Introduction

Il existe de nombreuses introductions à l'environnement et au langage SciLab, mais elles ne sont pas assez directe à mon goût. Les explications qui suivent ont pour but d'être minimaliste et très synthétiques. Si elles ne vous conviennent pas, référez-vous aux documents référencés sur le site Web : http://www.juggling.ch/gisin/scilab/
Une fois que vous serez initiez à SciLab, le document de référence suivant est conseillé :
http://www.juggling.ch/gisin/scilab/docs/Resume_condense_Scilab.pdf idem en .odt

SciLab est un proche cousin de MatLab, "Mat" venant de "Matrices". C'est un environnement de **programmation scientifique**, basé sur le **traitement matriciel** et vectoriel.

Première prise en main de SciLab

Exécutez le logiciel SciLab. La "Console Scilab x.x.x" s'ouvre.
SciLab peut être considéré comme une super-calculatrice.
Tapez : "`355 / 113`", sans les guillemets. Le résultat s'affiche.
Tapez : "`(ans - %pi) / %pi`", un autre résultat affiche l'erreur relative entre 355/113 et pi.
Tapez : `for i=1:10; disp(string(i) + "  " + string(i^2)); end;`

Certaines notions de bases sont communes à tous les langages de programmation. Nous allons les décrire ci-dessous, ainsi que leur syntaxe dans le langage SciLab.

Les variables

Il est indispensable de stocker des nombres, des chaînes de caractères (string) et divers structures dans des zones mémoire. Pour cela les **variables** sont utilisées. On leur donne un nom formé d'une lettre, suivit de 0, 1 ou plusieurs autres lettres, chiffres et du caractère _
On assigne une valeur à une variable à l'aide du symbole =
Tapez : `a = 5;` // le ; en fin de ligne indique de ne pas afficher de résultat. C'est un **inhibiteur d'affichage.**
Ce qui suit le // est un **commentaire**. Il est indispensable de **commenter vos programmes** !
Tapez : `disp(a); a = a + 1; disp(a);` // La valeur de la variable a change.
Tapez : `approx_e = 2 + 1/2 + 1/(2*3) + 1/(2*3*4) + 1/(2*3*4*5) + 1/(2*3*4*5*6)`
Tapez : `( approx_e - %e) / %e` // approx_e est donc une approximation du nombre d'Euler e

La boucle for

La "**boucle for**" est faite pour des calculs répétitifs. Tapez :
`s=0; for k=1:10 do s=s+1/k; end; disp(s); // au lieu de 10 on peut taper 1000 ou ...`

Exercice 1.1 :
La série : 1 + 1/2 + 1/3 + ... + 1/.. converge-t-elle ou diverge-t-elle ?
Jouez avec les instructions ci-dessus, pour chercher une réponse.

Exercice 1.2 :
Calculez la somme : $1 + 1/2^2 + 1/3^2 + ... + 1/100000^2$
Vérifiez qu'elle se rapproche de $\frac{\pi^2}{6}$ *qui se note %pi^2 / 6;*

Exercice 1.3 :
Écrivez un code qui permet de calculer $n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot ... \cdot n$, n étant donné

L'éditeur SciNotes

Rapidement, on désire mémoriser dans un fichier texte les instructions que l'on a tapé. Pour cela, lancez l'éditeur intégré **SciNotes** depuis le menu "Application -> SciNotes"
Dans l'éditeur, tapez :

```
exposant = 3;
somme=0;
for k=1:10^exposant do
   somme=somme+1/k;
end;
disp(somme);
```

Il est important de taper les instructions les unes à la suite des autres, pour la lisibilité !

Tapez ensuite sur la touche F5. Il vous sera demandé sous quel nom de fichier vous désirez sauver votre programme. Tapez le nom "Initiation_scilab.sce" et créer un **dossier Scilab** dans lequel vous sauverez vos productions.
Constatez que le résultat de votre somme est affichée dans la console.
!!! Dans la première ligne de votre texte dans SciNotes, tapez // suivit d'un **commentaire** intelligent indiquant à quoi correspond ce programme. Ce commentaire pourra évoluer !

Par la suite, sauf quelques rares exceptions, on tapera le code dans l'éditeur SciNotes.

Placez deux points .. en fin de ligne pour **continuer une instruction à la ligne suivante**.

Exercice 2.1 :
Le calcul : `approx_e = 2 + 1/2 + 1/(2*3) + 1/(2*3*4) + 1/(2*3*4*5) + 1/(2*3*4*5*6)`
*est répétitif, donc la **"boucle for"** est faite pour cela.*
L'intérêt est d'aller bien plus loin que le calcul de 6 termes.
Écrivez dans SciNotes le code qui calcule en même temps fact = k! et la somme
approx_e = 2 + 1/2! + 1/3! + 1/4! + ... + + 1/n! n étant donné. Exemples n = 6 ou n = 10.

Exercice 2.2 :
Écrivez dans SciNotes le code qui calcule en même temps fact = k! et la somme
approx_e_x = 1 + x + x^2/2! + x^3/3! + x^4/4! + ... + + x^n1/n! n et x étant donnés.
Exemples n = 10, x = 1 ou x = 2 ou x = log(10)

Les noms préexistants, annuler leur redéfinition

Dans la console :
tapez : `sum(1:10)` // le résultat de la somme 1+2+3+...+10 s'affiche.
Tapez : `sum=0`
Constatez le message : "`Redéfinition de la fonction : sum ...`"
Retapez : `sum(1:10)` ou revenez sur cette instruction à l'aide de la flèche vers le haut.
Constatez le message : "`Index invalide.`"
On a détruit par inadvertance la définition de la fonction sum.
Pour annuler cette redéfinition et toutes les définitions de variables, tapez : **`clear`**

Retapez : `sum(1:10)` // et constatez que la fonction sum fonctionne de nouveau correctement.
`clc` // efface le contenu de la console.

La boucle while

Notre programme de l'exercice 1.1, utilisant la boucle for, permet de calculer la somme des inverses des nombres entiers positifs, de 1 à une valeur au choix.
Question : *jusqu'à quelle valeur faut-il effectuer la somme pour obtenir le nombre le plus proche possible de 10, qui dépasse ?*
Par différents essais, on peut y arriver, mais ce n'est pas pratique.
C'est ici que la "**boucle while**" intervient.
Dans SciNotes, tapez :

```
somme=0;
k=1;
while (somme < 10) do
  somme = somme + 1/k;
  k = k+1;
end;
disp(somme, k);
```

Exécuter votre programme en pressant la touche F5.
La réponse à la question est affichée.

Exercice 3.1 :
Jusqu'à quelle nombre faut-il effectuer la somme $1 + 1/2^2 + 1/3^2 + 1/4^2 + 1/5^2 + ...$ pour obtenir une approximation du nombre %pi^2 / 6 avec une erreur absolue inférieure à 10^{-4} ?
Écrivez un commentaire avant votre programme !
Si votre programme ne s'arrête pas, il **plante**. Pour l'interrompre, dans le menu, cliquez sur "Contrôle -> Abandonner".
Vous pouvez faire plusieurs tests de votre programme.
Il est normal qu'il ne fonctionne pas du premier coup.

Exercice 3.2 :
Jusqu'à quelle nombre faut-il effectuer la somme $2 + 1/2 + 1/3! + 1/4! + 1/5! + ...$ pour obtenir une approximation du nombre %e avec une erreur absolue inférieure à 10^{-10} ?
Écrivez un commentaire avant votre programme !

L'écriture vectorielle et matricielle

Dans la console :
Tapez : `v1=1:10;  v2=1 ./ v1;` // attention à l'espace entre 1 et ./
Tapez : `v1`
Tapez : `v2`
`v1=1:10;` défini une variable contenant un vecteur ligne, c'est-à-dire un tableau de nombres placés sur une ligne.
En tapant : `v1` vous affichez les valeurs du tableau
En tapant : `v2` vous affichez les inverses des valeurs du tableau.
Quand cela est possible, utilisez l'écriture vectorielle de SciLab. Les calculs ainsi effectués sont beaucoup plus performants lorsque cette écriture est utilisée.
Une **matrice** est un tableau de nombres. Voici un petit exemple.
Tapez : `A = [1, 2, 3; 1, 4, 9; 7, 8, 9]`
Tapez : `B = 1 ./ A` // les coefficients de B valent les inverses des coefficients de A
Tapez : `B .* A` // tous les coefficients valent 1
Tapez : `C = 1 / A` // C est la matrice inverse de A, l'opérateur / est très différent de ./
Tapez : `C*A` // le résultat est presque la matrice identité.
Petit à petit vous apprendrez beaucoup plus sur les matrices.
Tapez : `for i=1:3; for j=1:3; disp(A(i,j)); end; end;`
Tapez : `for i=1:9; disp(A(i)); end;`
`A(2,3) = 0` // modifie un coefficient de la matrice

Le test if ... then ... elseif ... then ... else ... end

Souvent, il faut tester si une condition est satisfaite ou non et le programme continuera différemment suivant le résultat du test. C'est l'utilité du "**test if then else**".

Dans SciNotes, tapez :

```
// Test si un nombre est positif, nul ou négatif
n=7;  // nombre que l'on peut changer
if (n > 0) then
   disp(string(n) + " est un nombre positif");
elseif (n == 0) then
   disp("Le nombre n = 0");
else
   disp(string(n) + " est un nombre négatif");
end; // if
```

Autre exemple :

```
// Test si un nombre est pair ou non
// Si on assigne un nombre non entier à  n,  le teste sera faux
n = 7;  // nombre que l'on peut changer
if (modulo(n, 2) == 0) then  // modulo(n, 2)  donne le reste de la division de  n  par  2
   // n  le reste de la division de  n  par  2  est nul, donc  n  est pair.
   disp(string(n) + " est un nombre pair");
else
   disp(string(n) + " est un nombre impair");
end; // if
```

Exercice 4.1 :

Un nombre était donné, indiquez si ce nombre est divisible par 3.

Exemple :

Le problème de **Syracuse** est le suivant :
On part d'un nombre entier positif.
S'il est pair, on le remplace par sa moitié.
S'il est impair, on le remplace par son triple additionné de 1.
On recommence.
Malgré la simplicité de cet algorithme, on ne sait toujours pas en 2015 si le nombre 1 arrive dans la succession de nombres obtenus, quel que soit le nombre de départ. On sait que c'est le cas pour tous les nombres entre 1 et 10^{20}.
La programmation de ce problème nécessite de faire une boucle while et un test if.
Dans SciNotes, tapez :

```
// Le problème de Syracuse, liste de la succesion de nombres
n=7;
while (n > 1)
  disp(n);
  if (modulo(n, 2) == 0)  // modulo(n, 2)  donne le reste de la division de  n  par  2
    n = n / 2;
  else
    n = 3*n + 1;
  end; // if
end; // while
```

Exercice 4.2 :

a, b, c étant donnés, déterminez si le polynôme $ax^2 + bx + c$ possède des racines et indiquez-les.
sqrt(expr) retourne la racine carrée de expr.
.. à la fin d'une ligne permet de la continuer sur la ligne suivante.

Les fonctions

Souvent on désire :
° réutiliser un bout de programme plusieurs fois ou
° isoler un bout de programme ou
° définir une fonction qui retourne un résultat en fonction de données.

Dans SciNotes, tapez :

```
function sols=racinepol2(a, b, c)
//==============================
// résolution d'une équation du second degré
delta=b^2 - 4*a*c;
if (delta < 0) then
  sols = [];  // pas de solutions
elseif (delta == 0) then
  sols = [-b/(2*a)]; // une seule solution
else
  sols = [(-b - sqrt(delta))/(2*a), (-b + sqrt(delta))/(2*a)]; // deux sol
end; // if
end function;
```

Dans la console, tapez : `racinepol2(6, 5, 1) //` les deux racines sont retournées
Tapez : `mes_sols = racinepol2(1, 6, 9); length(mes_sols)` // donne le nombre de solutions
`mes_sols` // affiche le vecteur des solutions.
Cette fonction donne la solution à l'exercice 4.2

Exercice 5.1 :
Écrivez une fonction "f=fact(n)" qui pour un entier n donné, retourne n!

Exercice 5.2 :
Écrivez une fonction "sol=pmax2(n)" qui pour un entier n donné, retourne le plus grand exposant p telle que 2^p divise n.

Exercice 5.3 :
*Écrivez une fonction "sols=**Syracuse**(n)" qui pour un nombre n entier donné, retourne la liste des valeurs obtenus dans le problème de Syracuse.*
Si `v1` *et* `v2` *sont deux vecteurs,* `[v1, v2]` *est le vecteur ligne composé des deux vecteurs* `v1` *et* `v2`.

Si v1 est un vecteur et fct une fonction, fct(v1) est le vecteur des images par fct des coef. de v1.

Vous pouvez tester les fonctions depuis la console en tapant leur nom avec leur paramètre.
Exemples :
fact(7) // retournera 7!
pmax2(24) // retournera 3
Syracuse(14) // retournera la suite de Syracuse du nombre 14.

Les opérateurs

Les opérateurs de bases sur des nombres sont : `+ - * / ^` // ^ pour la mise à la puissance
Sur les **matrices** : + -
`.*` // multiplication coefficient par coefficient
`./` // division coefficient par coefficient
`.^` // mise à la puissance
`*` // multiplication matricielle
`/` // inversion matricielle
`\` // résolution matricielle d'équation
Sur les **string** :
`+` // concaténation de strings. "ab" + "cde" donne "abcde"
Pour obtenir des **boolean**, qui sont des résultats de tests logique
`==` // test d'égalité
`> < >= <=` // test d'égalité
`~=` // test de différence
`<>` // test de différence
Sur les **boolean**.
`a | b` // a or b a ou b, faux si et seulement si a et b sont faux
`a & b` // a and b a et b, vraie si et seulement si a et b sont vrais.
`~a` // not(a) ; négation de a.

`..` *à la fin d'une ligne permet de la continuer sur la ligne suivante.*

Quelques fonctions de base

`sqrt(nbr)`	// la racine carrée
`modulo(n, m)`	// reste de la division de n par m
`abs(nbr)`	// valeur absolue du nombre
`floor(nbr)`	// partie entière du nombre = plus grand nombre entier <= nbr
`round(nbr)`	// arrondi un nombre à l'entier le plus proche
`sin(nbr)`	// le sinus. Existe aussi cos tan asin acos atan
`exp(nbr)`	// exponentielle de nbr = e^{nbr}
`log(nbr)`	// logarithme naturel. log10(nbr) donne le logarithme en base 10
`sum(vec)`	// somme des coefficients du vecteur vec
`rand()`	// retourne un nombre aléatoire entre 0 et 1, bornes non comprise.
`string(nbr)`	// converti le nombre en string
`disp(nbr, str)`	// affiche un texte dans la console
`mprintf(...)`	// affiche un texte dans la console, en contrôlant le format d'affichage
`plot(x, y)`	// trace un graphique, c.f. le programme : http://www.juggling.ch/gisin/scilab/Plot_param.sce

Référez-vous à Resume_condense_Scilab.pdf et **l'aide de SciLab** pour plus d'information.

Avec de très nombreuses fonctions pré-définies, les **commentaires**, les **variables**, les **opérateurs**, les **boucles for** et **while**, le **test if** et les **fonctions** sont la base de tout langage de programmation.

En SciLab il existe également un test **select ... case ... else ... end**
Parfois il existe d'autres types de boucle, tels que **do ... while ...**
Dans certaine langages on peut définir des **structures**, des **objets** et des **opérateurs** sur ces objets.