Énoncés de l'APMEP : *www.apmep.fr*

ES - Centres Étrangers, juin 2018, exercice 2 (spé)

- affectation en parallèle
- while

- f_n la probabilité qu'un automate soit fonctionnelle n-ième jour ;
- s_n la probabilité qu'un automate soit en sursis le n-ième jour ;
- d_n la probabilité qu'un automate soit défaillant le n-ième jour.

On note alors $P_n = \begin{pmatrix} f_n & s_n & d_n \end{pmatrix}$ la matrice ligne de l'état probabiliste le n-ième jour.

Enfin, la société observe qu'au début de l'expérience tous ses automates sont fonctionnels : on a donc $P_0 = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$.

...

1. Justifier que pour tout entier naturel n, $s_{n+1} = 0{,}1 f_n + 0{,}8 s_n$.
2. On vérifierait de même que pour tout entier naturel n, $d_{n+1} = 0{,}2 s_n + d_n$ et $f_{n+1} = 0{,}9 f_n$ Compléter l'algorithme ci-dessous de sorte qu'il affiche le nombre de jours au bout duquel 30 % des automates ne fonctionnent plus.
3. Au bout de combien de jours la proportion d'automates défaillants devient-elle supérieure à 30 % ?
4. Dans le codage de la boucle « Tant que », l'ordre d'affectation des variables D, S et F est-il important ? Justifier.

le `print(N)` n'est pas nécessaire car Python renvoie la dernière valeur calculée... c'est ce qui a influencé la rédaction des aménagements de programmes ?

D ← 0
S ← ...
F ← 1
N ← 0
Tant que ...
    D ← 0,2 × S + D
    S ← 0,1 × F + 0,8 × S
    F ← 0,9 × F
    N ← ...
Fin Tant que
Afficher ...

```
# -*- coding:utf8 -*-
# python 3

D, S, F, N = 0, ... , 1, 0
while ..........
    D, S, F = .2 * S + D, .1 * F + .8 *
    ↪ S, .9 * F
    #D, F, S = .2 * S + D, .9 * F, .1
    ↪* F + .8 * S
    N = ..........
print ..........
```

S - Pondichéry, mai 2018, exercice 4 (spé)

- if
- for
- modulo (%)
- print

4. **a)** En déduire que $x(x+13) \equiv 3 \quad [33]$ équivaut aux quatre systèmes suivants :...
5. Compléter l'algorithme en **Annexe** pour qu'il affiche les quatre solutions trouvées dans la question précédente.

Annexe

Pour ..... allant de ..... à .....
    Si le reste de la division de ..... par ..... est égal à ..... alors
        Afficher .....
    Fin Si
Fin Pour

```
""" recherche des solutions de x*(x+13)=3 mod 33 pour 0<=x<33 """
for x ..........
    if ..........
        print..........
```

avec Xcas :
`solve((x*(x+13))%33=3,x)`

S - Centres Étrangers, juin 2018, exercice 1

$t \leftarrow 1{,}75$
$p \leftarrow 0{,}1$
$V \leftarrow 0{,}7$
Tant que $V > 0{,}035$
    $t \leftarrow t + p$
    $V \leftarrow (0{,}8t + 0{,}2)e^{-0{,}5t} + 0{,}03$
Fin Tant que

Quelle est la valeur de la variable t à la fin de l'algorithme ?

```
# -*- coding:utf8 -*-
# python 3

from math import exp

t, p, V = 1.75, .1, .7
while V > .035:
    t = t + p
    V = (0.8 * t + 0.2) * exp(-0.5 * t) + 0.0
    ↪3
print(t)
```

- while
- import de la librairie math
- print

S - Pondichéry, mai 2018, exercice 1

- while
- for ... in range
- def

Pour un nombre entier naturel n, on note T_n la température en degré Celsius du four au bout de n heures écoulées à partir de l'instant où il a été éteint. On a donc $T_0 = 1\,000$.

La température T_n est calculée par l'algorithme suivant :

$T \leftarrow 1\,000$
Pour i allant de 1 à n
    $T \leftarrow 0{,}82 \times T + 3{,}6$
Fin Pour

1. Déterminer la température du four, arrondie à l'unité, au bout de 4 heures de refroidissement.
2. Démontrer que, pour tout nombre entier naturel n, on a : $T_n = 980 \times 0{,}82^n + 20$.
3. Au bout de combien d'heures le four peut-il être ouvert sans risque pour les céramiques ?

```python
# -*- coding:utf8 -*-
# python 3

def temp(n):
    """ retourne la tempé
    ↪rature du four au
    bout de n heures
    """
    T = 1000
    for _ in range(n):
        T = .82 * T + 3.6
    return T
```

```python
def t_min():
    """ donne le nombre d'
    ↪heures minimum pour
    que la température
    ↪atteigne 70 degrés"""
    T, h = 1000, 0
    while T > 70:
        T = .82 * T + 3.6
        h = h + 1
    return h
```

`return` est une commande : les parenthèses ne sont pas utiles.
généralement `return` est en fin de fonction, mais ce n'est pas obligatoire ; une fonction peut ne pas avoir de commande `return`.

S - Asie, juin 2018, exercice 4

Compléter l'algorithme ci-après pour qu'il affiche tous les couples $(x\,;y)$ tels que :

$$\begin{cases} y^2 = 2x^2 + 1 \\ x \text{ et } y \text{ sont des nombres entiers} \\ 1 \leqslant x \leqslant 10 \text{ et } 1 \leqslant y \leqslant 10 \end{cases}$$

- print
- élévation à la puissance
- for ... range
- if

Pour x allant de 1 à ... faire
    Pour ...
        Si ...
            Afficher x et y
        Fin Si
    Fin Pour
Fin Pour

Lorsque l'on exécute cet algorithme, il affiche la valeur 2 pour la variable x et la valeur 3 pour la variable y.

```python
# -*- coding:utf8 -*-
# python 3

for x in range(1, ...):
    for ..........
        if ..........
            print("x = ",x," et y = "
    ↪,y)
```

S - Centres Étrangers (spé), juin 2017

```
m et n sont des entiers naturels non nuls et premiers entre eux
tant que m ≠ n faire
    si m < n alors
        Afficher « gauche »
        n prend la valeur n − m
    sinon
        Afficher « droite »
        m prend la valeur m − n
```

```python
# -*- coding:utf8 -*-
# python 3

n = int(input("n = "))
m = int(input("m = "))
while m != n:
    if m < n:
        print("gauche")
        n = n - m
    else:
        print("droite")
        m = m - n
```

- test !=
- while
- input() et conversion du type

S - Liban, juin 2017

Un numéro de carte bancaire est de la forme : $a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}a_{11}a_{12}a_{13}a_{14}a_{15}c$ où a_1, a_2, …, a_{15} et c sont des chiffres compris entre 0 et 9.

Les quinze premiers chiffres contiennent des informations sur le type de carte, la banque et le numéro de compte bancaire. c est la clé de validation du numéro. Ce chiffre est calculé à partir des quinze autres.

L'algorithme suivant permet de valider la conformité d'un numéro de carte donné.

```
I prend la valeur 0
P prend la valeur 0
R prend la valeur 0
pour k de 0 jusque 7 faire
    R prend la valeur du reste de la division euclidienne de 2a_{2k+1} par 9
    I prend la valeur I + R
pour k de 1 jusque 7 faire
    P prend la valeur P + a_{2k}
S prend la valeur I + P + c
si S est un multiple de 10 alors
    Afficher « Le numéro de la carte est correct. »
sinon
    Afficher « Le numéro de la carte n'est pas correct. »
```

```python
# -*- coding:utf8 -*-
# python 3

numCarte = input() # "5635400295613411"
# attention indices des chaînes commencent à 0

I, P, R = 0, 0, 0
for k in range(8):
    R = (2 * int(numCarte[2 * k])) % 9
    I = I + R
for k in range(7):
    P = P + int(numCarte[2*k + 1])
S = I + P + int(numCarte[15])
if S % 10 == 0:
    print("le numéro "+numCarte+" est valide")
else:
    print("le numéro "+numCarte+" n'est pas
    ↪valide")
```

- utilisation des chaîne : indice, opérateur + *
- indice des chaînes / des listes
- reste de la division euclidienne (%)
- if
- affectation en parallèle
- for in <liste>

Utiliser, modifier l'algorithme pour répondre aux questions (ce sont à peu près celles du sujet)

1. Justifier que le numéro de la carte 5635 4002 9561 3411 est correct.

2. On modifie le numéro de cette carte en changeant les deux premiers chiffres. Le premier chiffre (initialement 5) est changé en 6.
   Quel doit être le deuxième chiffre a pour que le numéro de carte obtenu $6a35400295613411$ reste correct ?
3. Un numéro de carte dont les chiffres sont tous égaux peut-il être correct ? Si oui, donner tous les numéros de carte possibles de ce type.

S - Amérique du Sud, septembre 2017

- for ... range
- module numpy
- calcul matriciel
- module sympy
- Rational

$U_n = (a_n ; b_n ; s_n)$ est la matrice qui à l'étape n donne la probabilité qu'un joueur soit dans l'équipe A (a_n) ou dans l'équipe B (b_n) ou soit solitaire (s_n).

On donne l'algorithme suivant, où la commande « U[i] » renvoie le coefficient de la i-ème colonne d'une matrice ligne U.

1 U **prend la valeur** $\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$
2 T **prend la valeur** $\begin{pmatrix} \frac{3}{5} & \frac{3}{20} & \frac{1}{4} \\ \frac{1}{5} & \frac{3}{5} & \frac{1}{5} \\ \frac{3}{14} & \frac{9}{14} & \frac{1}{7} \end{pmatrix}$
3 **pour** k **de** 1 **jusque** 7 **faire**
4 U **prend la valeur** UT
5 **Afficher** U[1]

```python
# -*- coding:utf8 -*-
# python 3

import numpy as np # permet le calcul
    ↪ matriciel

u = np.array([0, 0, 1])
t = np.array([[3/5, 3/20, 1/4],
              [1/5, 1/5, 3/5],
              [3/14, 9/14, 1/7]])
for k in range(7):
    u = np.dot(u, t)

print(u[0]) # attention indice
```

1. Quelle est la valeur numérique arrondie au millième de la sortie de cet algorithme ? L'interpréter dans le contexte de l'exercice.
2. Recopier et modifier cet algorithme pour qu'il affiche la fréquence de joueurs solitaires au bout de 13 jours.

Python a un module « sympy » qui permet d'effectuer des calculs avec les rationnels, les lettres...

```python
# -*- coding:utf8 -*-
# python 3

from sympy import * # permet le calcul littéral

u = Matrix([[0, 0, 1]]) # attention crochets
t = Matrix([[Rational(3,5), Rational(3,20), Rational(1,4)],
        [Rational(1,5), Rational(1,5), Rational(3,5)],
        [Rational(3,14), Rational(9,14), Rational(1,7)]])
for k in range(8):
    u = simplify(u * t)

print(u)
```

mais je ne dois pas le maîtriser ou bien, j'ai un parti-pris pour Xcas ;-)