TD et TP1

Prise en main de Python et des classes 603

Consigne importante valable pour toutes vos productions ;

Pour pouvoir faire fonctionner les tests des docString de chaque fonction, ajouter au début « du code main » :

```
if  __name__== "__main__":
  import doctest
  doctest.testmod()
```

Toute les fonctions produites en Info0603 devront comporter au moins un test rédigé **avant** l'écriture du code. Cette pratique permet ainsi d'avancer sa réflexion tout en documentant le code.

C'est aussi un bon canal de communication avec son binôme ou le professeur.

Note : pour gagner du temps, reprenez le fichier « arithmetiqueDansZEtd.py » et remplacer par du code les `raise NotImplementedError`.

Exercice 1: Arithmétique élémentaire

Implémenter les fonctions Python suivantes en reprenant les docStrings proposées dans votre propre code.

```
def secondDiviseur(a):
    """Renvoie le premier diviseur positif de a supérieur à 1
    >>> secondDiviseur(15845465)
    5
```

```
def eDiviseurs(a):
  """renvoie l'ensemble des diviseurs positifs de a
  >>> eDiviseurs(60)
  {1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 60, 30}
  >>> eDiviseurs(1)==set({1}) and eDiviseurs(13)==set({1, 13})
  True
```

Evaluer la taille des entiers donnant lieu à des calculs d'une durée dépassant 10 secondes.

Exercice 2: Calculs en modulo avec la classe {`ElmtZnZ`}

La classe `ElmtZnZ(object)` inclu les méthodes suivantes :
__init__; __str__ (qui renvoie une chaine lisible) ; __repr__ (qui renvoie une chaine correspondant à l'appel au constructeur nécessaire pour créer un objet identique) et __eq__ .

A la main ! Compléter les expressions suivantes :

```
elmtZnZ(2,256)+elmtZnZ(2,256)==elmtZnZ(….,256)
elmtZnZ(2,256)+elmtZnZ(255,256)==elmtZnZ(….,256)
elmtZnZ(2,256)-elmtZnZ(3,256)==elmtZnZ(….,256)
(elmtZnZ(-2,256)==elmtZnZ(254,256))==…….
130*elmtZnZ(2,256)==elmtZnZ(….,256)
elmtZnZ(2,256)*elmtZnZ(128,256)==elmtZnZ(…….,256)
elmtZnZ(10,256)**3==elmtZnZ(…….,256)
elmtZnZ(255,256)**10==elmtZnZ(…….,256)
elmtZnZ(254,256)**4==elmtZnZ(…….,256)
elmtZnZ(3,10)*elmtZnZ(...,10)==elmtZnZ(1,256)
elmtZnZ(2,10)*elmtZnZ(….,10)==elmtZnZ(1,10)
elmtZnZ(2,256)*elmtZnZ(….,256)==elmtZnZ(1,256)
```

`__add__` (qui peut additionner à un autre elmtEnz ou à un entier); `__radd__`; `__mul__` ; `__rmul__` ; `__floordiv__` ; `__neg__` ; `__sub__` , `__rsub__` . Cela en limitant le code redondant et en favorisant les appels aux fonctions déjà programmées.
Puis: `estInversible` , `inverse` et enfin `__pow__` que l'on veillera à optimiser tant elle est utile en cryptographie.

et enfin:

```
logDiscret(self,b):
  """Renvoie x tel que self.a**x==b(self.n) n doit être premier pour
  garantir l'existence
  >>> elmtZnZ(2,13).logDiscret(8)
  3
  >>> elmtZnZ(2,13).logDiscret(3)
  4
```

Description complète des fonctions magiques : https://docs.python.org/fr/3/reference/datamodel.html
Exemples d'exceptions : https://pythonbasics.org/try-except/

Note : cette classe devra être entièrement terminée car elle sera utilisée par la suite comme toutes les classes données à faire en TP.

Exercice 3: La classe Binaire603

Cette classe hérite de `list` et lui ajoute des méthodes afin que nous puissions faire facilement de la cryptographie.

Entre autre on trouve le constructeur appelable sous des formes variées : liste d'int, liste de `ElementDeZnZ,` chaine de caractère. On peut aussi en instancier à partir d'un fichier txt ou d'une image BMP et les sauvegarder comme fichier txt ou binaire.

Aperçu des méthodes :

```
class Binaire603(list):
    def __init__(self,param):
        """lbin est une liste ne contenant que :
        des entiers de [0..255] qui sont donc assumés comme des octets"
        ou bien une liste de ElementDeZnZ(..,256)
        lbin peut aussi être une chaine de caractère
        >>> Binaire603([ElementDeZnZ(2,256),3])
        Binaire603([ 0x02, 0x03])
        >>> Binaire603("Trop tôt !")
        Binaire603([ 0x54, 0x72, 0x6f, 0x70, 0x20, 0x74, 0xf4, 0x74, 0x20, 0x21])

    def toString(self):
        """
        >>> Binaire603("Trop tôt !").toString()
        'Trop tôt !'

    def sauvegardeDansFichier(self,nomFic):

    def bin603DepuisFichier(nomFic,verbose=False):
        """renvoie un Binaire603 d'après les données d'un fichier
      >>> b1=Binaire603.exBin603(taille=10,num=5)
      >>> b1.sauvegardeDansFichier("MonBin.bin")
      >>> b2=Binaire603.bin603DepuisFichier("MonBin.bin")
      >>> print(b2)
      10 octets :00ff00ff00ff00ff00ff
      >>> b=Binaire603.bin603DepuisFichier("les miserables UTF8.TXT")

     def exBin603(taille=1000,num=0):
```

```
        """Renvoie une instance Exemples de cette classe avec pour num :
        0: 255 fois 255 puis 254 fois 254 etc...
      1: un octet de chaque
      2: octets aléatoires
      3: 254 13 puis 255 14 puis 254 13 puis 255 14 ...
      4: 254 13 puis 255 14 puis 256 15 répétés
      5: et plus: 1000 fois un octet sur deux à 255

   def estOctet(val):
        Renvoie true si val est convertible en octet
        >>> Binaire603.estOctet(255) and not(Binaire603.estOctet(256))
        True
        >>>not(Binaire603.estOctet(-1))and not(Binaire603.estOctet("coucou"))
               and not(Binaire603.estOctet([128,12]))
        True
```

```
def ajouteOctet(self,data):
    Ajoute un octet ou une liste d'octets tout en vérifiant la validité des données
  >>> a=Binaire603([1,2,3,4]); a.ajouteOctet(10); a
  Binaire603([ 0x01, 0x02, 0x03, 0x04, 0x0a])
  >>> a.ajouteOctet([11,12]);a
  Binaire603([ 0x01, 0x02, 0x03, 0x04, 0x0a, 0x0b, 0x0c])

def __str__(self):
    >>> str(Binaire603([12,128]))
  '2 octets : 0c80'
def __repr__(self):
    Renvoie une chaine de caractère représentant TOUTES les données du Binaire603 self
  sous la forme d'un appel à son constructeur ex: Binaire603([ 0x57, 0x26, 0xfd])

def __eq__(self,other):
    Renvoie True lorsque les octets sont égaux deux à deux
     >>> Binaire603([ 0xcb, 0xba])==Binaire603([ 0xcb, 0xba])
     True
     >>> Binaire603([ ElementDeZnZ(1,256), ElementDeZnZ(2,256)])==Binaire603([1,2])
     True
     >>> Binaire603([ 0xcb, 0xba])==Binaire603([ 0xcb])
     False
     >>> Binaire603([ 0xcb, 0xba])==Binaire603([ 0xcb, 0xbb])
     False

def lFrequences(self):
    "Renvoie la liste des fréquences de chaque valeur


def afficheTableauDesFrequences(self):
def afficheTableauDesFrequencesDecroissantes(self):
def afficheHistogrammeDesFrequences(self):

def nbOctets(val):
    Renvoie le nb d'octets nécessaire pour coder l'entier val
def ajouteLongueValeur(self,val):
    Ajoute une valeur entière de taille quelconque de telle façon qu'elle
  puisse être récupérable par lisLongueValeur

def lisLongueValeur(self,pos):
    Renvoie tout ce qu'il faut pour enregistrer/ajouter une valeur entière de taille
     quelconque"à         la         position         pos
     Renvoie cette valeur et la nouvelle valeur de pos
  >>> monBin=Binaire603([12,13])
  >>> monBin.ajouteOctet(15)
  >>> monBin.ajouteLongueValeur(1000)
  >>> monBin.ajouteLongueValeur(100000)
  >>> pos=2
  >>> v0,pos=monBin.lisOctet(pos)
  >>> v1,pos=monBin.lisLongueValeur(pos)
  >>> v2,pos=monBin.lisLongueValeur(pos)
  >>> v0,v1,v2
```

Exercice 4: Premiers chiffreurs

Ecrire la classe ChiffreurParDecalage durant le TD, compléter son code durant le TP.

Ecrire la classe ChiffreurVigenere durant le TD, compléter son code durant le TP.

Proposer une méthode pour déchiffrer les documents DocChiffre1, DocChiffre2, DocChiffre3, DocChiffre4 puis les déchiffrer.

2. Eléments de Correction

Exercice 1: Dqs

Exercice 2: Dqs

Exercice 3: dsq

Exercice 4:

Le fichier chiffrage.py a créé les documents chiffrés et dechifrage montre comment déchiffré

Exercice 5: