    - les actions à exécuter sont toujours exactement les mêmes ou
    - elles dépendent de variables globales ou de données qui se trouvent hors le programme (p. ex. dans un fichier)
  - Exemple
    - function afficher_bienvenue
      disp("Bienvenue dans le système Infores")
      endfunction

Fonctions, arguments, résultats (2)

- Avec des arguments mais sans résultat renvoyé
  - function *nom*(*argument_1*, *argument_2*, ...)
    instructions (où on se sert des argument_x)
    endfunction
  - Cas plus fréquent car normalement
    - le fonctionnement de programmes est (doit être) paramétrable
    - les **paramètres de l'exécution d'une fonction** sont gardées dans des variables locales dans la mémoire
    - ils sont **communiqués = passées** à une fonction en tant que ses **arguments = données d'entrée**
  - Aucun effet de l'exécution de la fonction n'est sauvegardé (à moins que dans une variable globale ou p. ex. dans un fichier)
  - Exemple
    - function afficher_bienvenue(prenom)
      disp(["Bienvenue ", prenom, " dans le système Infores"])
      endfunction
  - **Indentation** : les instructions qui appartiennent à la fonction, sont décalées vers la droite par rapport à la ligne de début et à la ligne de fin de la fonction

Portée et espaces de variables

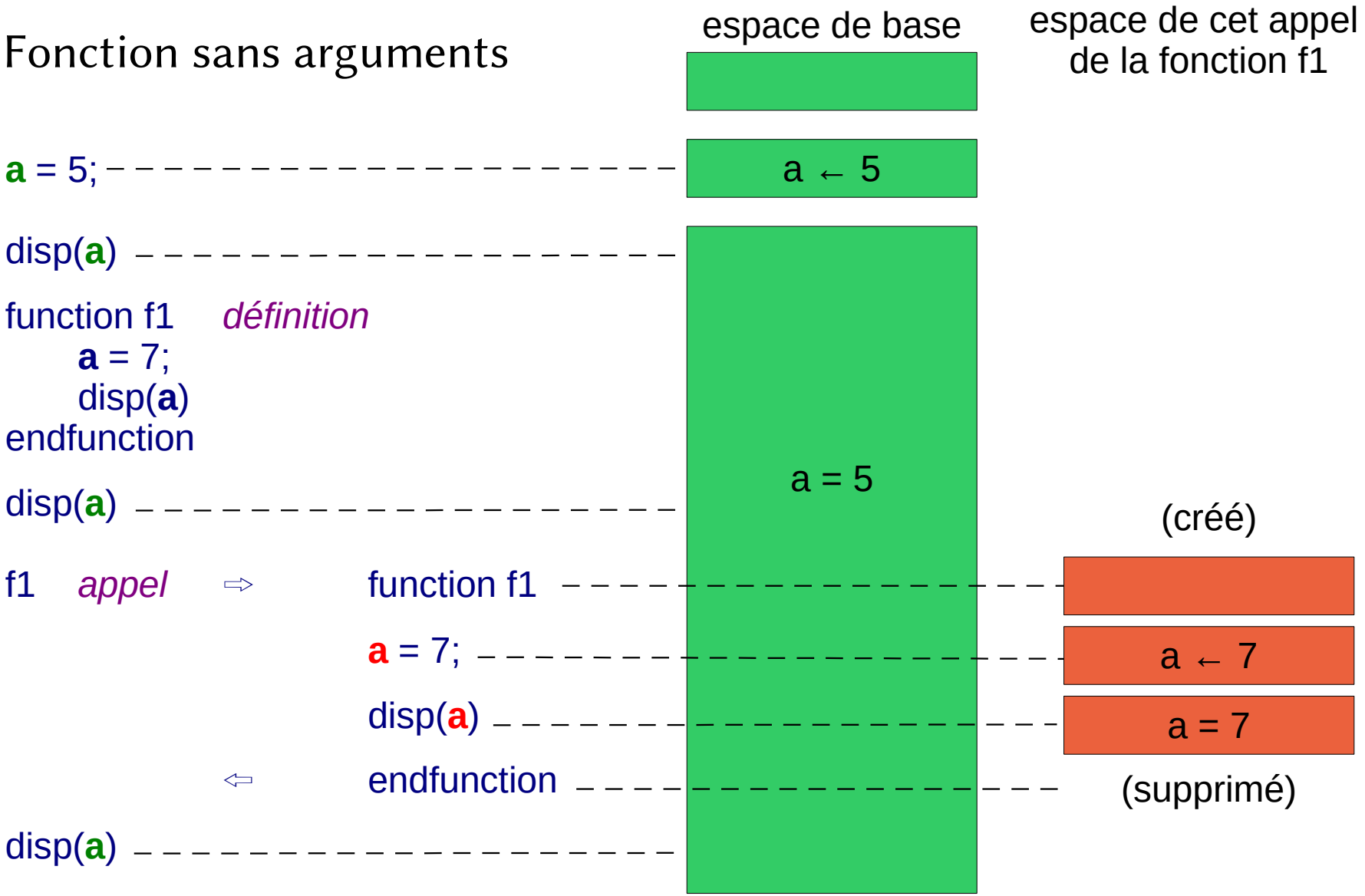
- Un **espace de variables** est un milieu dans lequel des variables sont reconnues et alors peuvent être utilisées
  - Chaque espace possède son propre tableau d'identifiants de variables
- En Matlab il existe :
  - l'**espace de base** – là où vous entrez des commandes en ligne (« invite des commandes », la fenêtre principale Octave) + fichiers M programmes
  - les **espaces des fonctions** – pour chaque fonction, au moment de son chaque appel (et non pas déclaration ou définition), est créé un nouveau, propre espace de variables avec un tableau d'identifiants vide
- Par défaut, une variable est seulement reconnue par l'interpréteur dans l'espace dans lequel elle avait été définie ; on l'appelle alors **variable locale** où **variable à portée locale**
  - Dans l'espace de base ne sont pas reconnues les variables définies dans les fonctions
  - Dans l'espace d'une fonction particulière ne sont pas reconnues les variables définies dans l'espace de base ni dans les espaces d'autres fonctions

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, appellées **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 de variables – exemple

- Fonction sans arguments



Portée de variables et arguments de fonctions

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

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

- Analyse
  - prenom est une variable déclarée pour la fonction *afficher_bienvenue* comme son argument
  - La variable prenom sera créée dans l'espace de variables de la fonction *afficher_bienvenue* 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

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