Contrôle NSI n°1

Énoncé

Exercice 1 (5 pts)

1. Écrire une fonction est_premier qui prend en paramètre un nombre entier et renvoie True si ce nombre est premier et False sinon. Un nombre premier est un nombre qui ne peut être divisé que par 1 et par lui-même. 1 n'est pas un nombre premier. (2 pts)
2. Écrire une fonction premiers qui prend en paramètre un nombre entier et renvoie la liste de tous les nombres premiers inférieurs strictement à ce nombre. (3 pts)

```
def est_premier(n):
    """
    renvoie true si l'entier est premier, false sinon
    param : n: int
    return: bool
    exemples:
    >>> est_premier(13)
    True
    >>> est_premier(4)
    False
    """
    for i in range(2,n):
        if n%i==0:
            return False
    return True


def premiers(n):
    """
    renvoie la liste des nombres premiers strictement inférieurs à un entier
    param : n: int
    return : list
    exemples:
    >>> premiers(13)
    [2, 3, 5, 7, 11]
    """
    liste=[]
    for i in range(2,n):
        if est_premier(i):
            liste.append(i)
    return liste
```

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

Exercice 2 (5 pts)

1. Écrire une fonction lettres_id_mot qui prend en argument un mot et renvoie True si le mot commence ou se termine par la même lettre et False sinon. (2 pts)
2. Écrire une fonction lettres_id_mots qui prend en argument deux mots et renvoie True si les deux mots commencent par la même lettre et se terminent par la même lettre et False sinon. (3 pts)

```
def lettres_id_mot(mot):
    """
    renvoie True si le mot commence et se termine par la même lettre, False sinon
    param : mot : str
    return: bool
    exemples:
    >>> lettres_id_mot("exemple")
    True
    >>> lettres_id_mot("ma")
    False
    """
#    if mot[0]==mot[len(mot)-1]:
#        return True
#    else:
#        return False
    return mot[0]==mot[len(mot)-1]

def lettres_id_mots(mot1,mot2):
    """
    renvoie True si les deux mots commencent par la même lettre ou finissent par la m
    param : mot1 : str
    param : mot2 : str
    return : bool
    exemples:
    >>> lettres_id_mots("lampe","canne")
    False
    >>> lettres_id_mots("lampe","lune")
    True
    """
#    if (mot1[0]==mot2[0]) and (mot1[-1]==mot2[-1]):
#        return True
#    else:
```

```
#         return False
    return (mot1[0]==mot2[0]) and (mot1[-1]==mot2[-1])


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

Exercice 3 (5 pts)

Écrire une fonction stat qui prend en paramètre un texte et renvoie un dictionnaire statistiques dont les clés sont les différentes lettres du texte et les valeurs le nombre d'occurrences de chaque lettre dans le texte. Le texte peut contenir des espaces ou des caractères de ponctuation qui ne devront pas être comptabilisés dans le dictionnaire fourni par la fonction.

Indications : parcourir les lettres du textes, créer s'il n'existe pas encore le nouvel item (lettre:1) dans le dictionnaire statistiques, ou augmenter de 1 sa valeur si l'item existe déjà.

```
def stat(texte):
    """
    renvoie un dictionnaire donnant pour chaque lettre son nombre d'occurence sans te
    param : texte : str
    return : dict
    exemples:
    >>> stat("exem;ple")
    {'e': 3, 'x': 1, 'm': 1, 'p': 1, 'l': 1}
    """
    dictionnaire={}
    ponctuations=[',',':',';','!','?','.',' ']
    for i in texte:
        if i not in ponctuations:
            if not dictionnaire.get(i):
#si on fait if not dictionnaire[i]: on a une erreur lorsqu'il ne trouve pas i dans le
                dictionnaire[i]=1
            else:
                dictionnaire[i]+=1
    return dictionnaire

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

Exercice 4 (5 pts)

On utilise 5 bits pour coder en binaire les entiers relatifs.

1. Comment code-t-on le nombre 9 ? (1 pt)

```
>>> 0b01001
9
```

2. Comment code-t-on le nombre -10 ? (2 pts)

```
>>> 0b01010
10
```

Première méthode : on prend le complément à deux de $(01010)_2$=10, pour obtenir (10101) puis on ajoute 1 pour obtenir $(10110)_2$= - 10.
Deuxième méthode : on peut aussi calculer 2^5-10=22, ce qui se code $(10110)_2$.

3. Si on utilise 5 bits pour coder les entiers relatifs, combien de nombres peut-on coder et lesquels ? (2 pts)

Sur 5 bits, on peut coder 2^5=32 nombres, ce sont les entiers relatifs compris entre $(10000)_2$ = -16 et $(01111)_2$ = +15.