```python
#Correction du TME Thème 5, MIPI 16.5
#Exercice 10 Générateur de phrase

#Question 1
import random # pour le générateur aléatoire
import math
def choix_parmi(n):
    """int -> int
    retourne un entier choisi aléatoirement dans
    l'intervalle [1;n]."""
    return math.floor(n*random.random()) + 1

# Jeu de tests
# Les tests avec la fonction random ne seront pas demandés à l'examen.
# En répétant sur un petit intervalle, on peut tout de même écrire un test
    pert assert (1 <= choix_parmi(3)) and (choix_parmi(3) <= 3)
assert (1 <= choix_parmi(5)) and (choix_parmi(5) <= 5)
assert (1 <= choix_parmi(5)) and (choix_parmi(5) <= 5)
assert (1 <= choix_parmi(5)) and (choix_parmi(5) <= 5)
assert (1 <= choix_parmi(5)) and (choix_parmi(5) <= 5)
assert (1 <= choix_parmi(5)) and (choix_parmi(5) <= 5)
assert (1 <= choix_parmi(5)) and (choix_parmi(5) <= 5)



#Question 2

def sujet(n):
    """int -> str
    Hypothèse: n > 0
    renvoie un sujet de phrase."""
    if n == 1:
        return 'le chat'
    elif n == 2:
        return 'la fleur'
    elif n == 3:
        return 'le voisin'
    elif n == 4:
        return "l'infirmière"
    else:
        return "l'aspirateur"


# Jeu de tests
assert sujet(2) == 'la fleur'
assert sujet(4) == "l'infirmière"
assert sujet(12) == "l'aspirateur"

def choix_sujet():
    """ -> str
    renvoie un sujet de phrase."""
    return sujet(choix_parmi(5)) # uniforme car exactement 5 choix

# Jeu de tests : pas de test facile à cause du random

#Question 3

def verbe(n):
    """int -> str
```

```
    Hypothèse: n > 0
    renvoie un sujet de phrase."""
    if n == 1:
        return 'mange'
    elif n == 2:
        return 'prend'
    elif n == 3:
        return 'imagine'
    elif n == 4:
        return "cultive"
    else:
        return "assemble"

# Jeu de tests
assert verbe(2) == 'prend'
assert verbe(4) == "cultive"
assert verbe(12) == "assemble"

def choix_verbe():
    """ -> str
    renvoie un sujet de phrase."""
    return verbe(choix_parmi(5)) # uniforme car exactement 5 choix

def complement(n):
    """int -> str
    Hypothèse: n > 0
    renvoie un sujet de phrase."""
    if n == 1:
        return 'le chat'
    elif n == 2:
        return 'la fleur des bois'
    elif n == 3:
        return "l'escalier"
    elif n == 4:
        return "du concombre pas frais"
    else:
        return "du chocolat noir"

# Jeu de tests
assert complement(2) == 'la fleur des bois'
assert complement(12) == "du chocolat noir"

def choix_complement():
    """ -> str
        renvoie un sujet de phrase."""
    return complement(choix_parmi(5)) # uniforme car exactement 5 choix


def phrase():
    """ -> str
    renvoie une phrase."""
    # sujet : str
    sujet=choix_sujet()
    # verbe : str
    verbe=choix_verbe()
    # complement : str
    complement=choix_complement()
    return sujet + ' ' + verbe + ' ' + complement + '.'
```

```
# EXERCICE 6 Brins d'ADN

# Question 1
def base_comp(base):
    """ str -> str
    Hypothèse : base est l'un des caractères suivant : A, T, C, G
    renvoie la base azotée complémentaire de base."""
    if base == 'A':
        return 'T'
    elif base == 'T':
        return 'A'
    elif base == 'C':
        return 'G'
    else:
        return 'C'

# Jeu de tests
assert base_comp('A') == 'T'
assert base_comp('T') == 'A'
assert base_comp('C') == 'G'
assert base_comp('G') == 'C'

#Question 2
def brin_comp(brin):
    """ str -> str
    Hypothèse : brin est composé uniquement des caractères A, T, C, G
    renvoie le brin complémentaire de brin."""
    # comp : str
    comp = '' # le brin complémentaire
    # b : str  (base courante)
    for b in brin:
        comp = comp + base_comp(b)
    return comp

#jeux de tests
assert brin_comp('ATCG') == 'TAGC'
assert brin_comp('ATTGCCGTATGTATTGCGCT') == 'TAACGGCATACATAACGCGA'
assert brin_comp('') == ''
assert brin_comp('AAAA') == 'TTTT'


#Question 3, solution avec sortie anticipée de fonction
def test_comp(b1,b2):
    """ str * str -> bool
    Hypothèse : b1 et b2 sont composés uniquement des caractères A, T, C, G
    renvoie True si et seulement si b1 et b2 sont complémentaires."""
    # i : int
    i = 0 # compteur
    if len(b1) != len(b2): # si les deux brins sont de longueur différente
        return False
    else:
        while i < len(b1):
            if b1[i] != base_comp(b2[i]):
                return False # on a trouvé deux bases non complémentaires
```

```
            i=i+1
        return True # si on sort de la boucle, c'est que toutes les bases sont
            complementaires

    #jeux de tests
    assert test_comp('','')
    assert not test_comp('','ATCG')
    assert not test_comp('ATCG','')
    assert test_comp('ATCG', 'TAGC')
    assert not test_comp('ATCG', 'TAAG')
    assert test_comp('ATTGCCGTATGTATTGCGCT','TAACGGCATACATAACGCGA')


    #Question 4
    def test_sous_sequence(b1,b2):
        """ str * str -> bool
        Hypothèse : b1 et b2 sont composés uniquement des caractères A, T, C, G
        renvoie True si b1 est une sous-séquence de b2."""
        # i : int
        i = 0 # compteur
        # n : int
        n = len(b1) # longueur du brin b1
        # res : bool
        res = False # valeur de retour
        # sseq : str
        sseq='' # sous-séquence de b2 à tester
        if n==0:
            return True
        else:
            while i < len(b2) - n + 1:
                sseq = b2[i:i+n] # on récupère la prochaine sous-chaine de b2 à
                    tester
                if b1 == sseq:
                    res = True # on a trouvé une correspondance
                i=i+1
            return res

    #jeux de tests
    assert test_sous_sequence('','')
    assert test_sous_sequence('','ATCG')
    assert not test_sous_sequence('ATCG','')
    assert test_sous_sequence('GC', 'TAGC')
    assert not test_sous_sequence('GC', 'TAAG')
    assert test_sous_sequence('CA','TAACGGCATACATAACGCGA')


    #Question 5
    def recherche_sous_sequence(b1,b2):
        """ str * str -> int + NoneType
        Hypothèse : b1 et b2 sont composés uniquement des caractères A, T, C, G
        renvoie l'indice de b2 correspondant au début de b1 si b1 est une sous-sé
            quence ou None si b1 n'est pas une sous-séquence de b2."""
        # i : int
        i = 0 # compteur
        # n : int
        n = len(b1) # longueur du brin b1
        # sseq : str
        sseq='' # sous-séquence de b2 à tester
        if n==0:
```

```
            return 0
        else:
            while i < len(b2) - n + 1:
                sseq = b2[i:i+n] # on récupère la prochaine sous-chaine de b2 à
                    tester
                if b1 == sseq:
                    return i # on a trouvé une correspondance, on renvoie l'indice
                        qui correspond
                i=i+1
            return None # si on a parcouru tous les indices
                        # alors la séquence b1 n'est pas une sous-séquence

    #jeux de tests
    assert recherche_sous_sequence('','') == 0
    assert recherche_sous_sequence('','ATCG') == 0
    assert recherche_sous_sequence('ATCG','') == None
    assert recherche_sous_sequence('GC', 'TAGC') == 2
    assert recherche_sous_sequence('GC', 'TAAG') == None
    assert recherche_sous_sequence('CATA','TAACGGCATACATAACGCGA') == 6
```