TP n° 1

Arithmétique et Prise en main de Python

note : Pour pouvoir faire fonctionner les tests des docString de chaque fonction, ajouter en dernière ligne de votre module :

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

Exercice 1 (Arithmétique élémentaire)
Implémenter les fonctions Python suivantes en reprenant les docStrings proposées dans votre propre code :

Division euclidienne : $\forall(a,b) \in \mathbb{Z} \times \mathbb{N}, \exists!(p,q)\mathbb{Z} \times \mathbb{N}$ tels que $a = b.q + r$ avec $r < b$

```
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}
True
>> eDiviseurs(1)==set({1}) and eDiviseurs(13)==set({1, 13})
True
```

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

Note : pour ce faire on pourra ajouter systématiquement dans une fonction **demoVitesse** une boucle comme celle-ci après la programmation de chaque fonction importante :

```
def demoVitesse():
    for p in range(5,20):
         for k in range(100):
            ld=eDiviseurs(10**p+k)
            if len(ld)<=3 or len(ld)>20:
                print(f"{p}:eDiviseurs(10**p+k)=={eDiviseurs(10**p+k)}")
```

Algorithme d'Euclide :

```
def PGCD(a,b): >> PGCD(360,304)
8
>> PGCD(517,513)==1 and PGCD(513,517)==1
True
```

```
def bezout(a,b):
"""Renvoie (u,v,d) tel que a.u+b.v=d avec d=PGCD(a,b)
>> bezout(360,304)
(11, -13, 8)
>> bezout(1254,493)
(-149, 379, 1)
>> bezout(513,517)
(129, -128, 1)
```

Th fondamentale de l'arithmétique : Tout entier strictement positif peut être écrit comme un produit de nombres premiers d'une unique façon, à l'ordre près des facteurs.

```
def chFacteursPremiers(n):
"""renvoie une chaine de caractère donnant la décomposition en
facteurs premiers de n
>> chFacteursPremiers(120)
'2^3 × 3 × 5'
>> chFacteursPremiers(3600)
'2? × 3^2 × 5^2'
>> chFacteursPremiers(1)
'1'
>> chFacteursPremiers(2)
'2'
>> chFacteursPremiers(21)
'3 × 7'
```

Evidemment on pourra se contenter d'un retour "moins beau".

On pourra aussi programmer (optionnel) :

```
def lDecompoPGCDetPPCM(a,b):
"""Renvoie le couple de listes de la décomposition en facteurs
premiers du PGCD et du PPCM de a et b
en utilisant la décomposition en facteurs premier de a et b
»> lDecompoPGCDetPPCM(60,700)
[(2, 2),(5, 1)], [(2, 2), (5, 2), (7, 1)]
```

Indicatrice d'Euler : $\varphi(n) = \text{card}\{m \in \mathbb{N}^* \mid m \leq n \text{ et } m \text{ premier avec } n\}$ et on a $\varphi(n) = \prod_{i=1}^{r}(p_i - 1)p_i^{k_i-1} = n\prod_{i=1}^{r}\left(1 - \frac{1}{p_i}\right)$

Exercice 2 (Calculs en modulo avec la classe ElmtZnZ)
Def : $a \equiv b(n) \Leftrightarrow a\%n = b\%n$

Vous allez développer la classe **ElmtZnZ(object)** avec les mêmes pratiques que ci-dessus (avec des tests systématiques dans les docStrings) et en programmant ses méthodes dans l'ordre ci-dessous :

__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__**.

__add__ (qui peut additionner à un autre elmtEnz ou à un entier) ; **__radd__ ; __mul__ ; __rmul__ ; __floordiv__ ; __neg__ ; __sub__ et __rsub__** . Cela en limitant le code redondant et en favorisant les appels aux fonctions déjà programmées.

Ajouter ensuite le Théorème chinois : Soit $(p, q) \in \mathbb{Z}^2$ tel que p et q soient premiers entre eux alors

$\forall (a, b) \in \mathbb{Z}^2 \exists c \in \mathbb{Z}$ tel que : $x \equiv a(p)$ et $x \equiv b(q) \Leftrightarrow x \equiv c(p.q)$

```
def valThChinois(self,other):
""" »> ElmtZnZ(2,7).valThChinois(ElmtZnZ(3,10))
ElmtZnZ(23,70)
```

Ajouter les fonctions **estInversible**(self) et **inverse**(self) puis enfin **__pow__**(que l'on veillera à optimiser tant elle peut être utile en cryptographie).

et enfin :

```
def 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.