# Scilab et SIVP

## Option ISN

**Sandrine Duflos**

**24/10/2011**

Initiation à Scilab, lecture et création d'images numériques avec SIVP

# Table des matières

0. Introduction ..... 2
1. Prise en main ..... 2
1.1 Les variables ..... 3
1.2 Sauvegarde des variables ..... 4
1.3 Affichage ..... 5
1.4 Calculs numériques simples ..... 5
1.5 Aide en ligne ..... 5
2 Les matrices ..... 6
2.1 Définition ..... 6
2.2 Extraction de matrice ..... 7
2.3 Opérations sur les matrices ..... 8
2.4 Matrices constantes ..... 8
2.5 Quelques fonctions utiles ..... 10
3 La programmation ..... 11
3.1 L'éditeur de texte ..... 11
3.2 Le langage de programmation Scilab ..... 11
3.2.1 Les opérateurs de comparaison ..... 11
3.2.2 Les instructions conditionnelles ..... 12
3.2.3 Les boucles ..... 14
3.2.3 Les fonctions ..... 16
4 Traitements des images avec SIVP ..... 18
4.1 Introduction ..... 18
4.2 Analogique et numérique ..... 18
4.3 Qu'est-ce une image numérique ? ..... 19
4.4 Représentation numérique d'une image ..... 20
4.5 Images en niveaux de gris ..... 21
4.6 Images en couleurs ..... 22
Correction des exercices ..... 23

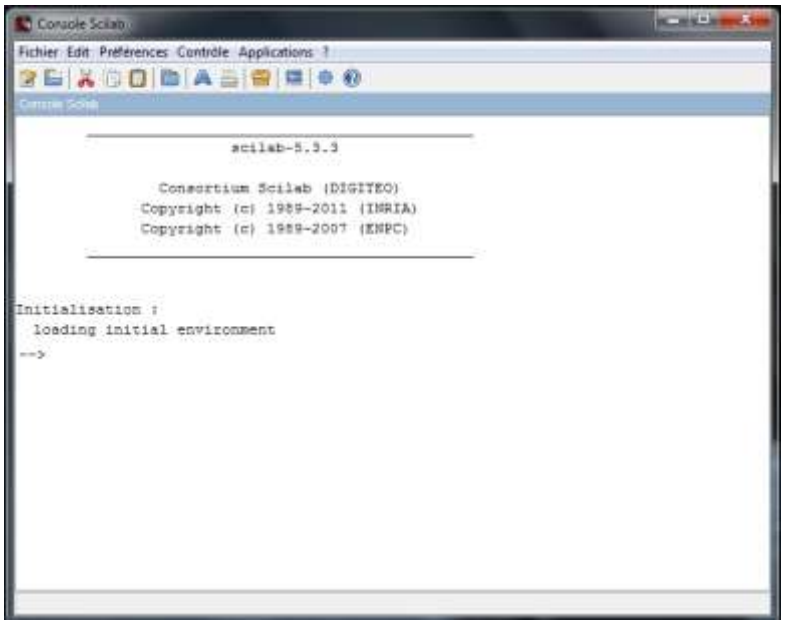
# 0. Introduction

SCILAB est un système interactif pour le calcul numérique, les calculs d'ingénierie, la programmation et la visualisation. Le logiciel SCILAB est un environnement pour la résolution de problèmes. On peut résoudre des calculs complexes beaucoup plus rapidement qu'avec d'autres langages comme le C.
C'est un logiciel écrit à l'origine par des chercheurs de l'INRIA ( Institut National de Recherche en Informatique et Automatique), il est libre et gratuit. Il est maintenant développé par un consortium comprenant l'INRIA et l'ENPC (Appedge, Axs Ingénierie, Cril Technology, CEA, CNES, Dassault Aviation, EDF, ENPC, Esterel Technologies, INRIA, PSA Peugeot Citroën, Renault et Thales).

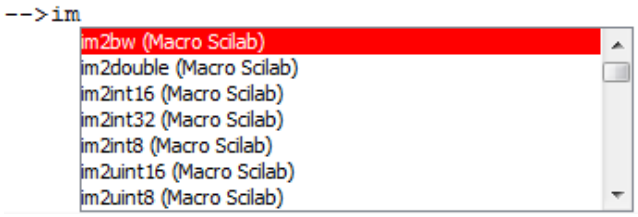
Vous pouvez le télécharger gratuitement sur http : www.scilab.org , ainsi qu'une nombreuse documentation gratuite. Scilab tourne sous UNIX, LINUX, Windows et Mac OSX.

# 1. Prise en main

Lorsqu'on lance Scilab, une fenêtre de commande s'ouvre avec un --> (un prompt, une invite de commande) attendant les données tapées au clavier :




Dans la fenêtre de commande, vous saisissez une commande et tapez la touche « Entrée » pour avoir le résultat.
Attention, Scilab fait la différence entre les minuscules et les majuscules.
Scilab s'utilise comme une super calculatrice.
Il est possible d'exécuter à nouveau un calcul en utilisant les « flèche HAUT » et « flèche BAS » du clavier pour récupérer une commande précédente à travers l'historique des commandes.
Il est également possible d'utiliser la touche « tabulation » pour compléter un nom de fonction ou de variable dont on a tapé les premières lettres. Par exemple, si l'on tape dans la fenêtre de commande :
--> im « tabulation », il apparait une petite fenêtre permettant de choisir toutes les fonctions et noms de variables commençant par « im ». Il suffit alors de cliquer sur la fonction souhaitée pour qu'elle s'affiche.




SCILAB est un langage interprété. Ce qui signifie qu'à chaque étape on obtient un résultat. Dans les langages informatiques usuels FORTRAN, C, ou Pascal on doit avoir un programme prêt avant d'obtenir une réponse.

## 1.1 Les variables

Scilab n'est pas un logiciel de calcul formel. Il n'y a donc pas de variables formelles, toute variable doit avoir une valeur.

Ainsi, demander la valeur de la variable A sans lui donner une valeur produit une erreur :

```
-->A
  !--error 4
Variable non définie: A



-->
```

Mais, si on affecte une valeur à la variable A, il n'y a pas d'erreur :

```
-->A=2
 A  =

    2.

-->
```

SCILAB étant un langage interprété, il n'y a pas besoin de déclarer une variable avant de l'utiliser. Les variables sont créées au moment même où on leur affecte une valeur.

Le principe est le suivant : *variable = expression*
Par exemple le fait de taper la commande : --> A=3 « Entrée » , produit la sortie :

```
 A  =

    3.

-->
```

Cela signifie que SCILAB a créé une variable nommée A, et qu'il lui a affecté pour valeur 3
On assigne à la variable dont le nom est donné à gauche, la valeur de l'expression de droite.

- *En pseudo codé, on écrit : A ← 3*

Si la variable n'existe pas, SCILAB la crée c'est ainsi qu'il a créé la variable A.
Cette variable est maintenant en mémoire et elle est facilement accessible :
Si on tape :
--> A « Entrée »
on obtient de nouveau :
-->A

```
 A  =

    3.

-->
```

Si la variable existe déjà, par exemple la variable A, si vous tapez : --> A=1 « Entrée » , l'ancienne valeur de A sera remplacée par 1.

Les noms de variables consistent en une lettre suivie d'un nombre quelconque de lettres, chiffres ou underscore. SCILAB utilisera les 24 premiers caractères d'un nom de variable.
On peut utiliser n'importe quel nom de variable qui n'est pas déjà défini par le système.

Si on écrit seulement *expression* (sans « *variable* = »), SCILAB crée une variable *ans* par défaut. Par exemple, si on tape --> 5.2 « Entrée »

```
-->5.2
 ans  =

    5.2

-->
```

Certaines variables sont prédéfinies, par exemple %pi, %i, %e

## 1.2 Sauvegarde des variables

Lorsqu'on a créé beaucoup de variables, il peut être utile de récapituler la liste des variables courantes. On obtient cette liste avec la commande **who**.

Si on veut effacer toutes les variables créées, on utilise la commande **clear**. Celle-ci va détruire toutes les variables qui ne sont pas protégées, c'est-à-dire les variables non prédéfinies.
On peut aussi demander la destruction que de certaines variables, par exemple pour détruire A et B :

```
-->clear A B
```

On peut également souhaiter sauvegarder des variables pour les utiliser ultérieurement. On utilisera la commande **save** : **save('fich1', A B)** pour sauver les variables A et B dans le fichier fich1 ou **save ('fich1')** pour sauver toutes les variables courantes dans le fichier fich1.
Si le fichier n'existe pas, Scilab va le créer. Il sera sauvegardé dans le répertoire courant que l'on peut connaître en allant dans le menu « fichier » puis « afficher le répertoire courant »
On peut récupérer toutes ces données sauvegardées avec la commande **load('fich1')**.

## 1.3 Affichage

Si on ne précise rien, SCILAB affichera toujours le résultat de la commande après avoir appuyé sur « Entrée ». Si on ne souhaite pas cet affichage, on ajoute un « ; » à la fin de la ligne. Par exemple :

```
-->A=4;

-->
```

Même s'il ne se passe apparemment rien, SCILAB a bien affecté la valeur 4 à la variable A.
On peut effectuer plusieurs commandes sur une même ligne en séparant chaque commande par un point-virgule si on ne veut pas que le résultat s'affiche à l'écran (par exemple, un calcul intermédiaire). Si au contraire on veut que le résultat soit affiché, il faut séparer les deux commandes par une virgule. Par exemple :

```
-->A=4,B=2,C=A*B;D=2+C,
 A  =

    4.
 B  =

    2.
 D  =

    10.

-->
```

On peut aussi étaler une commande sur plusieurs lignes, ce qui permet de voir entièrement une commande même si elle est trop longue pour tenir sur une seule ligne. Pour cela, on écrit trois points de suspension « ... » à la fin de la ligne qu'on souhaite prolonger. Par exemple :

```
-->B=3*(50)^2+...
-->1/2-sqrt(20)
 B  =

    7496.0279
```

## 1.4 Calculs numériques simples

Dans Scilab, tous les calculs sont numériques, il n'y a pas de calcul formel. Tous les calculs se font avec des nombres flottants, avec une précision maximale de 16 chiffres décimaux et leurs valeurs absolues est comprise entre $10^{-308}$ et $10^{+308}$ (norme IEEE-754).
Par défaut, les résultats sont affichés avec 10 caractères, comprenant le point décimal et le signe. La fonction **format** permet d'afficher plus de chiffre. Par exemple, pour afficher 25 chiffres, on tape :
$\longrightarrow$ format(25)

**<u>Exercice 1.4</u>** : Après avoir exécuté la commande $\longrightarrow$ format(25) , taper l'opération $\longrightarrow 1+10^{-12}$ , puis $\longrightarrow 1+10^{-14}$, puis $\longrightarrow 1+10^{-15}$ et enfin $\longrightarrow 1+10^{-16}$ . Observer les résultats et expliquer le phénomène.

*Pour plus de détail, voir les spécifications de la norme IEEE-754 : http://fr.wikipedia.org/wiki/IEEE_754*

## 1.5 Aide en ligne

Pour accéder à l'aide en ligne, on peut utiliser le menu « ? » puis « Aide Scilab » . On peut également demander directement de l'aide sur une fonction spécifique avec la commande **help** :

```
-->help matrix
```

# 2 Les matrices

Scilab possède une différence importante avec les simples calculatrices ; les calculs dans Scilab sont des calculs matriciels.

## 2.1 Définition

Une matrice est un tableau rectangulaire de nombres comportant un certain nombre de ligne et de colonnes.

Un tableau d'une ligne et $n$ colonnes ou un tableau de $n$ lignes et d'une colonne, sera considéré par Scilab comme un vecteur. Un tableau d'une ligne et d'une colonne est un nombre habituel que l'on appelle également scalaire.

Pour définir un vecteur colonne, on tape par exemple :

```
-->v=[1;2;3;4;5]
 v  =

    1.
    2.
    3.
    4.
    5.
```

Pour définir un vecteur ligne :

```
-->v=[1 2 3 4 5]
 v  =

    1.    2.    3.    4.    5.
```

Pour définir une matrice, par exemple une matrice de 3 lignes et 5 colonnes :

```
-->m=[1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15]
 m  =

    1.     2.     3.     4.     5.
    6.     7.     8.     9.     10.
    11.    12.    13.    14.    15.
```

Ainsi, on laisse un espace « » ou une virgule « , » pour passer d'une colonne à l'autre, et un point-virgule « ; » pour passer d'une ligne à l'autre.

## 2.2 Extraction de matrice

Pour accéder à un élément d'une matrice ou le modifier, par exemple l'élément du tableau situé à la 2ème ligne et 4ème colonne de la matrice **m** précédente :

```
-->m(2,4)
 ans  =

    9.
```

Pour accéder entièrement à une ligne ou à une colonne ou la modifier, par exemple la 2ème ligne ou la 4ème colonne de la matrice **m :**

```
-->m(2,:)
 ans  =

    4.    7.    8.    9.    10.

-->m(:,4)
 ans  =

    4.
    9.
    14.
```

Pour accéder entièrement à la dernière ligne ou à la dernière colonne ou la modifier, par exemple pour la matrice **m :**

```
-->m($,:)
 ans  =

    11.    12.    13.    14.    15.

-->m(:,$)
 ans  =

    5.
    10.
    15.
```

Et enfin, pour extraire une matrice plus petite à partir d'une autre, par exemple une matrice **m2** de 2 lignes et 3 colonnes, extraite de la matrice **m :**

```
-->m2=m(2:3,2:4)
 m2  =

    7.     8.     9.
    12.    13.    14.
```

**<u>Exercice 2.2 :</u>**

Créer la matrice M= $\begin{pmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \end{pmatrix}$

Sans redéfinir élément par élément la matrice M, modifier celle-ci pour obtenir :

$$M=\begin{pmatrix} \mathbf{2} & 12 & 13 & 14 & \mathbf{0} \\ 21 & 22 & \mathbf{5} & 24 & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} \end{pmatrix}$$

## 2.3 Opérations sur les matrices

On peut multiplier une matrice par un réel, par exemple **m3=2×m** donnera une matrice **m3**, de même dimensions que **m** et dont chacun des éléments est le double de celui de **m** à la même position, soit **m3**$(i,j)$=**2×m**$(i,j)$ **:**

```
-->m3=2*m
 m3  =

    2.     4.     6.     8.     10.
    8.     14.    16.    18.    20.
    22.    24.    26.    28.    30.
```

On peut ajouter deux matrices de mêmes dimensions entre elles, par exemple **m+m3,** on obtient une somme élément par élément :

```
-->m+m3
 ans  =

    3.     6.     9.     12.    15.
    12.    21.    24.    27.    30.
    33.    36.    39.    42.    45.
```

Attention, la multiplication « * » et la division « / » habituelles, sont des opérations matricielles que nous ne verrons pas ici. Pour une multiplication ou une division élément par élément de deux matrices, il faut utiliser le « .* » et le « ./ ». Par exemple pour obtenir une matrice contenant les carrés de chacun des éléments de **m** :

```
-->m.*m
 ans  =

    1.     4.     9.     16.    25.
    16.    49.    64.    81.    100.
    121.   144.   169.   196.   225.
```

## 2.4 Matrices constantes

Des matrices constantes sont prédéfinies dans Scilab. On les obtient avec les fonctions **ones, zeros** et **eye**.
Par exemple, pour obtenir une matrice 3 × 4 (3 lignes et 4 colonnes) ne contenant que des zéros :

```
-->zeros(3,4)
 ans  =

    0.     0.     0.     0.
    0.     0.     0.     0.
    0.     0.     0.     0.
```

**<u>Exercice 2.4 :</u>** 1) Regarder l'aide en ligne pour les fonctions **ones** et **eye**.
2) Créer une matrice M0 10 × 15 ne contenant que des 0. Créer ensuite deux matrices 5 × 5, une, Mi, avec la fonction **eye** et une autre, M1, ne contenant que des 1.
3) Transformer **M0** afin que **M0(1:5,1:5)=Mi**, **M0(6:10,6:10)=M1** et **M0(1:5,11:15)=2 ×Mi**.

**Résultat** : On obtient la matrice **M0** suivante :

```
-->M0
 M0  =

    1.    0.    0.    0.    0.    0.    0.    0.    0.    0.    2.    0.    0.    0.    0.
    0.    1.    0.    0.    0.    0.    0.    0.    0.    0.    0.    2.    0.    0.    0.
    0.    0.    1.    0.    0.    0.    0.    0.    0.    0.    0.    0.    2.    0.    0.
    0.    0.    0.    1.    0.    0.    0.    0.    0.    0.    0.    0.    0.    2.    0.
    0.    0.    0.    0.    1.    0.    0.    0.    0.    0.    0.    0.    0.    0.    2.
    0.    0.    0.    0.    0.    1.    1.    1.    1.    1.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    1.    1.    1.    1.    1.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    1.    1.    1.    1.    1.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    1.    1.    1.    1.    1.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    1.    1.    1.    1.    1.    0.    0.    0.    0.    0.
```

## 2.5 Quelques fonctions utiles

Il faut toujours connaître les dimensions d'une matrice pour le traitement des images puisque les images numériques y sont représentées par des matrices. On utilise alors la fonction **size** :

```
-->size(m)
 ans  =

    3.    5.
```

Si on doit utiliser ces dimensions plusieurs fois dans un programme, par exemple dans des boucles, on peut alors donner un nom spécifique au vecteur retourné par la fonction **size** :

```
-->v=size(m)
 v  =

    3.    5.
```

Le nombre de lignes de la matrice est alors **v(1,1)** et le nombre de colonnes **v(1,2)** :

```
-->v(1,1)
 ans  =

    3.

-->v(1,2)
 ans  =

    5.
```

On peut aussi créer des variables ayant pour valeur les dimensions de la matrice :

```
-->[l,c]=size(m)
 c  =

    5.
 l  =

    3.
```

Pour connaitre le plus grand ou le plus petit élément d'une matrice :

```
-->max(m)
 ans  =

    15.

-->min(m)
 ans  =

    1.
```

Pour connaître la moyenne ou l'écart type des éléments d'une matrice :

```
-->mean(m)
 ans  =

    7.8666667

-->st_deviation(m)
 ans  =

    4.5648763
```

# 3 La programmation

Scilab dispose d'un langage de programmation simple mais assez complet.
Comme nous l'avons déjà vu, contrairement à d'autres langages tels que C, C++, Pascal, …. il n'est pas nécessaire de déclarer les variables avant de les utiliser. C'est l'interpréteur de Scilab qui, lorsqu'il rencontre un nouvel objet, se charge de ce travail.
Cette caractéristique pourrait conduire à des erreurs dans des programmes importants, mais dans notre cas, ça ne posera pas de problème car nous n'allons manipuler que très peu de variables et programmer des petits programmes.
Ainsi, les programmes Scilab sont bien plus simples à écrire que ceux écrits dans des langages non interprétés, mais ils sont aussi vraiment beaucoup plus lents que des programmes écrits en C.
Lorsqu'on manipule des matrices importantes, ça sera notre cas avec les images, il faut utiliser au maximum les commandes disponibles et les instructions matricielles, c'est-à-dire réduire au maximum les boucles. Ceci permet à Scilab d'utiliser des routines compilées et de moins utiliser son interpréteur.

## 3.1 L'éditeur de texte

Vous trouverez un éditeur de texte intégré à Scilab dans le menu « Application » puis SciNotes.

Il faut écrire les lignes de commandes et les fonctions dans l'éditeur puis les charger ensuite dans Scilab à partir du menu « Exécuter », puis « ….. fichier sans écho » ou « …. fichier avec écho » si l'on veut voir apparaitre la suite d'instructions dans la fenêtre de la Console Scilab.

Avant d'exécuter cette liste de commandes et de fonctions de l'éditeur de texte, il faudra donc d'abord la sauvegarder dans un fichier.

On peut aussi exécuter une liste de commandes sauvegardée dans un fichier directement à partir de la console Scilab avec la commande **exec('nom_fichier.sce').** C'est une exécution du fichier avec écho.
**Attention** : Il faut faire attention au répertoire dans lequel on travaille. Scilab travaille toujours dans le répertoire par défaut. On peut le connaître en allant dans « menu » puis « Afficher le répertoire courant ». Vous allez devoir changer ce répertoire courant pour pouvoir enregistrer vos fichiers ou utiliser des images de votre compte. Pour cela, il faut aller dans « menu » puis « changer de répertoire courant » puis naviguer jusqu'à ce que vous soyez dans votre répertoire de travail Scilab personnel.

Tout texte écrit derrière « // » est un commentaire. Les commentaires ne sont là que pour aider le lecteur du fichier, ils sont ignorés par Scilab.

## 3.2 Le langage de programmation Scilab

### 3.2.1 Les opérateurs de comparaison

| Scilab | Maths |
|---|---|
| == ou = | = |
| < | < |
| > | > |
| >= | ≥ |
| <= | ≤ |
| <> ou ~= | ≠ |
| & | et |
| \| | ou |
| ~ | non |

### 3.2.2 Les instructions conditionnelles

#### *a) Le Si ….. Alors*

| Scilab | Pseudo-codé |
|---|---|
| **if** condition_1 **then**<br>Suite d'instructions 1<br>**else**<br>Suite d'instructions 2 ;<br>end | **Si** condition_1 **Alors**<br>Suite d'instructions 1<br>**Sinon**<br>Suite d'instructions 2 ;<br>**FinSi** |
| **if** condition_1 **then**<br>Suite d'instructions 1<br>**elseif** condition_2 then<br>Suite d'instructions 2<br>**elseif** condition_3 then<br>Suite d'instructions 3<br>**else**<br>Suite d'instructions 4 ;<br>**end** | **Si** condition_1 **Alors**<br>Suite d'instructions 1<br>**Sinon si** conditions_2 Alors<br>Suite d'instructions 2<br>**Sinon si** conditions_3 Alors<br>Suite d'instructions 3<br>**Sinon**<br>Suite d'instructions 4 ;<br>**FinSi** |

**Exemple :**

| Scilab | Pseudo-codé |
|---|---|
| a=grand(1,1,'uin',1,6)<br>if a==6 then disp(" gagné !")<br>else disp("perdu !")<br>end | **Variable** : a : entier ;<br>**Début**<br>a←Aléa_Ent(1 :6) ;<br>**Si** a=6 **Alors**<br>Afficher (« gagné ! »)<br>**Sinon**<br>Afficher (« perdu !») ;<br>**FinSi ;**<br>**Fin** |

**<u>Exercices 3.2.2.a</u>**:

1) Remplir le tableau de simulation ci-dessous puis décrire le programme ci-dessus.

| | valeur variable a | test de condition | instruction |
|---|---|---|---|
| Initialisation | 6 | | |
| a=6 ? | | vrai | « gagné » |
| FinSi | | | |

| | valeur variable a | test de condition | instruction |
|---|---|---|---|
| Initialisation | 2 | | |
| a=6 ? | | _____ | _____ |
| FinSi | | | |

2) Ecrire un programme permettant de simuler le jeu suivant :
Pour une mise initiale de 3€, un joueur lance deux dés équilibrés à six faces. Si le résultat est un double, le joueur empoche en euros la somme des deux faces. Si un seul 6 apparait, le joueur empoche en euros le montant indiqué sur l'autre face. Dans les autres cas, la partie est perdue.

**<u>Remarques</u>** : La commande **disp(A,B)** affiche d'abord B puis A
Pour utiliser le « OU », soit « | » en scilab ; tapez : Alt Gr 6

### b) Le selon

| Scilab | Pseudo-codé |
|---|---|
| **select** var<br>**case** 1 Suite d'instructions 1<br>**case** 2 Suite d'instructions 2<br>**case** 3 Suite d'instructions 3<br>**else** Suite d'instructions 4 ;<br>**end** | **Selon** var=1 Suite d'instructions 1<br>var=2 Suite d'instructions 2<br>var=3 Suite d'instructions 3<br>autres cas Suite d'instructions 4 ;<br>**FinSelon** |

**<u>Exercice 3.2.2.b</u>:** Réécrire le programme de l'exercice précédent en utilisant le **select case** (indice : utiliser la somme ou le produit de la valeur des faces des deux dés).

| | Valeur Face 1 | Valeur Face 2 | Somme | Test de condition | Gain |
|---|---|---|---|---|---|
| Initialisation | 4 | 4 | 8=2×4 | | −3 |
| Produit = ? | | | | vrai | 5 |
| FinSelon | | | | | |

| | Valeur Face 1 | Valeur Face 2 | Somme | test de condition | Gain |
|---|---|---|---|---|---|
| Initialisation | 6 | 2 | 8=2+6 | | −3 |
| Produit = ? | | | | faux | −3 |
| Produit = ? | | | | vrai | 1 |
| FinSelon | | | | | |

### 3.2.3 Les boucles

#### *a) La boucle pour*

| Scilab | Pseudo-codé |
|---|---|
| **for** i=$i_{début}$ : $i_{fin}$<br>Suite d'instructions<br>**end** | **Pour** i allant de $i_{début}$ à $i_{fin}$ **faire**<br>Suite d'instructions<br>**FinPour** |

**Exemple :**

| Scilab | Pseudo-codé |
|---|---|
| S=0;<br>for i=1:20<br>S=S+i;<br>end<br>disp(S) | **Variables** : i,S : entiers ;<br>**Début**<br>S← 0 ;<br>**Pour** i allant 1 à 20 **faire**<br>S← $S + i$ ;<br>**FinPour**<br>Afficher (S) ;<br>**Fin** |

**<u>Exercice 3.2.3.a</u>** : Reprendre les règles du jeu de l'exercice précédent et simuler le gain pour une partie de 10 jets de deux dés.

| | Valeur Face 1 | Valeur Face 2 | Gain |
|---|---|---|---|
| Initialisation | | | 0 |
| 1ère boucle | | | −3+ ____= _____ |
| 2ème boucle | | | _____−3+_____=_____ |
| 3ème boucle | | | |
| 4ème boucle | | | |
| 5ème boucle | | | |
| 6ème boucle | | | |
| 7ème boucle | | | |
| 8ème boucle | | | |
| 9ème boucle | | | |
| 10ème boucle | | | |

### b) La boucle Tant que

| Scilab | Pseudo-codé |
|---|---|
| **while** condition<br>Suite d'instructions<br>**end** | **Tant que** condition **faire**<br>Suite d'instructions<br>**FinTanque** |

**Exemple :**

| Scilab | Pseudo-codé |
|---|---|
| S=1000<br>i=0<br>while S<2000<br>S=S*1.05<br>i=i+1<br>end<br>disp(i,S) | **Variables** : i,S : entiers ;<br>**Début**<br>S← 1000 ; i← 0<br>**Tant que** S<2000 **faire**<br>S← S × 1,05 ;<br>i← i + 1;<br>**FinTantque**<br>Afficher (i,S) ;<br>**Fin** |

**<u>Exercice 3.2.3.b</u>** :

1) Que se passe-il si on initialise l'algorithme avec S=0 ?
2) Toujours en faisant une simulation du même jeu de dés, compter au bout de combien de parties le joueur arrive à avoir un gain de plus de 5 €, tout en limitant ses pertes à 20 € .

| | Test Gain <5 | Nbre partie | Valeur Face 1 | Valeur Face 2 | Gain |
|---|---|---|---|---|---|
| Initialisation | | 0 | | | 0 |
| 1ère boucle | vrai | 1 | | | |
| 2ème boucle | | | | | |
| 3ème boucle | | | | | |
| 4ème boucle | | | | | |
| 5ème boucle | | | | | |
| 6ème boucle | | | | | |
| 7ème boucle | | | | | |
| 8ème boucle | | | | | |
| 9ème boucle | | | | | |
| 10ème boucle | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

### 3.2.3 Les fonctions

#### a) *Définition et syntaxe*

Lorsqu'on doit utiliser plusieurs fois une suite d'instructions, on peut définir une fonction correspondant à celles-ci qui permettra de ne pas devoir réécrire toute la suite d'instructions à chaque fois.

##### i) *Les procédures*

Par exemple, si l'on a besoin plusieurs fois dans un programme d'afficher trois nombres aléatoires :

```
disp(rand(1,1)
disp(rand(1,1)
disp(rand(1,1)
```

On peut alors définir la fonction troisnbresaleatoires() avec la syntaxe suivante :

```
function troisnbresaleatoires()
    disp(rand(1,1))
    disp(rand(1,1))
    disp(rand(1,1))
endfunction
```

Lorsqu'on aura besoin de cette fonction dans un programme, il suffira de l'appeler avec l'instruction :

```
troisnbresaleatoires()
```

Une fonction peut aussi utiliser un ou plusieurs arguments en entrée. Si par exemple, on ne veut pas toujours afficher trois nombres aléatoires, mais un nombre $n$ pouvant varier. On définira alors la fonction avec cette syntaxe :

```
function nnbresaleatoires(n)
    for i=1:n
        disp(rand(1,1))
    end
endfunction
```

Si on veut afficher 4 nombres aléatoires, on pourra alors appeler cette fonction avec l'instruction:

```
nnbresaleatoires(4)
```

Pour une fonction avec plusieurs arguments en entrée : nomfonction(arg1,arg2,arg3,…).

##### ii) *Les fonctions*

On peut aussi souhaiter que la fonction retourne une ou plusieurs valeurs. Par exemple, la fonction calculant l'aire et le périmètre d'un triangle rectangle qui prend en argument les longueurs des plus petits côtés : on la définira avec

```
function [A, P]=triangle(a, b)
  A=(a*b)/2
  P=a+b+sqrt(a^2+b^2)
  disp(A,"Aire =",P,"Périmètre=")
endfunction
```

On pourra l'appeler, par exemple, avec l'expression :

```
[surface,perim]=triangle(4,3)
```

### b) *Variables locales et variables globales*

Exemple1: Soit la procédure définie de la manière suivante :

```
function argumentvauttrois(a)
   a=3;
   disp(a,"a=")
endfunction
```

1) Que fait cette procédure ?

2) Programmer celle-ci dans Scilab, initialiser **a**= 2, appeler **argumentvauttrois(a)** puis demander à scilab d'afficher la valeur de **a**. Quelle est cette valeur ?

Exemple 2 :

```
function a_vautdeux()
   a=2
   disp(a)
endfunction
```

Programmer cette nouvelle procédure dans Scilab, taper **clear a**, appeler **a_vautdeux()** puis demander à scilab d'afficher la valeur de **a**. Que se passe-t-il ?

Alors que dans le premier exemple Scilab affiche bien, **a=3**, **a** n'a pas changé de valeur et on a toujours **a=2** après avoir appelé cette procédure.
Ceci est dû au fait que la variable **a** de la fonction Scilab, est une variable locale ; c'est-à-dire que pour Scilab ce n'est pas la même variable que dans le programme principal même si elle porte le même nom.
Dans le second exemple, on voit que Scilab a bien créé la variable **a** le temps d'exécution de la fonction, celle-ci n'existe plus ensuite.

Essayer maintenant de taper la commande **global a** dans la console Scilab et d'ajouter la ligne **global a** dans la fonction **argumentvauttrois(a) :**

```
function argumentvauttrois(a)
   global(a) ;
   a=3;
   disp(a,"a=")
endfunction
```

Rappeler **argumentvauttrois(a)** puis demander à scilab d'afficher la valeur de **a**. La valeur de **a** est maintenant bien 3.
**a** est devenue une variable globale, c'est-à-dire qu'elle est partagée entre le programme principal et les fonctions l'utilisant.
Il est aussi possible de partager une variable entre plusieurs fonctions (par exemple lorsqu'une fonction fait appelle à une autre), mais pas avec le programme principal ou pas avec d'autres fonctions. Pour cela, il faudra écrire la commande **global a** uniquement dans les fonctions qui partagent cette variable.

# 4 Traitements des images avec SIVP

## 4.1 Introduction

SIVP (Scilab Image and Video Processing Toolbox) est un module qui a été développé afin de pouvoir créer et modifier des images et des vidéos avec Scilab.

Les deux commandes de bases de SIVP pour pouvoir manipuler des images dans Scilab est :

- pour lire une image : **imread('nom_de_l'image.jpg')** , en faisant bien attention que l'image se trouve dans le répertoire courant. Scilab peut lire plusieurs formats d'image (.png, .jpg, .bmp, …)
- pour afficher une image **: imshow(im),** im étant le nom de la matrice représentant l'image (*voir §4.3).*

**Attention** : L'espace mémoire occupé par les images étant très important, il est important d'augmenter la taille de la pile utilisée par Scilab en utilisant: **stacksize(3e7)**

## 4.2 Analogique et numérique

Lorsque l'on veut reproduire ou transmettre une information sur un phénomène naturel, par exemple enregistrer de la musique, prendre une photo, mesurer la température, etc…., il est possible de le faire soit sous forme analogique, soit sous forme numérique.
La forme analogique consiste à reproduire le phénomène naturel de façon analogue. Il s'agira par exemple d'un thermomètre à alcool dont le liquide va changer de volume selon la température et ainsi indiquer l'évolution de celle-ci. Les variations seront indiquées en continu, c'est-à-dire que si la température passe de 10°C à 12°C, le thermomètre indiquera toutes les températures entre les deux ; il y en a une infinité.
La forme numérique consiste à donner les informations avec des nombres. Il s'agira par exemple d'un thermomètre à affichage numérique où les températures seront indiquées de à 0,1°C près, on parle ici d'information discrète. Contrairement à une information continue, les différentes valeurs possibles seront toutes séparées. Dans notre exemple de la variation de T° entre 10°C et 12°C, la température indiquée sera une des 21 températures de ce tableau :

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 10,1 | 10,2 | 10,3 | 10,4 | 10,5 | 10,6 | 10,7 | 10,8 | 10,9 | 11 | 11,1 | 11,2 | 11,3 | 11,4 | 11,5 | 11,6 | 11,7 | 11,8 | 11,9 | 12 |

Pour plus d'informations :
http://fr.wikipedia.org/wiki/Analogique et http://fr.wikipedia.org/wiki/Num%C3%A9rique

## 4.3 Qu'est-ce une image numérique ?

Une image numérique est un fichier informatique contrairement à une image analogique (par exemple un dessin ou une photo prise par un appareil photo analogique) qui existeront sur un support physique tel qu'une pellicule, du papier …

Le terme "image numérique" regroupe tous types d'images (dessins, photos,…) stockés sous une forme binaire (suite de 1 et de 0) et lisibles par un ordinateur. Les images peuvent être numériques dès la source (appareil photo numérique, logiciel de dessin, etc….) ou avoir été d'abord créés sous forme analogique puis numérisées, par exemple avec un scanner.

Voici par exemple une photo numérique de la cour intérieure du lycée « les Pierres Vives » et un zoom sur la partie encadrée :

On peut voir que l'image est constituée de petits carrés, ayant chacun une couleur uniforme. Ces carrés s'appellent des pixels (picture elements).

Lorsque ces pixels sont suffisamment petits, notre œil ne les distingue plus et a l'impression de voir une image qui correspond à la réalité. Plus les pixels sont petits, et meilleure est la qualité de l'image.

A partir d'un certain seuil, les pixels sont suffisamment petits pour que notre œil s'en accommode. Ce seuil est une combinaison du nombre de pixel par unité de longueur et de la distance à laquelle on se trouve par rapport à l'image. Le nombre de pixels par unité de longueur (ppp : pixels par pouce ou ppi : pixel per inch) est la résolution de l'image.

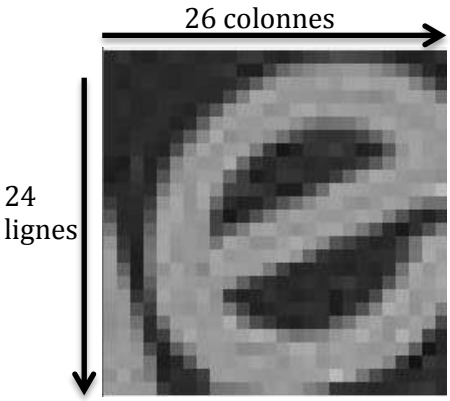
La définition de l'image est le nombre de pixels la constituant, dans le cas de notre image, elle est constituée de $167 \times 250$ pixels.

**<u>Exercice 4.3</u>** : Retrouver la résolution de l'image de la cour intérieure du lycée à partir de sa taille et de sa définition ( en supposant que l'imprimante, puis la photocopieuse ont bien imprimé tous les pixels, ce qui n'est en réalité pas le cas).

## 4.4 Représentation numérique d'une image

Une image numérique étant un rectangle constitué de petits carrés, on peut la représenter sous la forme d'un tableau, c'est-à-dire d'une matrice.

Voici par exemple une image en niveau de gris, le zoom sur un « e » de l'image et son codage matriciel :




26 colonnes

(1,1) (1,26)

24 lignes

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 42 | 36 | 43 | 46 | 41 | 39 | 42 | 44 | 41 | 44 | 58 | 93 | 135 | 155 | 152 | 152 | 160 | 152 | 164 | 165 | 157 | 159 | 159 | 131 | 94 | 50 |
| 42 | 44 | 40 | 41 | 45 | 46 | 44 | 42 | 43 | 73 | 123 | 164 | 164 | 151 | 149 | 149 | 146 | 144 | 147 | 142 | 136 | 146 | 162 | 162 | 150 | 114 |
| 42 | 46 | 39 | 40 | 47 | 42 | 37 | 57 | 88 | 145 | 159 | 158 | 143 | 145 | 163 | 166 | 153 | 161 | 158 | 156 | 155 | 151 | 148 | 150 | 155 | 159 |
| 42 | 40 | 41 | 46 | 45 | 35 | 44 | 95 | 150 | 160 | 153 | 151 | 157 | 157 | 143 | 123 | 110 | 113 | 118 | 137 | 159 | 164 | 153 | 146 | 148 | 159 |
| 42 | 37 | 46 | 46 | 40 | 50 | 89 | 137 | 168 | 141 | 148 | 153 | 142 | 112 | 78 | 57 | 51 | 50 | 55 | 70 | 99 | 134 | 157 | 160 | 153 | 149 |
| 42 | 41 | 48 | 40 | 36 | 77 | 139 | 163 | 147 | 155 | 158 | 123 | 59 | 23 | 34 | 51 | 52 | 61 | 50 | 33 | 40 | 88 | 145 | 163 | 151 | 153 |
| 33 | 29 | 59 | 35 | 55 | 130 | 153 | 147 | 150 | 169 | 106 | 50 | 40 | 47 | 44 | 41 | 45 | 42 | 46 | 59 | 89 | 127 | 155 | 161 | 155 | 149 |
| 30 | 36 | 54 | 36 | 82 | 142 | 149 | 157 | 155 | 103 | 60 | 30 | 38 | 53 | 51 | 44 | 43 | 61 | 100 | 145 | 167 | 162 | 151 | 146 | 148 | 141 |
| 41 | 42 | 47 | 43 | 118 | 157 | 145 | 166 | 142 | 67 | 61 | 55 | 46 | 31 | 31 | 67 | 109 | 161 | 161 | 158 | 153 | 149 | 150 | 158 | 166 | 185 |
| 60 | 42 | 43 | 58 | 147 | 165 | 144 | 162 | 103 | 44 | 34 | 35 | 55 | 83 | 110 | 137 | 157 | 160 | 147 | 141 | 151 | 161 | 155 | 133 | 114 | 113 |
| 84 | 39 | 41 | 78 | 160 | 162 | 147 | 151 | 65 | 26 | 45 | 80 | 120 | 149 | 160 | 157 | 151 | 149 | 155 | 163 | 157 | 126 | 82 | 49 | 35 | 50 |
| 117 | 43 | 39 | 98 | 163 | 154 | 151 | 144 | 69 | 90 | 125 | 159 | 164 | 151 | 143 | 151 | 162 | 164 | 148 | 121 | 84 | 43 | 26 | 48 | 80 | 124 |
| 147 | 53 | 35 | 112 | 164 | 148 | 152 | 145 | 117 | 163 | 155 | 143 | 143 | 158 | 169. | 157 | 135 | 84 | 66 | 47 | 37 | 39 | 63 | 115 | 164 | 163 |
| 158 | 63 | 31 | 119 | 165 | 145 | 151 | 150 | 164 | 149 | 153 | 156 | 160 | 165 | 149 | 101 | 53 | 43 | 40 | 42 | 46 | 48 | 68 | 119 | 169 | 155 |
| 157 | 97 | 31 | 114 | 145 | 161 | 147 | 132 | 162 | 159 | 156 | 153 | 123 | 69 | 39 | 39 | 42 | 39 | 35 | 42 | 37 | 48 | 114 | 164 | 148 | 160 |
| 155 | 125 | 34 | 95 | 156 | 158 | 153 | 150 | 151 | 142 | 97 | 61 | 46 | 38 | 42 | 47 | 41 | 51 | 57 | 45 | 41 | 83 | 138 | 159 | 154 | 166 |
| 150 | 148 | 51 | 76 | 165 | 151 | 149 | 160 | 143 | 54 | 35 | 38 | 50 | 43 | 36 | 41 | 44 | 32 | 44 | 39 | 74 | 149 | 169 | 148 | 151 | 158 |
| 145 | 151 | 82 | 60 | 153 | 149 | 144 | 154 | 147 | 119 | 67 | 35 | 39 | 44 | 47 | 50 | 47 | 30 | 50 | 80 | 131 | 172 | 162 | 145 | 161 | 122 |
| 142 | 148 | 113 | 43 | 111 | 156 | 157 | 150 | 153 | 171 | 119 | 77 | 56 | 38 | 33 | 43 | 48 | 92 | 116 | 155 | 163 | 145 | 149 | 159 | 147 | 73 |
| 145 | 153 | 134 | 42 | 63 | 144 | 166 | 154 | 156 | 158 | 161 | 174 | 165 | 131 | 113 | 122 | 131 | 151 | 159 | 173 | 156 | 143 | 174 | 154 | 68 | 41 |
| 151 | 155 | 145 | 75 | 35 | 93 | 133 | 153 | 154 | 149 | 138 | 139 | 145 | 153 | 168 | 170 | 151 | 161 | 157 | 151 | 147 | 153 | 148 | 93 | 22 | 40 |
| 156 | 151 | 152 | 117 | 30 | 39 | 85 | 146 | 154 | 150 | 156 | 163 | 155 | 145 | 156 | 162 | 148 | 153 | 158 | 153 | 155 | 133 | 58 | 21 | 56 | 50 |
| 143 | 161 | 139 | 121 | 35 | 46 | 44 | 61 | 154 | 157 | 154 | 150 | 149 | 151 | 154 | 157 | 159 | 186 | 160 | 118 | 77 | 49 | 38 | 40 | 45 | 43 |
| 154 | 151 | 150 | 148 | 57 | 42 | 42 | 53 | 114 | 120 | 126 | 137 | 150 | 155 | 147 | 129 | 114 | 53 | 49 | 43 | 39 | 40 | 45 | 52 | 57 | 43 |

(24,1) (24,26)

Les différents niveaux de gris sont codés entre 0 et 255, le 0 codant le noir et le 255 codant le blanc. Plus le numéro codant le niveau est grand, plus le pixel est clair.
Notre bout d'image contient des pixels codés entre 21 et 186

**Exercices 4.4** : 1) Créer la matrice identité A de 100 lignes et 100 colonnes avec la commande **eye**. Puis taper la commande : **Imshow(A).** Quelle type d'image obtient-on, est-ce une image en niveau de gris ?

2) a)Créer une image représentant un carré blanc sur fond noir, puis un carré noir sur fond blanc.
   b) Créer une fonction permettant de créer un rectangle noir sur une image, dont les dimensions et la position seront données en arguments
   c) Créer une fonction permettant d'afficher un rectangle noir sur une image, dont les dimensions et la position seront données en arguments, sans la transformer. Utiliser celle-ci pour faire parcourir une image de haut en bas par un rectangle noir.

3) Créer l'image d'un cercle blanc sur fond noir, puis d'un disque noir sur fond blanc.

## 4.5 Images en niveaux de gris

Créer la matrice A ci-dessous :

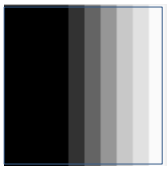
```
A  =

    0.    0.    0.    0.    50.    100.    150.    200.    225.    250.
    0.    0.    0.    0.    50.    100.    150.    200.    225.    250.
    0.    0.    0.    0.    50.    100.    150.    200.    225.    250.
    0.    0.    0.    0.    50.    100.    150.    200.    225.    250.
    0.    0.    0.    0.    50.    100.    150.    200.    225.    250.
    0.    0.    0.    0.    50.    100.    150.    200.    225.    250.
    0.    0.    0.    0.    50.    100.    150.    200.    225.    250.
    0.    0.    0.    0.    50.    100.    150.    200.    225.    250.
    0.    0.    0.    0.    50.    100.    150.    200.    225.    250.
    0.    0.    0.    0.    50.    100.    150.    200.    225.    250.
```

Puis demander à Scilab de l'afficher avec la commande : **Imshow(A)**
Vous pouvez constater que Scilab n'affiche aucune nuance, il s'agit d'une image en noir et blanc. Pour pouvoir afficher les nuances, il faut que la matrice soit une matrice d'entiers que l'on obtient avec la commande **uint8**

imshow(A) :

imshow(uint8(A)) :



uint8(A) est la matrice :

```
-->uint8(A)
 ans  =

  0   0   0   0   50   100   150   200   225   250
  0   0   0   0   50   100   150   200   225   250
  0   0   0   0   50   100   150   200   225   250
  0   0   0   0   50   100   150   200   225   250
  0   0   0   0   50   100   150   200   225   250
  0   0   0   0   50   100   150   200   225   250
  0   0   0   0   50   100   150   200   225   250
  0   0   0   0   50   100   150   200   225   250
  0   0   0   0   50   100   150   200   225   250
  0   0   0   0   50   100   150   200   225   250
```

**<u>Exercice 4.5 :</u>** Créer une mire de définition 50 × 256, en niveaux de gris.

## 4.6 Images en couleurs

Dans Scilab, les images sont des images RVB (Rouge Vert Bleu) ou RGB (Red Green Blue). Ce système utilise le fait qu'il suffit de ces trois couleurs pour créer toutes les autres par synthèse additive.

Le codage d'une image en couleur par Scilab se fait avec une matrice en trois dimensions : le nombre de lignes × le nombre de colonnes × 3 (pour les 3 couleurs Rouge, Vert et Bleu).

On peut imaginer que l'on a trois images de deux dimensions (hauteur et largeur), comme celles que l'on a obtenues en niveau de gris, mais l'une est en niveaux de rouge, la seconde en niveaux de vert et la troisième en niveaux de bleu. La superposition des trois permet le rendu de toutes les couleurs (en réalité seulement 256 × 256 × 256 couleurs, soit environ 16 millions).

**Exemple** : a) lancer la suite de commandes suivantes :

```
A=zeros(50,50,3);
A(10:30,10:30,1)=1;
A(10:30,20:40,2)=1;
A(20:40,15:35,3)=1;
imshow(A)
```

b) Expliquer chacune des étapes de cette suite de commandes.

Dans cet exemple, on peut voir les couleurs secondaires dans la synthèse additive :

- Cyan, complémentaire du rouge = lumières verte et bleue
- Magenta, complémentaire du vert=lumières rouge et bleue
- Jaune, complémentaire du bleu = lumières verte et rouge

Mais ici, nous n'avons aucune nuance, il s'agit d'une image en rouge, vert et bleu (comme une image noir et blanc) et non pas en niveau de rouge, niveau de vert et niveau de bleu (comme une image en niveau de gris).

Pour pouvoir obtenir des nuances, il faudra aussi utiliser la commande **uint8.**

**Exercice 4.6 :** 1) Créer une fonction permettant de créer l'image d'un rectangle dans une des trois couleurs fondamentales sur fond noir, avec les dimensions, la position et la couleur données en argument.
2) Créer l'image d'un carré rose sur fond cyan
3) Créer l'image d'un disque vert sur fond magenta
4) Créer une fonction permettant d'ajouter un cercle avec pour arguments, la couleur, le rayon et la position.
5) Créer une fonction permettant de tracer une ligne entre deux pixels avec pour arguments, la couleur et la position de chacune des extrémités.

# Correction des exercices

**Exercice 2.2 :**

```
-->M=[11:15;21:25;31:35]
 M  =

    11.    12.    13.    14.    15.
    21.    22.    23.    24.    25.
    31.    32.    33.    34.    35.

-->M(1,1)=2
 M  =

    2.     12.    13.    14.    15.
    21.    22.    23.    24.    25.
    31.    32.    33.    34.    35.

-->M(3,:)=1
 M  =

    2.     12.    13.    14.    15.
    21.    22.    23.    24.    25.
    1.     1.     1.     1.     1.

-->M(:,5)=0
 M  =

    2.     12.    13.    14.    0.
    21.    22.    23.    24.    0.
    1.     1.     1.     1.     0.
```

**Exercice 2.4 :**

```
-->M0=zeros(10,15); Mi=eye(5,5);M1=ones(5,5);

-->M0(1:5,1:5)=Mi; M0(6:10,6:10)=M1; M0(1:5,11:15)=2*Mi
 M0  =

    1.     0.     0.     0.     0.     0.     0.     0.     0.     0.     2.     0.     0.     0.     0.
    0.     1.     0.     0.     0.     0.     0.     0.     0.     0.     0.     2.     0.     0.     0.
    0.     0.     1.     0.     0.     0.     0.     0.     0.     0.     0.     0.     2.     0.     0.
    0.     0.     0.     1.     0.     0.     0.     0.     0.     0.     0.     0.     0.     2.     0.
    0.     0.     0.     0.     1.     0.     0.     0.     0.     0.     0.     0.     0.     0.     2.
    0.     0.     0.     0.     0.     1.     1.     1.     1.     1.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.     1.     1.     1.     1.     1.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.     1.     1.     1.     1.     1.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.     1.     1.     1.     1.     1.     0.     0.     0.     0.     0.
    0.     0.     0.     0.     0.     1.     1.     1.     1.     1.     0.     0.     0.     0.     0.
```

**Exercices 3.2.2**:

| | valeur variable a | test de condition | instruction |
|---|---|---|---|
| Initialisation | 6 | | |
| a=6 ? | | vrai | « gagné » |
| FinSi | | | |

| | valeur variable a | test de condition | instruction |
|---|---|---|---|
| Initialisation | 2 | | |
| a=6 ? | | **faux** | **« perdu »** |
| FinSi | | | |

```
a=grand(1,1,'uin',1,6)
b=grand(1,1,'uin',1,6)
disp(b,a,"vos faces sont :")
if a==b then
   S=2*a;
   disp("€",S,"vous avez gagné :")
elseif (a==6 | b==6) then
   S=a+b-6;
   disp("€",S,"vous avez gagné :")
else disp("vous avez perdu");
end
```

**Exercice 3.2.2.b**:

```
a=grand(1,1,'uin',1,6)
b=grand(1,1,'uin',1,6)
disp(b,a,"vos faces sont :")
P=a+b;
select P
   case 2*a disp("€",P,"vous avez gagné :")
   case a+6 disp("€",a,"vous avez gagné :")
   case b+6 disp("€",b,"vous avez gagné :")
   else disp("vous avez perdu")
end
```

**Exercice 3.2.3** :

```
S=0
for i=1:10
S=S-3
G=0;
a=grand(1,1,'uin',1,6)
b=grand(1,1,'uin',1,6)
disp(b,a,"vos faces sont :")
if a==b then
   G=2*a;
  disp("€",G,"vous avez gagné :")
elseif (a==6 | b==6) then
   G=a+b-6;
   disp("€",G,"vous avez gagné :")
else disp("vous avez perdu");
end
S=S+G
end
if S<0 then disp("€",S,"Finalement, vous avez perdu :")
   else disp("€",S,"Finalement, vous avez gagné :")
end
```

**Exercice 3.2.3.b** :

```
S=0;
k=0;
while S<5 & S>-20
   S=S-3
   G=0;
   a=grand(1,1,'uin',1,6)
   b=grand(1,1,'uin',1,6)
   if a==b then
      G=2*a;
      elseif (a==6 | b==6) then
      G=a+b-6;
   end
   S=S+G;
   k=k+1
end
if S>=5 then disp("parties",k,"vous avez gagné plus de 5 € en")
else disp("parties",k,"vous avez perdu plus de 20 € en")
end
```

**Exercices 4.4** : a)

```
-->Image=zeros(100,100);
-->Image(20:50,20:50)=1;
-->imshow(Image)
```

```
-->Image=ones(100,100);

-->Image(20:50,20:50)=0;

-->imshow(Image)
```

b) La fonction est définie par :

```
function A=rectangle(A, a, b, l, L)
   A(a:a+l,b:b+L)=0;
endfunction
```

Par exemple, si on veut incruster dans l'image Im un rectangle noir $10 \times 20$ dont le coin haut à gauche est sur le pixel (30,15) : **Im=rectangle(Im,30,15,10,20)**

c) La fonction est définie par :

```
function rectangle(A, a, b, l, L)
   A(a:a+l,b:b+L)=0;
   imshow(A);
endfunction
```

Par exemple, pour faire parcourir un rectangle noir $5 \times 10$, de haut en bas sur la colonne 20 (gauche du rectangle) :

```
for i=1:100
   rectangle(Im,i,20,5,10)
end
```

3) Pour créer un disque noir sur une image, de centre (a,b) et de rayon c :

```
function cercle(A, a, b, c)
   for i=1:200
      for j=1:200
         if (i-a)^2+(j-b)^2<c^2 then A(i,j)=0
         end
      end
   end
   imshow(A);
endfunction
```

**Exercice 4.5 :**

```
for i=1:256
   Mire(1:50,i)=i;
end
Mire=uint8(Mire)
imshow(Mire)
```

**Exercice 4.6 :**

```
function A=rectanglecouleur(A, a, b, l, L, c)
   if c="rouge" then c=1
      elseif c="vert" then c=2
      elseif c="bleu" then c=3
   end
   A(a:a+l,b:b+L,c)=1;
   imshow(A);
endfunction
```

```
function A=segment(A, a, b, c, d, couleur)
   if couleur="rouge" then couleur=1
      elseif couleur="vert" then couleur=2
      elseif couleur="bleu" then couleur=3
   end
   if b=d then A(a:c,b,couleur)=1;
   else
      for j=b:d
         i=uint8(a+((c-a)/(d-b))*j);
         A(i,j,couleur)=1;
      end
   end
endfunction
```