BCPST	Informatique : D.S. n°1	lundi 17 novembre

Vous devrez rendre 2 copies :

Une copie pour les exercices 1, 2 et 3

Une copie pour les exercices 4 et 5

Une partie du barème est donnée sur la syntaxe du langage Scilab et sur votre interprétation de l'algorithme nécessaire pour répondre à chaque problème. Vous devrez donc expliquer chaque programme par un texte en français.

Exercice 1 (3 points)

Voici un programme Scilab :

```
a=input('premier nombre entier naturel: ');
b=input('second nombre entier naturel: ');
if a<b then
   M=a;
else
   M=b;
end;
for i=1:M do
   if ((modulo(a,i)==0) & (modulo(b,i)==0)) then
      disp(i);
   end;
end;
```

1) Testez ce programme pour a=12 et b=18

2) Que réalise ce programme ?

Exercice 2 (3,5 points)

Un nombre abondant est un nombre entier naturel *n* qui est strictement inférieur à la somme de ses diviseurs stricts.

Exemples : 24 est un nombre abondant parce que on a: 1+2+3+4+6+8+12=36 et 24<36.
6 ou 9 ne le sont pas. En effet, 1+2+3=6 et 1+3=4.

Ecrire un programme Scilab demandant à l'utilisateur deux valeurs x et y (x<y) pour ensuite déterminer et afficher la liste de tous les nombres abondants compris entre x et y. S'il n'y a pas de tels nombres vous afficherez un message pour le signaler.

Exercice 3 (3,5 points)

On démontre que $\lim_{n \to \infty} \sqrt{\underbrace{2 + \sqrt{2 + \sqrt{\ldots + \sqrt{2}}}}_{n \text{ chiffres } 2}} = 2$. On peut traduire cette limite par celle du terme général de la suite définie par la relation de récurrence $u_{n+1} = \sqrt{2 + u_n}$ pour $n \geqslant 1$ et $u_1 = \sqrt{2}$.

1) Ecrire un programme Scilab qui, après avoir demandé à l'utilisateur un entier *n* au clavier, permet de calculer et d'afficher le terme d'indice *n* de la suite définie à partir de la relation de récurrence donnée ci-dessus.

2) Quelle démarche emploieriez-vous pour tenter de connaître la limite de la suite à partir d'exécutions de votre programme ?

Exercice 4 (5 points)

1) a) Ecrire un programme permettant de simuler le lancer d'un dé non pipé à six faces.
   b) Modifier le programme précédent pour simuler 100 lancers d'un tel dé. Vous afficherez la fréquence de la face 1, c'est-à-dire le nombre de fois que la face 1 est sortie sur les 100 lancers.

2) a) Ecrire un programme permettant de simuler le lancer d'un dé pipé à six faces tel que la probabilité de tomber sur la face "1" est double de toutes les autres probabilités.
   b) Modifier le programme précédent pour afficher la fréquence de la face 1.

Exercice 5 (5 points)

1) On veut réaliser les tâches suivantes : un entier *n* étant donné, on veut déterminer chacun de ses chiffres, ensuite calculer le produit *p* de ces chiffres. Ecrire un programme ou la fonction [p]=produit_chiffres(n), qui réalise les tâches demandées.
   Vous pourrez supposer que l'entier *n* est inférieur à 9 999.

2) Pour cette question, on peut, si on le souhaite, utiliser la fonction précédente ou modifier le programme précédent. Il s'agit d'écrire un programme scilab qui, après avoir demandé à l'utilisateur un entier strictement positif, calcule le produit des chiffres de ce nombre entier et recommence le calcul avec le résultat obtenu tant que celui-ci n'est pas compris entre 0 et 9.
   Le programme affichera le nombre entre 0 et 9 obtenu.

   Exemple : si le nombre donné par l'utilisateur est 1849 le programme devra trouver successivement 288 (car 1×8×4×9 = 288) puis 128 (car 2×8×8 = 128) puis 16 (car 1×2×8 = 16) puis 6 (car 1×6 = 6) et afficher 6.

Correction du DS n°1

Exercice 1

1) a = 12 b = 18 //sont les données fournies par l'utilisateur
a = 12 b = 18 M = 12 //car a < b
i = 1 Scilab **affiche 1** car 1 divise 12 et 1 divise 18
i = 2 Scilab **affiche 2** car 2 divise 12 et 2 divise 18
i = 3 Scilab **affiche 3** car 3 divise 12 et 3 divise 18
i = 4, 5 rien ne s'affiche
i = 6 Scilab **affiche 6** car 6 divise 12 et 6 divise 18
i = 7,…, 12 rien ne s'affiche

2) Ce programme affiche tous les diviseurs communs à deux entiers donnés par l'utilisateur (et il suffit d'examiner tous les diviseurs éventuels parmi le plus petit des deux entiers fournis).

Exercice 2

Voici une proposition de correction :

```
x=input("Entrez la borne inférieure");
y=input("Entrez la borne supérieure");
while x>=y do
  disp("Il faut que la première valeur soit inférieure à la seconde")
  x=input("Entrez la borne inférieure");
  y=input("Entrez la borne supérieure");
end

s2=0;            //pour determiner le nombre d'entiers abondants
for n=x:y do
  s=0 ;          //pour la somme des diviseurs de chaque entier
  for i=1:n-1 do
    if n/i==floor(n/i) then s=s+i;end
  end
  if n<s then disp(n);s2=s2+1;
  end
end
if s2==0 then disp('Il n''y a pas de nombre abondant dans l''intervalle proposé');end
```

Exercice 3

1) Voici une proposition de correction :

```
n=input("Veuillez donner le rang pour lequel vous voulez calculer le terme de la suite");
u=sqrt(2);
for i=2:n do
  u=sqrt(2+u);
end
disp(u)
```

2) Vous ne pouvez pas rentrer une grande valeur pour *n* car certaines suites convergent très vite, d'autres non. Ne connaissant pas la vitesse de convergence de celle-ci, vous ne pouvez pas décider d'une valeur qui serait proche de la limite.
Deux possibilités : soit vous modifiez le programme de façon à trouver la valeur à partir de laquelle la variation des valeurs de la suite est très faible (différence relative inférieure à $1.10^{-??}$), soit vous exécutez le programme précédent pour quelques valeurs de *n* et vous estimerez très rapidement la limite en comparant les valeurs obtenues (peu de variation de la valeur pour deux valeurs de *n* supposées grandes).

Exercice 4

1) a) floor(rand()*6)+1 permet de simuler un dé à 6 faces

b) Une proposition de correction est :

```
s=0
for i=1:100 do
  if floor(rand()*6+1)==1 then s=s+1;
  end
end
disp('La fréquence est',s/100)
```

2) a) x=floor(rand()*7)

if x==0 then x=1;end //pour "doubler" la fréquence du 1 par rapport aux autres valeurs

b) Une proposition de correction est :

```
s=0
for i=1:100 do
  x=floor(rand()*7)
  if x==0 then x=1;end
  if x==1 then s=s+1;
  end
end
disp('La fréquence est',s/100)
```

Exercice 5

1) Une proposition de correction est :

```
function [p]=produit_chiffres(n)
p=1;
//une variante serait de considérer les cas d'entiers à 1 chiffre, à 2 chiffres, à 3 chiffres ou à 4 chiffres.
//ces quatre cas définis par des conditionnelles seront construits
while n>0 do
  a=n-floor(n/10)*10; //pour ne conserver que les unités dans n
  p=p*a;
  n=(n-a)/10;
end
endfunction
```

2) Une proposition de correction est :

```
n=input("Veuillez enter un entier positif");
while n<>floor(n) | n<0 do
    n=input("Veuillez enter un entier positif !!");
end

while n>9 do
  n=produit_chiffres(n);
end

disp(n)
```

BCPST	Informatique : D.S. n°2	lundi 23 mars

Vous devrez rendre 2 copies, une copie pour les exercices 1 et 2 et une copie pour les exercices 3 et 4.

Une partie du barème est donnée sur la syntaxe du langage Scilab et sur votre interprétation de l'algorithme nécessaire pour répondre à chaque problème. Vous devrez donc expliquer chaque programme par un texte en français.

Exercice 1 (4 points)

1) Un palindrome est un mot ou une phrase pouvant être lus de droite à gauche et de gauche à droite tout en conservant exactement le même sens. Ecrire une fonction qui vérifie si une chaîne de caractères donnée est un palindrome (on ne s'intéressera pas à la signification du mot ici).

2) Ecrire une fonction qui, étant donné *n* crée aléatoirement un palindrome de taille *n*.

On rappelle deux fonctions scilab : ascii(97) = 'a' et str2code('a') = 10.

Exercice 2 (5 points)

1) Construire une fonction permettant de trier un tableau donné en paramètre. Vous pourrez utiliser une des nombreuses méthodes existantes (vous expliciterez la vôtre) ou considérez ces deux propositions :

- Tri d'un tableau par sélection du minimum ou du maximum.

Voici le principe de l'algorithme du tri par sélection du minimum : Le principe de ce tri consiste à réaliser un premier parcours complet pour rechercher le minimum parmi les *n* éléments d'un tableau, puis un deuxième pour rechercher le minimum parmi les *n* - 1 éléments restants et ainsi de suite. A chaque tour, il faut comparer l'élément de départ, appelé *élément courant, à* tous ses successeurs et effectuer un échange d'emplacement lorsque l'élément courant est supérieur à l'élément utilisé dans la comparaison.

Par exemple, pour trier le tableau 6 2 8 1 5 4
on obtiendra successivement

1 2 8 **6** 5 4	car le minimum est 1
1 **2** 8 6 5 4	car le minimum des autres éléments est 2
1 2 **4** 6 5 **8**	car le minimum des autres éléments est 4
1 2 4 **5** **6** 8	car le minimum des autres éléments est 5
1 2 4 5 **6** 8	car le minimum des autres éléments est 6
1 2 4 5 6 **8**	

- Tri bulle consistant à "faire remonter le maximum partiel" en sa position dans le tableau.

2) Ecrire une fonction [*t*]=creer_tableau(*n*) qui construit un tableau de taille *n* de la façon suivante :
On détermine un nombre choisi aléatoirement dans l'ensemble {1;2;…;1000}. On teste ensuite si ce nombre est un nombre premier. Si c'est le cas alors on le met dans le tableau.
Et ainsi de suite jusqu'à obtenir *n* entiers premiers parmi ceux choisis aléatoirement.
Enfin on utilisera la fonction précédente pour trier le tableau *t* ainsi obtenu par ordre croissant.

Exercice 3 (4 points)

On décide de jouer avec les nombres de la manière suivante : on se donne un nombre, on ajoute les carrés des chiffres qui le composent, on recommence avec le nombre obtenu, et on continue...
Prenons par exemple le nombre 153, il devient successivement $1^2 + 5^2 + 3^2 = 35$ puis $3^2 + 5^2 = 34$ puis …
Construire la fonction [c]=carre(n) permettant de construire le nombre obtenu par ce processus s'il s'arrête (au sens de produire deux fois de suite le même nombre) ou bien celui obtenu à la 10 000ème itération.

Exercice 4 (7 points)

Le premier terme de la suite de Conway est égal à 1. Chaque terme de la suite se construit en annonçant le terme précédent, c'est-à-dire en indiquant combien de fois chacun de ses chiffres se répète.

Explication : $X_1 = 1$.

Ce terme comporte juste "un 1". Par conséquent, le terme suivant est : $X_2 = 11$.

Celui-ci est composé de "deux 1", d'où : $X_3 = 21$.

En poursuivant le procédé : $X_4 = 1211$; $X_5 = 111221$; $X_6 = 312211$.

Et ainsi de suite.

1) On se propose de construire et d'étudier de telles suites, qu'on présentera sous forme de chaînes de caractères.

Ecrire une fonction [c] = Conway(n) qui, étant donné l'entier *n* supérieur ou égal à 1, construit la chaîne *c* représentant le terme de rang *n* de la suite de Conway. On peut utiliser pour cela l'algorithme suivant :

On part d'une chaine *c* initiale.

Puis on construit une chaîne *d* de la façon suivante :

- A partir du début de la chaîne *c*, on compte tous les caractères égaux à celui de la position en cours.
- Ensuite on se positionne au niveau du premier caractère différent (du caractère actuel) et on recommence l'étape précédente.
- Et ainsi de suite …..
- Enfin *d* remplace *c*.

2) On veut réaliser une étude statistique des chiffres composant chaque terme de la suite de Conway. Pour cela vous devez construire une fonction [T] = Statistique(n).

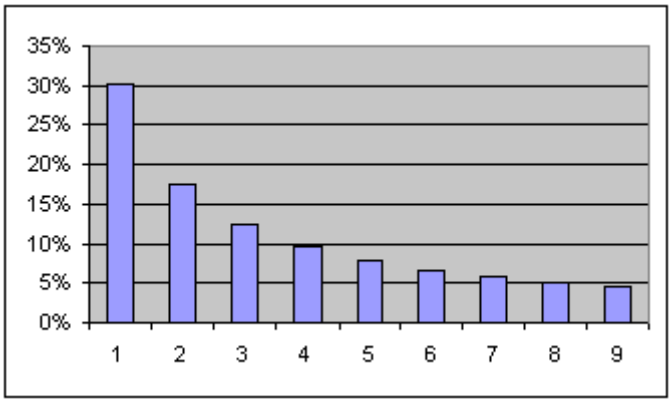
Cette fonction peut utiliser la fonction précédente pour déterminer le terme X_n de la suite de rang *n*. Ensuite elle construit un tableau *T* numéroté de 1 à 9, où chaque $T(i)$ devra contenir le nombre de chiffres de la suite X_n égaux à *i*.

3) Construire la fonction [k] = Chiffre_preponderant(T) qui, à partir du tableau *T* construit précédemment détermine le chiffre le plus représenté dans le terme X_n de la suite.

4) Ecrire une fonction [F] = Moyenne(n), qui permet de construire le tableau *F* à trois éléments contenant les fréquences des chiffres 1, 2 et 3 du terme X_n de la suite de Conway (car vous pourriez remarquer que cette suite ne comporte pas de chiffres strictement supérieurs à 3).

5) Ecrire une fonction A_moitie_des_uns(n) permettant d'afficher tous les termes de la suite jusqu'au rang *n* formés de 50% de chiffres 1.

Remarque : Pour vérifier qu'un fichier de données ne contient pas de valeurs irrégulières ou des erreurs de saisie, on peut utiliser très simplement la loi de Benford. Selon cette loi, la fréquence théorique d'apparition du premier chiffre d'un nombre est vérifiable. Par exemple, dans une suite de nombres aléatoires, le « 2 » a trois fois plus de chance d'être le premier chiffre de chaque nombre que le « 7 ».



Révélée en 1938 par le physicien et statisticien américain Frank Benford cette loi n'a été démontrée mathématiquement qu'en 1996 par Terence Hill malgré une utilisation fréquente par les experts comptables, statisticiens et contrôleurs du fisc.

$$\text{Fréquence du premier chiffre } C = \log_{10}\left(1 + \frac{1}{C}\right)$$

Correction du DS n°2

Exercice 1

```
function [b]=palindrome(c)
  n=length(c);b=%t;
  for i=1:floor(n/2) do
    if part(c,i)<>part(c,n+1-i) then b=%f;end
  end
endfunction

function [c]=creation_palindrome(n)
  c=""
  if modulo(n,2)==0 then for i=1:n/2 do
                          x=floor(rand()*26)
                          c=c+code2str(10+x)
                        end
                        d=""
                        for i=1:n/2 do
                          d=part(c,i)+d
                        end
                        c=c+d
                 else for i=1:floor(n/2) do
                          x=floor(rand()*26)
                          c=c+code2str(10+x)
                        end
                        e=code2str(10+floor(rand()*26))
                        d=""
                        for i=1:floor(n/2) do
                          d=part(c,i)+d
                        end
                        c=c+e+d
  end
endfunction
```

Exercice 2

```
function [t]=tri_minimum(t)
  for i=1:length(t) do
    minimum=i
    for k=i:length(t) do
      if t(k)<t(minimum) then minimum=k;end
    end
    if minimum>i then c=t(minimum);t(minimum)=t(i);t(i)=c;end
  end
endfunction

function [t]=tri_bulle(t)
  for k=1:length(t)-1 do
    for i=1:length(t)-k do
      if t(i+1)<t(i) then c=t(i);t(i)=t(i+1);t(i+1)=c;end
    end
  end
endfunction

function [b]=premier(n)
  s=0;
  for i=1:n do
    if modulo(n,i)==0 then s=s+1;end
  end
  if s==2 then b=%t; else b=%f; end
endfunction

function [t]=creer_tableau(n)
  t=zeros(1,n)
  for i=1:n do
    x=floor(rand()*1000)+1
    while premier(x)==%f do
      x=floor(rand()*1000)+1
    end
    t(i)=x;
  end
endfunction
```

Exercice 3

```
function [c]=carre(n)
  k=1;
  m=n-1
  while m<>n & k<10000 do
    m=n
    c=string(n)
    n=0
    for i=1:length(c) do
      n=n+evstr(part(c,i))^2
    end
    k=k+1
  end
endfunction
```

Exercice 4

```
function [c]=conway(n)
  c='1';
  for k=2:n do
    d=''
    j=1;i=2;
    while j<=length(c) do
      while i<=length(c) & part(c,i)==part(c,j) do
        i=i+1;
      end
      d=d+string(i-j)+part(c,j)
      j=i
    end
    c=d
  end
endfunction
```

```
function [T]=statistique(n)
  T=zeros(1,9)
  c=conway(n)
  for i=1:9 do
    for j=1:length(c) do
      if part(c,j)==string(i) then T(i)=T(i)+1;end
    end
  end
endfunction

function [k]=chiffre_preponderant(T)
  maximum=T(1);m=1;
  for i=2:9 do
    if T(i)>maximum then maximum=T(i);m=i;end
  end
  k=m
endfunction

function [F]=moyenne(n)
  T=statistique(n)
  F=[T(1),T(2),T(3)]
endfunction

function []=A_moitie_des_uns(n)
  for i=1:n do
    F=moyenne(i)
    a=F(1)/(F(1)+F(2)+F(3))
    if a==0.5 then disp(i);end
  end
endfunction
```

Vous devrez rendre 2 copies, une copie pour les exercices 1, 2 et 3 et une copie pour les exercices 4 et 5.

Une partie du barème est donnée sur la syntaxe du langage Scilab et sur votre interprétation de l'algorithme nécessaire pour répondre à chaque problème. Vous devrez donc expliquer chaque programme par un texte en français.

Exercice 1 (3 points) Extrait de l'épreuve de Mathématiques, épreuve A, Agro-Véto 2009 :

Considérons la suite u définie par : $\forall n \in \mathbb{N}^*$, $u_n = \sum_{k=0}^{n-1} \frac{1}{k+n}$,

Ecrire un algorithme ou un programme Scilab qui, pour un entier naturel non nul n donné, calcule la valeur de u_n.

Exercice 2 (4 points)

Une population est classée en deux catégories : juvénile (ou enfant) et adulte. Entre l'année n et l'année $n + 1$:

1/7 de la population juvénile devient adulte,
4/7 de la population juvénile reste juvénile,
2/7 de la population juvénile meurt,
3/4 de la population adulte reste adulte,
1/4 de la population adulte meurt,
Pour trois adultes, il naît en moyenne deux enfants.

On suppose qu'à l'année 2000, il y a 2000 enfants pour 1000 adultes. On note J_n et A_n le nombre d'enfants et d'adultes dans la population à l'année 2000 + n.

1) A l'aide du calcul matriciel, établir une relation permettant de calculer J_{n+1} et A_{n+1} en fonction de J_n et A_n.

2) Construire un programme Scilab permettant de tracer l'évolution de J_n et A_n entre les années 2000 et 2010.

3) Calculer, en utilisant un programme Scilab, la proportion d'enfants dans la population en 2010.

Exercice 3 le problème du fumeur (4 points)

Un fumeur cherche à réduire sa consommation de cigarettes. Au moment où il commencer l'expérience, il fume chaque jour avec une probabilité de 0,9. Il décide d'adopter la stratégie suivante :

Pour savoir s'il va fumer ou non au jour J_{n+1}, il regarde s'il a fumé au jour J_n :

Si oui, alors, il fume avec une probabilité de 0,8
Si non, il fume avec une probabilité de 0,3

Le fumeur cherche à savoir quelle sera sa probabilité de fumer à long terme chaque jour et à déterminer l'influence des probabilités de transition choisies.

1) En notant u_n (respectivement v_n) la probabilité de fumer (respectivement de ne pas fumer) au jour n, écrire une relation matricielle reliant les vecteurs colonnes $\begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix}$ et $\begin{pmatrix} u_n \\ v_n \end{pmatrix}$. Ecrire alors une fonction Scilab (appelée *fumeur*) permettant de calculer la probabilité pour l'individu de fumer au jour n.

2) Ecrire une fonction *affichage* pour permettre d'afficher de manière claire la probabilité de fumer (en pourcentage) pour l'individu du jour 0 au jour n.

3) Ecrire un programme *fumeur_indécis* en rajoutant aux fonctions précédente une courbe représentant en fonction du jour, la probabilité de fumer pour l'individu.

Exercice 4 le triangle de Pascal (6 points)

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1                   1
.
.
1       ….                      1
```

Voici comment on peut obtenir un tel triangle :

Des 1 à la première colonne. Des 1 sur la diagonale

Les autres valeurs sont obtenues ainsi : à partir de la troisième ligne, chaque valeur est la somme de celle qui est située juste au dessus avec celle qui la précède sur la même ligne.

1) On demande d'écrire la fonction [P]= triangle(n) qui, étant donné un entier *n*, construit le triangle de Pascal de *n* + 1 lignes. P devra être une matrice carrée d'ordre *n* + 1.
2) Ecrire la fonction afficher (P) qui devra effectuer l'affichage suivant de la matrice P :

```
1   *   *   *   *   *   *   *   *   *
1   1   *   *   *   *   *   *   *   *
1   2   1   *   *   *   *   *   *   *
1   3   3   1   *   *   *   *   *   *
1   4   6   4   1   *   *   *   *   *
1                   1   *   *   *   *
.                                   .
.
.                                   *
1       ….                          1
```

3) Le triangle de Pascal intervient, par exemple, dans le calcul de $(a + b)^n$.
Par exemple pour le calcul de $(a + b)^3$ les coefficients sont donnés par la ligne 4 du triangle de Pascal.
On obtient alors : $(a + b)^3 = a^3 + 3\,a^2\,b + 3\,ab^2 + b^3$
Ecrire une fonction *Puissance* qui :
Après avoir demandé à l'utilisateur trois entiers *a*, *b* et *n*, construit la matrice représentant le triangle de Pascal qui va permettre d'expliciter tous les termes de l'expression $(a + b)^n$.
Il faudra vérifier que l'entier *n* est inférieur ou égal à 12. L'affichage quant à lui devra être semblable à celui de l'exemple ci-dessus contenant les *.

Exercice 5 Codage de César (4 points)
Ce codage est le plus rudimentaire que l'on puisse imaginer. Il a été utilisé par Jules César (et même auparavant) pour certaines de ses correspondances. Le principe est de décaler les lettres de l'alphabet vers la gauche de 1 ou plusieurs positions. Par exemple, en décalant les lettres de 1 position, le caractère a se transforme en z, le b en a, ... le z en y. Le texte avecesar devient donc zudbdrzq.
1) Que donne le codage du texte *maitrecorbeau* en utilisant un décalage de 5 ?
2) Ecrire la fonction codageCesar(c,d) qui prend en arguments la chaîne *c* (que nous supposerons constituée uniquement de majuscules non accentuées) et un entier *d* ; et qui retourne une chaîne de même taille que *c* contenant le texte *c* décalé de *d* positions.
3) Ecrire de même la fonction decodageCesar(c,d) prenant les mêmes arguments mais qui réalise le décalage dans l'autre sens.

Exercice 1

```
u=0 ;     for k=0:n-1 do       u=u+1/(k+n); end
disp(u)
```

Exercice 2

1) $\begin{pmatrix} J_{n+1} \\ A_{n+1} \end{pmatrix} = \begin{pmatrix} \frac{4}{7} & \frac{2}{3} \\ \frac{1}{7} & \frac{3}{4} \end{pmatrix} \begin{pmatrix} J_n \\ A_n \end{pmatrix}$

2)
```
p=[2000 ;1000] ;
for i=1 :10 do
p= M*p;
disp('La population juvenile est de  ' +string(p(1)) +  ' individus')
disp('La population adulte est de  ' +  string(p(2)) +  ' individus' )
end
```

3)
```
p=[2000 ;1000] ;   M= [4/7  2/3; 1/7  3/4];
p=M^10*p;    j=p(1,1);   a=p(2,1);  disp( j/(a+j))
```

Exercice 3 le problème du fumeur

1) On obtient : $\begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} 0,8 & 0,3 \\ 0,2 & 0,7 \end{pmatrix} \begin{pmatrix} u_n \\ v_n \end{pmatrix}$

```
function [p] = fumeur_indecis(n)
M=[0.8 0.3 ; 0.2 0.7] ; p=[0.9 ;0.1] ;
for i=1:n do p=M*p; end
```

2)
```
function probabilites_fumeur
M=[0.8 0.3 ; 0.2 0.7] ; p=[0.9 ;0.1] ;
disp('La probabilité de fumer au jour n=0 est égale à 90 %')
for i=1:n do
p=M*p;
disp('La probabilité de fumer au jour' +  string(i) + ' est égale à ' + string(p(1,1)*100) + ' %')
end
endfunction
```

3)
```
function [t]=tableau_fumeur
n=input('entrez n') ;  M=[0.8 0.3 ; 0.2 0.7] ; p=[0.9 ;0.1];
t(1)=0.9 ;
for i=1:n do p=M*p;  t(i+1)=p(1,1), end
endfunction
```

Exercice 4 le triangle de Pascal

1)
```
function [p]= triangle(n)
p=zeros(n+1,n+1); p(1,1)=1;
for i=2:n+1 do p(i,1)=1;
for j=2:n+1 do p(i,j)=p(i-1,j-1)+p(i-1,j); end
end
endfunction
```

2) function afficher(p)
for i=1 :n+1 do
c='';
for j=1:n+1 do
if p(i,j)==0 then c=c+' * ';
else c=c+string(p(i,j)) + '   '; end
end
disp(c)
end
endfunction

3) function Puissance

n=13 ;
while n>12 do n= input('entrez le nombre n') ; end
p=triangle(n) ;
c= '(a + b)^' + string(n) + '= a^'+ string(n) + ' + ' ;

for j=2:n do
c=c + string(p(n+1,j))+ '*'+ 'a^'+ string(n-j+1) +'*b^' + string(j-1) + ' + ';
end
c=c+'b^' + string(n);
disp(c)
endfunction

Exercice 5 Codage de César
1) Le texte *maitrecorbeau* devient :
hvdomzxjmwzvp

function [e] = codageCesar(c,d)
if d>26 then d= modulo(d,26) ; end
e='' ;
for i=1 :length(c) ;
x = part(c,i) ; k=ascii(x) - d;
if k < ascii('A') then k=k+26 ; end
e=e+ascii(k);
end
endfunction

function [e] = decodageCesar(c,d)
if d>26 then d= modulo(d,26) ; end
e='' ;
for i=1 :length(c) ;
x = part(c,i) ; k=ascii(x)+d;
if k>ascii('Z') then k=k-26 ; end
e=e+ascii(k);
end
endfunction