==========
Version 1 : Date: 15/7/2007
Version 2 : Date: 28/7/2007 : ajout des chapitres 4 à 11.
Version 3 : Date: 05/9/2009 : ajout chapitre 24.
==========

==========
**** Cours de script *****
==========

Ce cours va vous apprendre à écrire des scripts simples pour le chat.

Chapitre 1 : premier script

Pour créer un script, ouvrez le bloc-notes (notepad) et écrivez ceci :

```
// mon premier script


proc timer ()
{
  print ("Salut les amis !");
}
```

Ensuite sauvez sous un nom qui se termine par script.txt,
par exemple mon-script.txt, et placez le fichier dans le répertoire
du bot, de préférence dans le sous-répertoire script.

Quand c'est fait, cliquez sur la fenêtre de chat du bot et appuyez sur F5.
Le script est alors chargé et compilé (transformé en code exécutable),
ce qui va faire apparaitre les messages suivants :

```
  system info: loading script files : ok
  system info: compiling mon-script.txt ...
```

Quand la compilation s'est bien passée, le script s'exécute alors et
affiche sur le chat :

```
  Salut les amis !
```

Voilà, vous avez écrit votre premier script !

Chapitre 2 : les erreurs

Durant la compilation du script, celui-ci est entièrement vérifié
et contrôlé, ce qui peut faire apparaitre des erreurs de syntaxe.

Par exemple si vous aviez oublié l'accolade finale } à la fin,
vous auriez eu, lors de la compilation, un message d'erreur
qui vous indique exactement où le problème se situe :

```
ERROR: instruction expected
AT LINE: 8 COLUMN: 1
```

```
GENERATED AT: 15/07/2007 13:42:00

     1 |
     2 |
     3 | proc timer ()
     4 | {
     5 |   print ("Salut les amis !");
     6 |
     7 |
---------^
***ERROR: instruction expected
```

Il suffit alors de corriger le fichier et de ré-appuyer sur F5.

Chapitre 3 : style

Pour éviter les erreurs de syntaxe et aussi faciliter la lecture,
il est important d'avoir un style d'écriture clair.

Exemple :

```
// mon 2ème script

proc timer ()
{
  var i, j, l$, e;

  for (j=0; j<32; j++)
  {
    l$ = "";
    for (i=0; i<25; i++)
    {
      e = j*25 + i;
      if (e == 416)
        break;
      l$ = l$ + chr$(0, e % 256, e / 256, 0);
    }
    print (l$);
    if (e == 416)
      break;
  }
}
```

Vous voyez, dans l'exemple ci-dessus, que les lignes
entres accolades { } sont à chaque fois décalées de
deux blancs à droite. En plus, une accolade {
se trouve toujours à la verticale d'une accolade }.
C'est une habitude que vous devez absolument prendre,
cela vous évite d'oublier des accolades !

Le script suivant fonctionne aussi mais est tout à fait
déconseillé parce qu'on n'y comprend plus rien !

Exemple :

```
// mon 3ème script (déconseillé)
```

```
proc timer ()
{
var i, j, l$, e;

for (j=0; j<32; j++)
{
l$ = "";
for (i=0; i<25; i++)
{
e = j*25 + i;
if (e == 416)
break;
l$ = l$ + chr$(0, e % 256, e / 256, 0);
}
print (l$);
if (e == 416)
break;
}
}
```

Quand vous postez un script sur un groupe ou un forum,
changez le font en "Courier New" car avec ce font
les décalages sont conservés.

l'importance des commentaires

N'hésitez pas, pour clarifier un script,
à ajouter plein de lignes de commentaires
n'importe où dans votre script. Il suffit
pour cela de commencer une ligne par //

Ces lignes sont ignorées par la compilation.

Exemple:

```
// voici mon premier script
// que j'ai créé pour animer
// un peu ma salle.

// Copyright le scripteur

proc timer ()
{
  // la commande print va afficher un texte sur le chat !
  print ("Salut les amis !");
}
```

Chapitre 4 : les variables numériques

Une variable est une case de la mémoire RAM du PC à laquelle on donne un nom.
Chaque variable contient une valeur numérique entre -2147483648 et +2147483647.
Celle valeur change généralement au fur et à mesure que le programme se déroule.

Pour créer une variable, il faut la 'déclarer', c'est-à-dire réserver
de la place mémoire pour son contenu, à l'aide de la commande 'var'.

exemple:

```
  var compteur;    // je crée mon compteur qui contient zéro au début
```

A leur création, les variables reçoivent toujours la valeur zéro.

Pour changer la valeur d'une variable, on écrit une 'assignation'
qui ressemble à ceci :

exemple:

```
  compteur = 2;   // maintenant, compteur a la valeur 2
```

Pour changer la valeur d'une variable, il suffit de lui donner
une nouvelle valeur :

exemple:

```
  compteur = 5;   // maintenant, compteur a la valeur 5
```

On peut faire des calculs aussi. Dans les exemples suivants, on peut voir :
- à droite du signe égal la valeur à calculer,
- à gauche du signe égal la variable où sera stocké le résultat du calcul.

exemples:

```
  compteur = 2 + 5*7;      // maintenant, compteur a la valeur 37
  compteur = (5 + 4) * 7;  // maintenant, compteur a la valeur 49
  compteur = -45;          // maintenant, compteur a la valeur -45
  compteur = 100 - 101;    // maintenant, compteur a la valeur -1
```

Voici comment augmenter la valeur d'une variable de 1 :

exemple:

```
  compteur = compteur + 1;     // calcule compteur + 1 et stocke le résultat
dans compteur
```

Il existe cependant une manière abbréviée de faire la même chose,
l'incrémentation :

exemple:

```
  compteur++;     // augmente compteur de 1
```

De la même façon, pour diminuer une variable de 1, on fait une décrémentation.

exemple:

```
  compteur--;     // diminue compteur de 1
```

Voici comment augmenter la valeur d'une variable de 10 :

exemple:

```
  compteur = compteur + 10;     // calcule compteur + 10 et stocke le résultat
dans compteur
```

Il existe cependant une manière abbréviée de faire la même chose, l'assignation avec addition :

exemple:

```
  compteur += 10;     // augmente compteur de 10
```

De la même façon, pour diminuer une variable de 10 :

exemple:

```
  compteur -= 10;     // diminue compteur de 10
```

Chapitre 5 : les variables chaines de caractères

Il existe une 2ème sorte de variable : les variables chaines de caractères, également appelées 'string' en anglais.

Celles-ci ne contiennent pas de nombres mais des chaines de caractères (lettres, chiffres, ponctuation, etc ..).

On reconnait une variable chaine de caractère au signe dollar ($) qui est ajouté à la fin de son nom.

exemple:

```
  var nom$;        // je crée une variable string
```

Pour changer la valeur d'une variable string, on fait une 'assignation' qui ressemble à ceci :

exemple:

```
  nom$ = "Bonjour";     // maintenant, nom$ a la valeur "Bonjour"
```

Pour changer la valeur d'une variable, il suffit de lui donner
une nouvelle valeur :

exemple:

```
  nom$ = "Bonsoir";      // maintenant, nom$ a la valeur "Bonsoir"
```

On peut coller plusieurs chaines ensembles avec l'opérator "+".

exemple:

```
  nom$ = "Bon" + "soir";       // maintenant, nom$ a la valeur "Bonsoir"
  nom$ = nom$ + " mes amis";   // maintenant, nom$ a la valeur "Bonsoir mes
amis"
```

Attention: la longueur d'une variable string ne peut pas dépasser 8192
caractères.

Lors de leur création, les variables string contienent une chaine vide
(donc de 0 caractères), ce qui est équivalent à l'exemple suivant.

exemple:

```
  nom$ = "";       // maintenant, nom$ est vide
```

Chapitre 6 : conversions

On peut convertir une variable numérique en variable string à l'aide
de la fonction str$()

exemple:

```
  var compteur, nom$;

  compteur = 12;

  nom$ = str$(compteur);    // maintenant, nom$ contient "12"

  nom$ = "compteur = " + str$(compteur);    // maintenant, nom$ contient
"compteur = 12"
```

A l'inverse, on peut convertir une variable string en variable numérique
à l'aide de la fonction val()

exemple:

```
  var compteur, nom$;

  nom$ = "12";

  compteur = val (nom$);      // maintenant, compteur égale 12

  compteur = val ("13");      //  maintenant, compteur égale 13

  compteur = val ("ab");      //  ceci va donner une erreur qui arrête le
script,
                              //  car "ab" ne peut être converti en valeur
numérique !
```

Chapitre 7 : affichages à l'écran

L'affichage à l'écran du chat se fait à l'aide de la procédure print()

exemple:

```
  print ("Bonjour !");
  print (nom$);
```

Il n'est pas permis de donner à print() une variable numérique, par exemple ceci ne va pas fonctionner :

exemple:

```
  print (compteur);         // erreur
```

Pour afficher la valeur de compteur, il faut le convertir d'abord en string :

exemple:

```
  print (str$(compteur));
  print ("la valeur de compteur égale " + str$(compteur));
```

Chapitre 8 : les tableaux

A la place de réserver chaque variable séparemment, on peut réserver tout un tableau de variables.

exemple:

```
  var compteur[100];       // je crée 101 compteurs
```

En fait, l'exemple ci-dessus crée les 101 compteurs suivants :

```
  compteur[0]
  compteur[1]
  compteur[2]
```

```
  compteur[3]
  ...
  compteur[100]
```

On peut stocker des valeurs dans ces compteurs, comme ceci :

exemple:

```
  compteur[5] = 12;    // stocker 12 dans le compteur numéro 5

  compteur[5]++;       // augmenter le compteur 5 de 1

  compteur[i] = 13;    // stocker 13 dans le compteur i
```

Le dernier exemple ci-dessus va donner une erreur si la variable "i"
n'a pas de valeur comprise entre 0 et 100.

Chapitre 9 : l'instruction IF

Les conditions servent à "tester" des variables.

On peut par exemple dire "Si la variable machin est égale à 50, fais ceci"...

Mais ce serait dommage de ne pouvoir tester que l'égalité !

Il faudrait aussi pouvoir tester si la variable est inférieure à 50,
inférieure ou égale à 50, supérieure, supérieure ou égale...

Ne vous inquiétez pas, le langage script a tout prévu (mais vous n'en doutiez
pas hein)

Avant de voir comment on écrit une condition de type "if... else",
il faut donc que vous connaissiez 2-3 symboles de base.
Ces symboles sont indispensables pour réaliser des conditions.

Voici un petit tableau de symboles à connaître par coeur :

Symbole	Signification
==	Est égal à
>	Est supérieur à
<	Est inférieur à
>=	Est supérieur ou égal à
<=	Est inférieur ou égal à
!=	Est différent de

Faites très attention, il y a bien 2 symboles "==" pour tester l'égalité.
Une erreur courante que font les débutants et de ne mettre qu'un symbole =.
Je vous en reparlerai un peu plus bas.

Un if simple

Attaquons maintenant sans plus tarder. Nous allons faire un test simple, qui va
dire à l'ordinateur :

SI la variable vaut ça
ALORS fais ceci

Par exemple, on pourrait tester une variable "age" qui contient votre âge.
Tenez pour s'entraîner, on va tester si vous êtes majeur,
c'est-à-dire si votre âge est supérieur ou égal à 18 :

exemple:

```
  if (age >= 18)
  {
    print ("Vous êtes majeur !");
  }
```

Le symbole >= signifie "Supérieur ou égal", comme on l'a vu dans le tableau tout à l'heure.

S'il n'y a qu'une instruction entre les accolades, alors celles-ci deviennent facultatives.
Vous pouvez donc écrire :

exemple:

```
  if (age >= 18)
    print ("Vous êtes majeur !");
```

Si vous voulez tester les codes précédents pour voir comment le if fonctionne, il faudra placer le if à l'intérieur d'une procédure timer et ne pas oublier de déclarer
une variable age à laquelle on donnera la valeur de notre choix.

Ca peut paraître évident pour certains, mais apparemment ça ne l'était pas pour tout le monde
aussi ai-je rajouté cet exemple :

exemple:

```
  proc timer ()
  {
    var age;

    age = 20;

    if (age >= 18)
      print ("Vous êtes majeur !");
  }
```

Ici, la variable age vaut 20, donc le "Vous êtes majeur !" s'affichera.

Essayez de changer la valeur initiale de la variable pour voir.
Mettez par exemple 15 : la condition sera fausse, et donc "Vous êtes majeur !" ne s'affichera pas cette fois.

Servez-vous de ce code de base pour tester les prochains exemples du chapitre.

Une question de propreté

La façon dont vous ouvrez les accolades n'est pas importante,
votre programme marchera aussi bien si vous écrivez tout sur une même ligne.

Par exemple :

```
  if (age >= 18) {  print ("Vous etes majeur !"); }
```

Pourtant, même si c'est possible d'écrire comme ça, c'est ultra déconseillé !!
En effet, tout écrire sur une même ligne rend votre code difficilement lisible.

Si vous ne prenez pas dès maintenant l'habitude d'aérer votre code,
plus tard quand vous écrirez de plus gros programmes vous ne vous y retrouverez plus !

Essayez donc de présenter votre code source de la même façon que moi :
une accolade sur une ligne, puis vos instructions (précédées de deux blancs
pour les "décaler vers la droite"), puis l'accolade de fermeture sur une ligne.

Il existe plusieurs bonnes façons de présenter son code source.

Ca ne change rien au fonctionnement du programme final, mais c'est une question de "style informatique" si vous voulez. Si vous voyez un code de quelqu'un d'autre
présenté un peu différemment, c'est qu'il code avec un style différent.

Le principal, c'est que son code reste aéré et lisible.

Le "else" pour dire "sinon"

Maintenant que nous savons faire un test simple, allons un peu plus loin :
si le test n'a pas marché (il est faux), on va dire à l'ordinateur
d'exécuter d'autres instructions.

En français, nous allons donc écrire quelque chose qui ressemble à cela :

```
SI la variable vaut ça
ALORS fais ceci
SINON fais cela
```

Il suffit de rajouter le mot else après l'accolade fermante du if.
Petit exemple :

```
  if (age >= 18) // Si l'âge est supérieur ou égal à 18
  {
    print ("Vous etes majeur !");
  }
  else   // Sinon...
  {
    print ("Ah c'est bete, vous etes mineur !");
  }
```

Les choses sont assez simples : si la variable age est supérieure ou égale à 18,
on affiche le message "Vous êtes majeur !", sinon on affiche "Vous êtes mineur".

Le "else if" pour dire "sinon si"

On a vu comment faire un "si" et un "sinon".
Il est possible aussi de faire un "sinon si".

On dit dans ce cas à l'ordinateur :

SI la variable vaut ça ALORS fais ceci
SINON SI la variable vaut ça ALORS fais ça
SINON fais cela

Traduction:

```
  if (age >= 18)       // Si l'âge est supérieur ou égal à 18
  {
    print ("Vous etes majeur !");
  }
  else if (age > 4)    // Sinon, si l'âge est au moins supérieur à 4
  {
    print ("Bon t'es pas trop jeune quand meme...");
  }
  else                 // Sinon...
  {
    print ("Aga gaa aga gaaa gaaa");   // Langage Bébé, vous ne pouvez pas
comprendre ;o)
  }
```

L'ordinateur fait les tests dans l'ordre :

D'abord il teste le premier if : si la condition est vraie, alors il exécute ce qui se trouve entre les premières accolades.

Sinon, il va au "sinon si" et fait à nouveau un test : si ce test est vrai, alors il exécute les instructions correspondantes entre accolades.

Enfin, si aucun des tests précédents n'a marché, il exécute les instructions du "sinon".

Le "else" et le "else if" ne sont pas obligatoires. Pour faire une condition, il faut juste au moins un "if" (logique me direz-vous, sinon il n'y a pas de condition !)

Notez qu'on peut mettre autant de "else if" que l'on veut. On peut donc écrire :

SI la variable vaut ça
ALORS fais ceci
SINON SI la variable vaut ça ALORS fais ça
SINON SI la variable vaut ça ALORS fais ça
SINON SI la variable vaut ça ALORS fais ça
SINON fais cela

Plusieurs conditions à la fois

Il peut aussi être utile de faire plusieurs tests à la fois dans votre if.
Par exemple, vous voudriez tester si l'âge est supérieur à 18 ET si l'âge est inférieur à 25.
Pour faire cela, il va falloir utiliser de nouveaux symboles :

Symbole	Signification
&&	ET
||	OU
!	NON

Test ET

Si on veut faire le test que j'ai mentionné plus haut, il faudra écrire :

exemple:

```
if (age > 18 && age < 25)
```

Les deux symboles "&&" signifient ET.

Notre condition se dirait en français :

"Si l'âge est supérieur à 18 ET si l'âge est inférieur à 25"

Test OU

Pour faire un OU, on utilise les 2 signes ||. Je dois avouer que ce signe n'est pas facilement accessible sur nos claviers. Pour le taper sur un clavier AZERTY français, il faudra faire Alt Gr + 6. Sur un clavier belge, il faudra faire Alt Gr + &.

Imaginons un programme débile qui décide si une personne a le droit d'ouvrir un compte en banque. C'est bien connu, pour ouvrir un compte en banque, il vaut mieux ne pas être trop jeune (on va dire arbitrairement qu'il faut avoir au moins 30 ans) ou bien avoir plein d'argent (parce que là même à 10 ans on vous acceptera à bras ouverts)

Notre test pour savoir si le client a le droit d'ouvrir un compte en banque pourrait être :

```
if (age > 30 || argent > 100000)
{
  print ("Bienvenue chez Picsou Banque !");
}
else
{
  print ("Hors de ma vue, misérable !");
}
```

Ce test n'est valide que si la personne a plus de 30 ans ou si elle possède plus de 100 000 euros

Test NON

Le dernier symbole qu'il nous reste à tester est le point d'exclamation.
En informatique, le point d'exclamation signifie "Non".
Vous devez mettre ce signe avant votre condition pour dire "Si cela n'est pas vrai" :

exemple:

```
if (!(age < 18))
```

Cela pourrait se traduire par "Si la personne n'est pas mineure".

Si on avait enlevé le "!" devant, cela aurait signifié l'inverse : "Si la personne est mineure".

Quelques erreurs courantes de débutant

N'oubliez pas les 2 signes ==

Si on veut tester si la personne a tout juste 18 ans, il faudra écrire :

exemple:

```
  if (age == 18)
  {
    print ("Vous venez de devenir majeur !");
  }
```

N'oubliez pas de mettre 2 signes "égal" dans un if, comme ceci : ==

Le point-virgule de trop

Une autre erreur courante de débutant : vous mettez parfois un point-virgule à la fin de la ligne d'un if. Or, un if est une condition, et on ne met de point-virgule
qu'à la fin d'une instruction et non d'une condition.
Le code suivant ne marchera pas comme prévu car il y a un point-virgule à la fin du if :

exemple:

```
  if (age == 18); // Notez le point-virgule ici qui ne devrait PAS être là
  {
    print ("Tu es tout juste majeur");
  }
```

Les booléens, le coeur des conditions

Nous allons maintenant rentrer plus en détail dans le fonctionnement d'une condition de type if... else.
En effet, les conditions font intervenir quelque chose qu'on appelle les booléens en informatique.

C'est un concept très important, donc ouvrez grand vos oreilles (euh vos yeux plutôt)

Quelques petits tests pour bien comprendre

En cours de Physique-Chimie, mon prof avait l'habitude de nous faire commencer par quelques petites expériences avant d'introduire une nouvelle notion.
Je vais l'imiter un peu aujourd'hui.

Voici un code source très simple que je vous demande de tester :

exemple:

```
  if (1)
```

```
  {
    print ("C'est vrai");
  }
  else
  {
    print ("C'est faux");
  }
```

Résultat :

```
  C'est vrai
```

Mais ??? On n'a pas mis de condition dans le if, juste un nombre.
Qu'est-ce que ça veut dire ça n'a pas de sens ?

Si ça en a, vous allez comprendre.
Faites un autre test maintenant en remplaçant le 1 par un 0 :

exemple:

```
  if (0)
  {
    print ("C'est vrai");
  }
  else
  {
    print ("C'est faux");
  }
```

Résultat :

```
  C'est faux
```

Faites maintenant d'autres tests en remplaçant le 0 par n'importe quel
autre nombre entier, comme 4, 15, 226, -10, -36 etc...
Qu'est-ce qu'on vous répond à chaque fois ? On vous répond : "C'est vrai".

Résumé de nos tests : si on met un 0, le test est considéré comme faux,
et si on met un 1 ou n'importe quel autre nombre, le test est vrai.

Des explications s'imposent

En fait, à chaque fois que vous faites un test dans un if,
ce test renvoie la valeur 1 s'il est vrai, et 0 s'il est faux.

Par exemple :

```
  if (age >= 18)
```

Ici, le test que vous faites est "age >= 18".

Supposons que age vaille 23. Alors le test est vrai,
et l'ordinateur "remplace" en quelque sorte "age >= 18" par 1.

Ensuite, l'ordinateur obtient (dans sa tête) un "if (1)".

Quand le nombre est 1, comme on l'a vu, l'ordinateur dit que la condition
est vraie, donc il affiche "C'est vrai" !

De même, si la condition est fausse, il remplace age >= 18 par le nombre 0, et du coup la condition est fausse : l'ordinateur va lire les instructions du "else".

Un test avec une variable

Testez maintenant un autre truc : envoyez le résultat de votre condition dans une variable,
comme si c'était une opération (car pour l'ordinateur, c'est une opération !).

exemple:

```
var age, majeur;

age = 20;

majeur = age >= 18;

print ("majeur vaut : " + str$(majeur));
```

Comme vous le voyez, la condition age >= 18 a renvoyé le nombre 1 car elle est vraie.
Du coup, notre variable majeur vaut 1, on vérifie d'ailleurs ça en faisant un print
qui montre bien qu'elle a changé de valeur.

Faites le même test en mettant age = 10 par exemple. Cette fois, majeur vaudra 0.

Cette variable "majeur" est un booléen !

Retenez bien ceci :

On dit qu'une variable à laquelle on fait prendre les valeurs 0 et 1 est un booléen.

Et aussi ceci :

0 = Faux
1 = Vrai

Pour être tout à fait exact, 0 = faux et tous les autres nombres valent vrai (on a eu l'occasion de le tester plus tôt).

Ceci dit, pour simplifier les choses on va se contenter de n'utiliser que les chiffres 0 et 1, pour dire si "quelque chose est faux ou vrai".

Les booléens dans les conditions

Souvent, on fera un test "if" sur une variable booléenne :

exemple:

```
var majeur;
```

```
majeur = 1;

if (majeur)
{
  print ("Tu es majeur !");
}
else
{
  print ("Tu es mineur");
}
```

Comme majeur vaut 1, la condition est vraie, donc on affiche "Tu es majeur !".

Ce qui est très pratique, c'est que la condition se lit facilement par un être humain.
On voit "if (majeur)", ce que peut traduire par "Si tu es majeur".

Les tests sur des booléens sont donc faciles à lire et à comprendre,
pour peu que vous ayez donné des noms clairs à vos variables
comme je vous ai dit de le faire depuis le début.

Tenez, voici un autre test imaginaire :

exemple:

```
if (majeur && garcon)
```

Ce test signifie "Si tu es majeur ET que tu es un garçon".
garcon est ici une autre variable booléenne qui vaut 1 si vous êtes un garçon,
et 0 si vous êtes... une fille ! Bravo, vous avez tout compris !

Les booléens servent donc à exprimer si quelque chose est vrai ou faux.
C'est très utile, et ce que je viens de vous expliquer vous permettra
de comprendre bon nombre de choses par la suite.

Petite question : si on fait le test "if (majeur == 1)", ça marche aussi non ?

Tout à fait. Mais le principe des booléens c'est justement de raccourcir
l'expression du if et de la rendre plus facilement lisible.
Avouez que "if (majeur)" ça se comprend très bien non ?

Retenez donc : si votre variable est censée contenir un nombre,
faites un test sous la forme "if (variable == 1)".
Si au contraire votre variable est censée contenir un booléen (c'est-à-dire
soit 1 soit 0 pour dire vrai ou faux), faites un test sous la forme "if
(variable)".

La condition "if... else" que l'on vient de voir est le type de condition le
plus souvent utilisé.
En fait, il n'y a pas 36 façons de faire une condition. Le "if... else" permet
de gérer tous les cas.

exemple:

```
if (age == 2)
{
  print ("Salut bebe !");
```

```
  }
  else if (age == 6)
  {
    print ("Salut gamin !");
  }
  else if (age == 12)
  {
    print ("Salut jeune !");
  }
  else if (age == 16)
  {
    print ("Salut ado !");
  }
  else if (age == 18)
  {
    print ("Salut adulte !");
  }
  else if (age == 68)
  {
    print ("Salut papy !");
  }
  else
  {
    print ("Je n'ai aucune phrase de prete pour ton age ");
  }
```

Chapitre 10 : les boucles

Qu'est-ce qu'une boucle ?
C'est une technique permettant de répéter les mêmes instructions plusieurs fois.

Tout comme pour les conditions, il y a plusieurs façons de réaliser des boucles.
Au bout du compte, cela revient à faire la même chose :
répéter les mêmes instructions un certain nombre de fois.

Dans tous les cas, le schéma est le même :

```
       <----------
  Instructions     ^
  Instructions     ^
  Instructions     ^
  Instructions     ^
       -----------^
```

Voici ce qu'il se passe dans l'ordre :

L'ordinateur lit les instructions de haut en bas (comme d'habitude)
Puis, une fois arrivé à la fin de la boucle, il repart à la première instruction
Il recommence alors à lire les instructions de haut en bas...
... Et il repart au début de la boucle.

Le problème dans ce système c'est que si on ne l'arrête pas,
l'ordinateur est capable de répéter les instructions à l'infini !
Il n'est pas du genre à se plaindre vous savez, il fait ce qu'on lui dit de faire ..

Et c'est là qu'on retrouve... des conditions !

Quand on crée une boucle, on indique toujours une condition.

Cette condition signifiera "Répète la boucle tant que cette condition est vraie".

Il y a plusieurs manières de s'y prendre comme je vous l'ai dit.
Voyons voir sans plus tarder comment on réalise une boucle de type "while".

La boucle while

Voici comment on construit une boucle while :

exemple:

```
while ( Condition )
{
    // Instructions à répéter
}
```

C'est aussi simple que cela. While signifie "Tant que".
On dit donc à l'ordinateur "Tant que la condition est vraie : répète les instructions entre accolades".

On veut que notre boucle se répète un certain nombre de fois.
On va pour cela créer une variable "compteur" qui vaudra 0 au début du programme et que l'on va incrémenter au fur et à mesure.

Vous vous souvenez de l'incrémentation ? Ca consiste à ajouter 1 à la variable en faisant "variable++;".

Regardez attentivement ce bout de code et, surtout, essayez de le comprendre :

exemple:

```
var compteur;

compteur = 0;

while (compteur < 5)
{
  print ("Salut !");
  compteur++;
}
```

Résultat :

```
Salut !
Salut !
Salut !
Salut !
Salut !
```

Ce code répète 5 fois l'affichage de "Salut !".

Comment ça marche exactement ?

Au départ, on a une variable compteur initialisée à 0. Elle vaut donc 0 au début

du programme.

La boucle while ordonne la répétition TANT QUE compteur est inférieur à 5.
Comme compteur vaut 0 au départ, on rentre dans la boucle.

On affiche la phrase "Salut !" via un print.

On incrémente la valeur de la variable compteur, grâce à l'instruction
"compteur++;".
Compteur valait 0, il vaut maintenant 1.
On arrive à la fin de la boucle (accolade fermante), on repart donc au début,
au niveau du while.

On refait le test du while : "Est-ce que compteur est toujours inférieur à 5 ?".
Ben oui, compteur vaut 1 Donc on recommence les instructions de la boucle.

Et ainsi de suite... Compteur va valoir progressivement 0, 1, 2, 3, 4, 5.

Lorsque compteur vaut 5, la condition "compteur < 5" est fausse.
Comme l'instruction est fausse, on sort de la boucle.

On pourrait voir d'ailleurs que la variable compteur augmente au fur
et à mesure dans la boucle, en l'affichant dans le print :

exemple:

```
  var compteur;

  compteur = 0;

  while (compteur < 5)
  {
    print ("la variable compteur vaut " + str$(compteur));
    compteur++;
  }
```

```
La variable compteur vaut 0
La variable compteur vaut 1
La variable compteur vaut 2
La variable compteur vaut 3
La variable compteur vaut 4
```

Voilà, si vous avez compris ça, vous avez tout compris.
Vous pouvez vous amuser à augmenter la limite du nombre de boucles
 ("< 10" au lieu de "< 5").

Attention aux boucles infinies

Lorsque vous créez une boucle, assurez-vous toujours qu'elle peut s'arrêter
à un moment ! Si la condition est toujours vraie, votre programme ne s'arrêtera
jamais !

Voici un exemple de boucle infinie :

exemple:

```
  while (1)
  {
    print ("Boucle infinie");
  }
```

Souvenez-vous des booléens : 1 = vrai, 0 = faux. Ici, la condition est toujours vraie,
donc ce programme affichera "Boucle infinie" sans arrêt !

Pour arrêter un tel programme, vous n'avez pas d'autre choix que de fermer le chat !

Faites donc très attention : évitez à tout prix de tomber dans une boucle infinie.

La boucle for

En théorie, la boucle while permet de réaliser toutes les boucles que l'on veut. Toutefois, il est dans certains cas utiles d'avoir un autre système de boucle plus "condensé", plus rapide à écrire.

Les boucles for sont très très utilisées en programmation.
Je n'ai pas de statistiques sous la main, mais sachez que vous utiliserez certainement autant de for que de while, donc il vous faudra savoir manipuler ces deux types de boucles.

Comme je vous le disais, les boucles for sont juste une autre façon de faire une boucle while.

Voici un exemple de boucle while que nous avons vu tout à l'heure :

exemple:

```
  var compteur;

  compteur = 0;

  while (compteur < 10)
  {
    print ("Salut !");
    compteur++;
  }
```

Voici maintenant l'équivalent en boucle for :

exemple:

```
  var compteur;

  for (compteur = 0 ; compteur < 10 ; compteur++)
  {
    print ("Salut !");
  }
```

Quelles différences ?

Il y a beaucoup de choses entre les parenthèses après le for (nous allons détailler ça après)
Il n'y a plus de compteur++; dans la boucle.

Intéressons-nous à ce qui se trouve entre les parenthèses, car c'est là que réside
tout l'intérêt de la boucle for. Il y a 3 instructions condensées,
séparée chacune par un point-virgule :

La première est l'initialisation : cette première instruction est utilisée
pour préparer notre variable compteur. Dans notre cas, on initialise la variable
à 0.

La seconde est la condition : comme pour la boucle while, c'est la condition qui dit
si la boucle doit être répétée ou pas. Tant que la condition est vraie, la
boucle for continue.

Enfin, il y a l'incrémentation : cette dernière instruction est exécutée à la fin
de chaque tour de boucle pour mettre à jour la variable compteur.
La quasi-totalité du temps on fera une incrémentation, mais on peut aussi
faire une décrémentation (variable--;) ou encore n'importe quelle autre opération
(variable += 2; pour avancer de 2 en 2 par exemple).

Bref, comme vous le voyez la boucle for n'est rien d'autre qu'un condensé du while.
Sachez vous en servir, vous en aurez besoin plus d'une fois !

Chapitre 11 : procédures et fonctions de base

Qu'est-ce qu'une procédure ? ben par exemple "print" que vous connaissez bien.
Mais il en existe bien d'autres !

Vous connaissez déjà deux fonctions aussi : str$ et val, qui permettent
de convertir entre string et valeur numérique.

En fait la différence entre procédure et fonction, c'est qu'une procédure
c'est une commande, tandis qu'une fonction ça calcule quelque chose et le renvoie
comme résultat.

Nous allons étudier quelques fonctions courantes pour voir comment ça marche :

```
d$ = now$ ();           // fonction qui calcule la date du jour
                        // exemple: now$() == "2007/01/17 12:23:12"

s$ = mid$(s$,a,N);      // renvoie N caractères de s$ commençant au 'a'ième
                        // exemple: mid$("ABCDEF",2,3) == "BCD"

n = random (a,b);       // renvoie un nombre aléatoire entre a et b
                        // exemple: random(1,50) == 12
```

La fonction now$() est simple : elle renvoie la date et l'heure du PC.
Par exemple on peut écrire :

exemple:

```
  var ma_date$;

  ma_date$ = now$();   // mettre la date dans ma_date$
```

```
  print (ma_date$);
```

ce qui va imprimer par exemple:

```
  2007/01/17 12:23:12
```

La fonction mid$() est très utile pour extraire un sous-string d'un string.
Par exemple imaginez que je veuille extraire juste l'heure de la date ci-dessus.

exemple:

```
  var ma_date$, heure$;

  ma_date$ = now$();   // mettre la date dans ma_date$

  heure$ = mid$ (ma_date$, 12, 2);   // prendre une copie de la date à partir du
12 ième caractère,
                                     // et copier 2 caractères
  print (heure$);
```

ce qui va imprimer par exemple:

```
  12
```

La fonction random() permet de tirer un nombre au hasard, ce qui est très utile
pour les jeux.

exemple:

```
  var i;

  i = random (1, 100);   // choisir un nombre de 1 à 100 à mettre dans i

  print ("on choisit " + str$(i));
```

Dans la documentation du langage script, vous trouverez des listes
de toutes les procédures et fonctions existantes.

Chapitre 12 : le texte qui se répète

Voici un script qui répète un texte toutes les 10 secondes.

```
proc timer ()
{
  print ("Salut les amis !");
  print ("J'espère que vous allez bien :p");

  set_timer (10);
}
```

Vous savez déjà depuis le tout premier exemple
de ce cours que la procédure timer est exécutée
une fois au lancement du script.

La procédure set_timer(10); est nouvelle et signifie:

"répète la procédure timer dans 10 secondes !"

Quand la procédure est répétée après 10 secondes,
elle ré-écrit le texte, et le set_timer la rédéclenche
pour dans 10 secondes, etc ... et ça tourne en boucle !

Chapitre 13 : plusieurs répétitions

Comment faire pour répéter une phase toutes les 10 secondes
et une autre toutes les 20 secondes ?

```
proc timer ()
{
  print ("Salut les amis !");
  set_timer (10);

  print ("J'espère que vous allez bien :p");
  set_timer (20);
}
```

Le script ci-dessus est incorrect !
En effet, seul le dernier set_timer est utilisé pour la répétition.
Ce script va donc se dérouler en ignorant complètement le premier set_timer !

Il n'est pas permis non plus d'avoir plusieurs proc timer() dans le même
script !

Par contre, ceci est possible :

```
// Exemple de répétition à plusieurs branches

var secondes;

proc timer ()
{
  secondes++;

  if (secondes == 10)
  {
    print ("Salut les amis !");
  }

  if (secondes == 20)
  {
    print ("J'espère que vous allez bien :p");
  }

  if (secondes == 20)
    secondes = 0;

  set_timer (1);
}
```

Cet exemple, déjà assez complexe, montre plusieurs techniques très utilisées.

Tout d'abord, remarquez le set_timer(1); en fin de script qui fait en sorte que la procédure timer sera exécutée une fois par seconde.

Regardez ensuite tout en haut du script. On y déclare une variable que j'ai appelée "secondes". Cette variable va compter le nombre de fois qu'on est passé dans la procédure timer.

Comme cette variable est déclarée tout en haut du script, elle va garder sa valeur
tout au long du déroulement du script. Sa valeur est zéro au démarrage.

Maintenant regardons ce qu'on fait une fois par seconde :

La commande secondes++; a pour effet d'augmenter de 1 le compteur.
Ensuite, si secondes égale 10, on affiche une ligne de texte.
Si secondes égale 20, on affiche une autre ligne de texte.
Et finalement, si secondes égale 20, on remet 'secondes' à zéro, ce qui va répéter le cycle.

Donc si vous avez bien compris l'exemple, la 1ère phrase s'affiche après 10 secondes,
la 2ème phrase après 20 secondes, et ensuite le cycle recommence.

Chapitre 14 : les actions

Aux chapitres précédents, vous avez pu écrire des scripts qui tournaient tout seuls en ignorant complètement les chatteurs.

Maintenant, nous allons passer à un autre genre de script :
ceux qui interagissent avec les chatteurs et qui leur répondent !

En effet, les chatteurs provoquent des actions, par exemple :

- ils entrent en salle
- ils écrivent une ligne de texte
- ils sont kickés
- ils vont en tasse
- ils mettent leur cam

etc ...

Il faut savoir que pendant 99% du temps, un script ne fait rien, il dort !

Mais dès qu'un chatteur provoque une action, le script est réveillé !
Il exécute alors quelques instructions (par exemple il affiche un message), et quand c'est terminé, le script se remet à dormir en attendant la prochaine action.

Entre deux actions, donc pendant qu'il dort, le script a complètement perdu la mémoire, c'est-à-dire que toutes les variables sont remises à zéro !
Il y a une seule exception à cette règle, ce sont les variables qui sont déclarées
tout en haut du script. Ces dernières sont donc très utiles car elles gardent leur valeur tant que le script tourne !

A chaque fois qu'une action se produit, une procédure event est exécutée.

Exemple :

```
// une procédure event qui affiche un message de bienvenue à ceux qui entrent en
salle


proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  if (action == 128)
    print ("Bienvenue " + userid$ + " !  j'espère que tu vas bien :) ");
}
```

Dès qu'une action se produit, la procédure event est exécutée. Elle reçoit tout un tas de paramètres concernant l'action :

- le nom du chatteur (userid$),
- le type d'action (action),
- s'il a un toc (toc_capab),
- etc ..

La première chose à faire est de tester le type d'action qui est contenu dans le paramètre 'action'. Dans l'exemple ci-dessus, on teste s'il s'agit d'une action numéro 128 (arrivée en salle). Si c'est le cas, on imprime un message avec le nom du chatteur.

Chapitre 15 : compteurs

Le script suivant dit à chaque chatteur combien de chatteurs sont rentrés avant lui en salle. Pour cela, il utilise une variable déclarée en haut du script.

Exemple :

```
// un script qui compte les chatteurs

var compte_chatteurs;

proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  if (action == 128)
  {
    compte_chatteurs++;
    print ("Bienvenue " + userid$ + " !");
    print ("Tu es le chatteur numéro " + str$(compte_chatteurs) + " depuis que
mon bot tourne !");
  }
}
```

Chapitre 16 : la procédure print

==========

La procédure print() permet d'afficher un texte en salle, par exemple :

```
  print ("Bonjour !");
```

On peut coller plusieurs morceaux de textes ensembles en utilisant l'opérateur + (plus), par exemple:

```
  print ("Bonjour "    +    "Chers Amis !");
```

bon, dans ce cas-ci, on aurait pu écrire aussi :

```
  print ("Bonjour Chers Amis !");
```

Mais voici un cas plus intéressant :

```
  print ("Bonjour "   +   userid$   +   " !");
```

ici, on colle le bonjour et le nom du chatteur ensembles.

Remarquez que userid$ se termine par un signe dollar ($).

Tous les noms qui se terminent par dollar contiennent du texte !
A l'inverse, les noms ne se terminant pas par dollar contiennent une valeur numérique, par exemple compte_chatteurs au chapitre précédent.

Notez que si vous écrivez :

```
    print (compte_chatteurs);
```

vous allez avoir une erreur car print() n'accepte que du texte !

Pour afficher une valeur, il faut d'abord la convertir en texte avec la fonction str$(). Exemple :

```
    print ("Tu es le chatteur numéro " + str$(compte_chatteurs) + " depuis que mon bot tourne !");
```

Chapitre 17 : macros et fonctions

==========

Le chapitre "Macros" du chat explique comment changer de couleur ou de style de texte.
En script, il suffit pour cela d'intercaler une suite de 4 codes avec la fonction chr$().

Exemple:

```
// un message de bienvenue en ROUGE !
// voir chapitre macros : pour passer au rouge, chr$(1,255,0,0)

proc event (session_key, userid$, sex$, has_photo, age, is_away,
```

```
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  if (action == 128)
    print (chr$(1,255,0,0) + "Bienvenue " + userid$ + " !");
}
```

Cet exemple affiche le message de bienvenue en rouge !

Mais bon, si vous affichez beaucoup de texte rouge
ce n'est pas pratique.

Car le jour où vous voulez changer de couleur, vous devez
changer tous les chr$(1,255,0,0) dans votre programme.

Ce que vous pouvez faire c'est d'écrire votre propre fonction
de changement de couleur.

Exemple:

```
func ftitle$ ()
{
  return chr$(1,255,0,0) + chr$(2,8,0,0);  // couleur rouge + font Impact
}



// un message de bienvenue en ROUGE font Impact !

proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  if (action == 128)
    print (ftitle$() + "Bienvenue " + userid$ + " !");
}
```

Voilà donc si vous voulez changer les couleurs ou le font,
il vous suffit de modifier la fonction ftitle$.

Les numéros de font se trouvent au chapitre Macros dans la documentation du
chat.

Voici un autre exemple qui montre qu'on peut mettre tout un texte en fonction :

Exemple:

```
func red$ ()
{
  return chr$(1,255,0,0);
}


func Impact$ ()
{
  return chr$(2,8,0,0);
}
```

```
func Bienvenue$ (userid$)
{
  return red$() + Impact$() + "Bienvenue " + userid$ + " !";
}


// un message de bienvenue en ROUGE font Impact !

proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  if (action == 128)
    print (Bienvenue$ (userid$));
}
```

Remarquez dans l'exemple ci-dessus qu'on a passé le paramètre userid$
à la fonction Bienvenue$() car elle en avait besoin !

Une autre solution aurait été d'utiliser une procédure au lieu d'une fonction.
La différence entre les deux est très mince : une fonction renvoie une valeur
ou un texte, une procédure ne renvoie rien !

Exemple:

```
func red$ ()
{
  return chr$(1,255,0,0);
}


func Impact$ ()
{
  return chr$(2,8,0,0);
}


proc Bienvenue (userid$)
{
  print (red$() + Impact$() + "Bienvenue " + userid$ + " !");
}


// un message de bienvenue en ROUGE font Impact !

proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  if (action == 128)
    Bienvenue (userid$);
}
```

Ci-dessus, vous pouvez remarquer que la procédure print() n'est plus
dans la procédure event, mais dans la procédure Bienvenue.

Chapitre 18 : d'autres actions

Le script reçoit une action 200 quand un chatteur fait un pv au bot.

Exemple:

```
// bot dénonciateur

proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  if (action == 200)
  {
    print ("Eh les amis, il y a " + userid$ + " qui me fait un PV sauvage !!");
  }
}
```

L'action 2 est un peu plus compliquée : elle indique qu'on va en tasse
ou bien qu'on en sort. La différence entre les deux est indiquée
par le paramètre is_away.

Exemple:

```
// message tasse

proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  if (action == 2)
  {
    if (is_away)
      print ("y a " + userid$ + " qui va en tasse !");
    else
      print ("bon retour, " + userid$);
  }
}
```

Le paramètre is_myself indique si l'action est provoquée par moi-même
(donc le bot). Il est important de tester cette variable si on ne veut
pas que le script réagisse sur le bot.

Exemple:

```
// message tasse SAUF POUR LE BOT !!

proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  if (!is_myself)        // PAS moi-même   (le signe ! est une inversion)
  {
    if (action == 2)
    {
      if (is_away)
        print ("y a " + userid$ + " qui va en tasse !");
      else
```

```
        print ("bon retour, " + userid$);
    }
  }
}
```

Le paramètre toc_capab indique si l'action concerne un chatteur
normal, un animateur ou un proprio.
Le paramètre sex$ indique s'il s'agit d'un homme, femme ou couple.

Exemple:

```
// messages de bienvenue différents

proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  if (action == 128)
  {
    if (toc_capab == 2)
    {
      if (sex$ == "M")
        print ("attention, voici le proprio " + userid$ + " !");
      else if (sex$ == "F")
        print ("attention, voici la proprio " + userid$ + " (f) !");
      else
        print ("attention, voici le couple de proprios " + userid$ + " (f) !");
    }
    else if (toc_capab == 1)
      print ("attention, voici l'animateur " + userid$ + " !");
    else
      print ("bienvenue, " + userid$ + " !");
  }
}
```

Chapitre 19 : tirer un nombre au hasard

Comment afficher un message de bienvenue aléatoire ?

A l'aide de la fonction random() !

Exemple:

```
// message de bienvenue aléatoires

proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  var n;

  if (action == 128)
  {
    n = random (1, 3);   // tire un nombre aléatoire entre 1 et 3

    if (n == 1)
      print ("salut, ça va ?");
```

```
    else if (n == 2)
      print ("oula, toi je ne t'aime pas !");
    else
      print ("reviens plus tard, j'suis pas là :p");
  }
}
```

Remarquez ici que la variable 'n' n'a pas été déclarée en haut du script
comme d'habitude mais dans la procédure : elle perd donc sa valeur
entre deux actions pendant que le script dort, mais cela n'est pas grave
puisqu'on ne souhaite pas la garder.

Chapitre 20 : lire un fichier

Le script suivant va choisir un texte de bienvenue au hasard dans un fichier
que vous avez placé dans le répertoire du bot :

Exemple:

```
// message de bienvenue aléatoire venant d'un fichier

proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  var count, n, ligne$;

  if (action == 128)
  {
    // compte le nombre de lignes dans le fichier bienvenue.txt
    count = count_lines ("bienvenue.txt");

    // tire un nombre au hasard entre 1 et count
    n = random (1, count);

    // extrait la ligne choisie du fichier
    ligne$ = read_line$ ("bienvenue.txt", n);

    // affiche la ligne
    print (ligne$ + userid$);
  }
}
```

Chapitre 21 : ajouter des commandes

Pour ajouter des commandes sur le bot, il faut savoir analyser une ligne
de texte écrite par un chatteur.

C'est assez compliqué car une ligne de texte contient beaucoup de codes
spéciaux (changement de couleur, de font, emoticons, images, sons, ..).

Le plus facile, pour travailler proprement, est de nettoyer d'abord la ligne
de texte en enlevant tous ces codes pour ne garder que le texte pur.

On peut ensuite tester, par exemple à l'aide de la fonction pos(),
si une commande est contenue dans la ligne.

Voici un exemple pratique :

```
// nettoye la ligne line$ en remplaçant toutes les icones par des blancs,
// en convertissant toutes les lettres en minuscules,
// et en entourant tous les mots par des blancs.

func clean_line$ (action, pseudo$, line$)
{
  var last, r$, i, c;

  r$ = " ";

  i = 1;
  last = len(line$);

  if (action == 0)        // ligne de texte
    i += (4*(asc(line$, 1) == 6)) + 4 + len (pseudo$) + 3;
  else if (action == 1)   // avec la bulle
    i += (4*(asc(line$, 1) == 6)) + 4 + 8 + len (pseudo$) + 1;

  while (i <= last)
  {
    c = asc (line$,i);
    if (c <= 9)  // séquence spéciale
    {
      i += 4;    // passer 4 caractères
      if (c > 5)
        i += 4;
      if (c == 1 || c == 2)  // ignorer couleur ou font
        continue;
      c = 32;  // remplacer par un blanc
    }
    else   // caractère normal
    {
      if (c >= 65 && c <= 90)  // si entre A et Z
        c += 32;    // convertir en minuscules
      i++;
    }
    r$ = r$ + chr$(c);
  }

  return r$ + " ";
}


proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  var word$;

  if (action == 0 || action == 1)     // texte normal ou bulle
  {
    word$ = clean_line$ (action, userid$, line$);

    if (pos (word$, " je sors ") > 0)
    {
      kick (session_key, "oué tu sors !");
    }
```

```
    if (pos (word$, " !game ") > 0)
    {
      print ("démarrer le jeu !!");
    }
  }
}
```

La commande !game est entourée exprès par des blancs dans le test

```
    if (pos (word$, " !game ") > 0)
```

en effet, d'une part, la fonction clean_line$ nettoye la ligne
et entoure chaque mot par un blanc, et d'autre part, cela évite
qu'on déclenche la commande par erreur si on tape par exemple

```
     !game2
```

Remarquez aussi la procedure kick() qui sert à éjecter un chatteur de salle.

Chapitre 22 : les tableaux

Pour faire des scripts avancés, il manque encore un concept important :
les tableaux.

En effet, comment faire un script qui kicke les chatteurs
après 30 secondes en tasse puisque le script ne réagit toujours
qu'à une action immédiatement ?

Pour faire ce genre de scripts avancés, il faut que le script garde
en mémoire quelque part tous les chatteurs qui sont actuellement en salle.

On déclare donc, tout en haut du script, des variables tableaux.

L'information qu'on garde (donc le nombre de tableaux différents)
dépend d'un script à l'autre. Souvent on garde le nom du chatteur,
sa session_key (pour pouvoir le kicker plus tard), éventuellement
la date et l'heure à laquelle il est entré en salle.

Voici un script qui ne fait rien du tout quand il tourne !
Il garde seulement en mémoire, dans deux tableaux, les pseudos
et session_key des chatteurs. Vous pouvez cependant l'examiner,
et y ajouter vos propres commandes, voir plus loin.

Exemple:

```
var tab_session[256];     // tableau des session_key des chatteurs
var tab_pseudo$[256];     // tableau des pseudos des chatteurs
var tab_count;            // nombre d'éléments dans tous les tableaux


// ajoute une session_key dans les 2 tableaux

proc ajouter_dans_les_tables (session_key, pseudo$)
{
  var i;
```

```
  for (i=0; i<tab_count; i++)
  {
    if (session_key == tab_session[i])    // trouvé
      break;
  }

  if (i == tab_count)   // pas trouvé, donc agrandir table
    tab_count++;

  tab_session[i] = session_key;
  tab_pseudo$[i] = pseudo$;
}


proc supprimer_des_tables (session_key)
{
  var i;

  for (i=0; i<tab_count; i++)
  {
    if (session_key == tab_session[i])    // trouvé
      break;
  }

  if (i < tab_count)   // trouvé
  {
    tab_session[i] = tab_session[tab_count-1];
    tab_pseudo$[i] = tab_pseudo$[tab_count-1];
    tab_count--;
  }
}



proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  if (!is_myself)
  {
    if (action == 127 || action == 128)  // entrée en salle
    {
      ajouter_dans_les_tables (session_key, userid$);
    }
    else if (action == 129 || action == 130 || action == 256)  // sortie de
salle
    {
      supprimer_des_tables (session_key);
    }
  }


  // ....

}
```

Imaginez que vous voulez kicker tous les chatteurs de la salle.
Il suffit alors de faire une procédure comme ceci que vous pouvez intégrer dans le script précédent :

```
  proc kick_all ()
```

```
  {
    var i;

    // faire une boucle pour parcourir toute la table !

    for (i=0; i<tab_count; i++)
    {
      kick (tab_session[i], "on ferme la salle !");
    }
  }
```

La suite, à vous ...

Chapitre 23 : un script qui affiche la date et l'heure

```
//  Calendrier

//-------------------------------------

func jour_semaine$ (date$)
{
  var sem$[7];

  sem$[1] = "Lundi";
  sem$[2] = "Mardi";
  sem$[3] = "Mercredi";
  sem$[4] = "Jeudi";
  sem$[5] = "Vendredi";
  sem$[6] = "Samedi";
  sem$[7] = "Dimanche";

  return sem$[weekday(date$)];
}

//-------------------------------------

func nom_mois$ (date$)
{
  var mois$[12];

  mois$[1] = "Janvier";
  mois$[2] = "Février";
  mois$[3] = "Mars";
  mois$[4] = "Avril";
  mois$[5] = "Mai";
  mois$[6] = "Juin";
  mois$[7] = "Juillet";
  mois$[8] = "Août";
  mois$[9] = "Septembre";
  mois$[10] = "Octobre";
  mois$[11] = "Novembre";
  mois$[12] = "Décembre";

  return mois$[val(mid$(date$,6,2))];
}

//-------------------------------------
```

```
func heure_de (date$)
{
  return val (mid$(date$,12,2));
}

//-----------------------------------

proc event (session_key, userid$, sex$, has_photo, age, is_away,
            admin, cam_on, is_bot, toc_capab, signature$, suffix$, profile_life,
            action, is_myself, line$)
{
  var date$, heure, phrase$, phrase2$;

  if (action == 128)   // arrivée en salle
  {
    date$ = now$();

    heure = heure_de (date$);

    if (heure >= 12 && heure <= 13)
      phrase$ = "Bon Appétit$mi ";
    else if (heure >= 0 && heure <= 4)
      phrase$ = "Agréable Nuit(s) ";
    else if (heure >= 6 && heure <= 19)
      phrase$ = "Bonjour:) ";
    else
      phrase$ = "Bonsoir;) ";

    if (sex$ == "F")
      phrase2$ = " (f)";
    else
      phrase2$ = " (y)";

    print (phrase$ + userid$ + phrase2$
            + ", nous sommes le " + mid$(date$,9,2)
            + " " + nom_mois$(date$) + " " + mid$(date$,1,4)
            + " et il est " + mid$(date$,12,5)  + "(o)");
  }
}
```

Chapitre 24 : utiliser la database

Vous savez que les variables déclarées en haut du script gardent leur valeur entre deux actions. Mais que se passe-t-il lorsque le bot sort et revient en salle ? Elles sont évidemment effacées ..

Ce qu'il faudrait, c'est un moyen de garder des données de façon permanente, sur le disque dur ...

On utilise pour cela les fonctions store et fetch$ qui permettent d'écrire des informations dans le fichier chat.db du bot puis de les relire plus tard.

```
store (tiroir$, valeur$);   // stocke la valeur$ dans le tiroir$ de la database
chat.db
                            // exemple: store ("JEU-DEVINE", "1 35 16 32 89
12");
                            // la longueur de tiroir$ ne peut pas dépasser 100.

a$ = fetch$ (tiroir$);      // renvoie la valeur contenue dans le tiroir$ de la
```

database
                                // exemple: fetch$ ("JEU-DEVINE") == "1 35 16 32 89
12"

Comment cela fonctionne-il ?

Imaginez que vous êtes dans une grande bibliothèque avec un mur de tiroirs
devant vous.

Sur chaque tiroir est écrit un nom (tiroir$).

Quand vous ouvrez un tiroir vous pouvez y mettre une valeur$ et le refermer.
Plus tard, vous pouvez ouvrir le tiroir à nouveau et récupérer la valeur$ que
vous y avez mise.

C'est exactement comme cela que fonctionnent store et fetch$.

Choix du nom

Quand vous concevez votre programme, vous devez bien réfléchir au nom
que vous allez mettre sur le tiroir.

D'une part, il ne faut pas qu'un autre script utilise les mêmes noms que vous,
sinon il risque de changer le contenu de votre tiroir. Il faut donc ajouter
le nom de votre script au nom du tiroir (ici on va écrire un script "Biblio").

D'autre part, vous devez réfléchir à ce que vous voulez mettre comme nom.
Imaginez que vous voulez garder de l'information sur chaque chatteur.
La première idée est de prendre la signature$ comme nom de tiroir
afin d'avoir un tiroir par chatteur.

Comme plusieurs chatteurs peuvent utiliser le même PC, il vaut cependant
mieux ajouter encore le suffixe de la signature, comme cela chaque
chatteur aura son tiroir :

On va donc utiliser comme nom de tiroir :

```
   "BIBLIO-" + signature$ + suffix$
```

En pratique, pour enregistrer l'info d'un chatteur, vous écrivez :

```
     store ("BIBLIO-" + signature$ + suffix$, info_chatteur$);
```

et pour la relire :

```
     info_chatteur$ = fetch$ ("BIBLIO-" + signature$ + suffix$);
```

Un autre exemple : si vous voulez enregistrer des scores, vous pouvez
utiliser la position dans le score comme nom de tiroir$.
Par exemple:

```
   "BIBLIO-" + "1"  pour le 1er chatteur du tableau de scores,
   "BIBLIO-" + "2"  pour le 2e chatteur du tableau de scores,
   ...
```

Pour mettre à jour votre tableau de 20 scores, vous devrez donc
faire 20 store, ainsi que 20 fetchs$ pour le relire.

La longueur maximum autorisée pour le nom de tiroir$ est de 100 caractères.

Choix de la valeur

Vous êtes évidemment libre de mettre la valeur que vous voulez dans le tiroir. Cela peut être un score, des points, le nombre de fois qu'un chatteur est entré en salle, la date et l'heure de sa dernière venue, ou toute autre info.

La longueur maximum autorisée pour valeur$ est de 8192 caractères.
Mais rien ne vous empêche d'utiliser plusieurs tiroirs ..

Fin

Voilà vous savez tout (ou presque).

Pour approfondir encore les scripts, lisez le chapitre script dans la documentation du chat
au lien ici : http://chat-webcam-samuro.com/chat.fr/script.html

Le même langage script permet aussi d'écrire des applications CONSOLE comme le pacman ou le jeu de dessin. Pour cela, allez voir au lien ici : http://chat-webcam-samuro.com/chat.fr/console/manual-console.html