TP Info 11 - corrigé

I 1 - Statistiques d'un tableau de données

$$\begin{cases} P(X=0) = p \\ P(X=1) = q = 1-p \end{cases}$$

1. $E(X) = \sum_i p_i x_i = p \times 0 + (1-p) \times 1 = 1-p$

2. $V(X) = \sum_i p_i (x_i - E(X))^2 = \sum_i p_i x_i^2 - 2E(X) \underbrace{\sum_i p_i x_i}_{E(X)} + E(X)^2$.

$$V(X) = E(X^2) - E(X)^2$$

Ici $E(X^2) = p \times 0^2 + (1-p) \times 1^2 = 1-p$

$E(X)^2 = (1-p)^2$

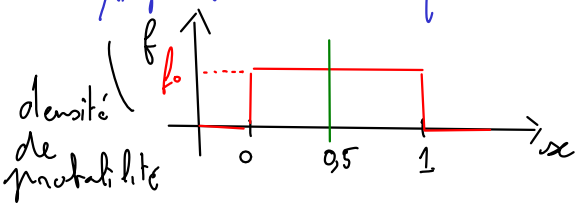
$V(X) = 1-p-(1-p)^2$

$= (1-p)[1-(1-p)]$

$$\boxed{V(X) = p(1-p)}$$

3.
```
def epreuve(p):
    res=uniform(0,1)
    if res<=p:
        return 0
    else:
        return 1
```

La fonction uniform renvoie de manière équiprobable un flottant $< 0,5$ et $> 0,5$



4.
```
def serie_epreuves(p,N):
    res=[]
    for i in range(N):
        r=epreuve(p)
        res.append(r)
    return res
```

De façon triviale, appelle N fois la fonction epreuve.

5.
```
def moyenne(T):
    N=len(T)
    s=0
    for i in range(N):
        s=s+T[i]
    m=s/N
    return m
```

Complexité : $O(N)$

6.
```
def variance(T):
    N=len(T)
    sc=0
    for i in range(N):
        sc=sc+T[i]**2
    m=moyenne(T)
    v=sc/N-m**2
    return v
```

Complexité : $O(N)$

$$V = \frac{1}{N} \sum_{i=1}^{N} (x_i - m)^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} x_i^2 - m^2$$

7. Appel des fonctions pour p et N à fixer

```
p=
N=
l=serie_epreuves(p,N);
m=moyenne(l);print(m)
v=variance(l);print(v)
```

) feedback

Exemple : $p = 1/10$

Valeur de la variance et de l'espérance :

$E(X) = p = 0,1$, $V(X) = p(1-p) = 0,09$

Résultat du calcul

N	moyenne	variance
10^2	0,9600	0,0384
10^4	0,9015	0,0888
10^6	0,9001	0,0898
10^8		

$N \to +\infty$: la moyenne → espérance.
la variance statistique → variance probabiliste.

I 3. Nombre de mots dans un texte

1/ Algorithme.

Idée : compter le nombre d'espace.

```
def nb_mots(t):
    """
    Dénombre les mots du texte t
    t : texte type:string
    nb : nombre de mots, type:int
    """
    nb=0
    for i in range(len(t)):
        if t[i]==" ":
            nb=nb+1
    nb=nb+1#nb_mot=nb_espace+1
    return nb
```

car pas d'espace à la fin du texte

Nombre de mots du texte ≈ nombre d'espace.

← Solution un peu simpliste.

On prend en compte les sauts de ligne, les espaces multiples ...

```
def nb_mots_mieux(t):
    """
    Dénombre les mots du texte t
    t : texte type:string
    nb : nombre de mots, type:int
    """
    nb=0
    for i in range(len(t)-1):
        if t[i] not in " \n\t" and t[i+1] in " \n\t":
            nb=nb+1
    nb=nb+1#nb_mot=nb_espace+1
    return nb
```

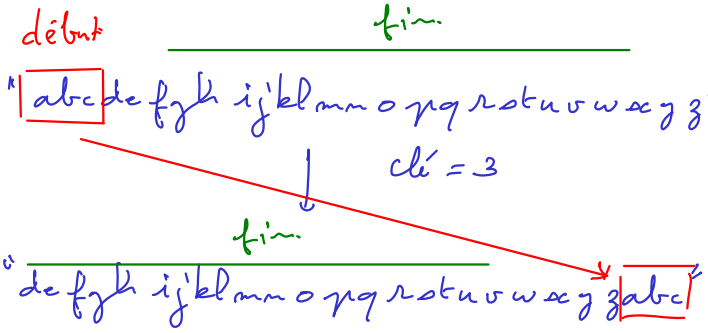
Un mot est terminé si un caractère est suivi d'un espace au sens général.

Evidemment, on a supposé que le texte respectait les normes d'écriture.

I2 - Code de César.

1.

```
def permutation_circulaire(alphabet,cle):
    n=len(alphabet)
    cle=cle%n#si cle>n
    debut=alphabet[0:cle]
    fin=alphabet[cle:n]
    alphabet_perm=fin+debut
    return alphabet_perm
```



Complexité en temps : $O(n)$ (lecture / écriture d'un caractères)
Complexité en mémoire : $O(n)$ (car il faut stocker en mémoire la chaîne de caractères alphabet_perm)

2.

```
def chiffre_cesar(alphabet,message,cle):
    alphabet_perm=permutation_circulaire(alphabet,cle)
    message_code=""
    for lettre in message:
        i=rech_pos(alphabet,lettre) # à écrire a posteriori
        if i!=None:
            message_code=message_code+alphabet_perm[i]
        else:
            message_code=message_code+lettre
    return message_code
```

Faire la correspondance entre chaque lettre du message et la lettre de substitution : elles ont même indice dans alphabet et alphabet_perm

Complexité : $O(mn)$

Rem : un dictionnaire serait une structure de données mieux adaptée au problème.

```
def rech_pos(s,elt):
    """
    s : sequence sans doublon
    """
    n=len(s)
    for i in range(n):
        if s[i]==elt:
            return i
    return None
```

Trivial.
Complexité : $O(n)$

3.
```
def dechiffre_cesar(alphabet,message,cle):
    alphabet_perm=permutation_circulaire(alphabet,cle)
    message_decode=""
    for lettre in message:
        i=rech_pos(alphabet_perm,lettre)
        if i!=None:
            message_decode=message_decode+alphabet[i]
        else:
            message_decode=message_decode+lettre
    return message_decode
```

Identique à chiffre_cesar en permutant les rôles de alphabet_perm et alphabet.
Complexité : $O(mn)$ où $m = \text{len(message)}$, $n = \text{len(alphabet)}$

De manière plus concise, on peut écrire :

```
def dechiffre_cesar(alphabet,message,cle):
    alphabet_perm=permutation_circulaire(alphabet,cle)
    message_decode=chiffre_cesar(alphabet_perm,message,-cle)
    return message_decode
```

$\neq$ déchiffrer = chiffrer à l'envers (clé → − clé)

4. Casser un code par force brute consiste à essayer toutes les possibilités.

```
def casse_cesar(alphabet,message):
    sortie=[]
    for cle in range(0,len(alphabet)):
        tentative=dechiffre_cesar(alphabet,message,cle)
        sortie.append(tentative)
    return sortie
```

m fois ; $O(1)$; $O(mn)$

Complexité : $O(m^2 n)$

Conclusion : le code de César peut-être cassé en un temps polynômial par force brute $\Rightarrow$ pas sûr !!!