```python
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 14 23:03:48 2015

@author: dconduche
"""


"""Exercice 3"""


def mystere(n):
    resultat = ""
    while n > 0:
        resultat = str(n % 2) + resultat
        n = n // 2
    return resultat

"""1)"""
print(mystere(13))

"""Cette fonction décompose en base 2 un nombre n. Elle retourne une chaîne
de caractères."""

"""2)"""
"""Il y a un problème si n = 0. Une façon de réparer ce bug est de tester
le cas n = 0. On peut aussi tester si n < 0 et retourner une erreur, ou
proposer d'étendre la fonction aux n < 0"""


def base2_debug(n):
    if n == 0:
        return "0"
    resultat = ""
    while n > 0:
        resultat = str(n % 2) + resultat
        n = n // 2
    return resultat

"""3)"""
"""On teste d'abord la condition de sortie, puis sort le chiffre des unités
(n % 2) et on rappelle la fonction sur le quotient (n // 2)."""


def base2_rec(n):
    if n in [0, 1]:  # Condition de sortie
        return str(n)
    return str(n % 2) + base2_rec(n // 2)

"""4)"""


def baseN_rec(n, N=2):
    if n in range(N):
        return str(n)
    return str(n % N) + base2_rec(n // N)


"""Exercice 4"""
```

```python
def Parenthesage_entier(chaine):
    """On compte +1 pour une parenthèse ouvrante, -1 pour une fermante.
    La chaîne est bien parenthésée si et seulement si :
        n == 0 à la fin du parcourt de la chaîne;
        et n n'est jamais négatif pendant le parcours

    Comme il n'y a qu'un type de parenthèse, pas de problème d'entrelacement.
    """
    n = 0
    for x in chaine:
        if x == '(':
            n = n+1
        elif x == ')':
            n = n-1
            if n < 0:
                return False
    return n == 0


def Parenthesage_pile(chaine):
    pile = []
    for x in chaine:
        if x == '(':
            pile.append(x)  # On empile
        elif x == ')':
            if len(pile) == 0:  # On veut dépiler alors qu'il n'y a rien...
                return False
            else:
                pile.pop()  # On dépile
    return len(pile) == 0


def Parenthesage_complexe(chaine):
    """Pour ceux qui veulent aller plus loin, la bonne façon de le faire
    est d'utiliser un dictionnaire.
    Dans tous les cas, pour la lisibilité de la fonction, mettre à part
    la liste des parenthèses."""
    liste_parentheses = [['(', ')'], ['{', '}'], ['[', ']']]
    ouvrantes = [couple[0] for couple in liste_parentheses]
    fermantes = [couple[1] for couple in liste_parentheses]
    pile = []
    for x in chaine:
        if x in ouvrantes:
            pile.append(x)  # On empile
        elif x in fermantes:
            if len(pile) == 0:  # On veut dépiler alors qu'il n'y a rien...
                return False
            else:  # On vérifie la compatibilité
                y = pile.pop()
                if not([y, x] in liste_parentheses):
                    return False
    return len(pile) == 0  # on vérifie qu'il n'y a plus rien dans la pile


liste_test = ['(', ')', ')echec!(', 'blabla', '(ploum)((pidoum)(éàç))']
for chaine in liste_test:
    print('---')
```

```
    print(chaine)
    print('entier  :', Parenthesage_entier(chaine))
    print('pile    :', Parenthesage_pile(chaine))
    print('complexe:', Parenthesage_complexe(chaine))

liste_test2 = ['{([o]u[i]!!)}', 'n{(})non!']

for chaine in liste_test2:
    print('---')
    print(chaine)
    print('complexe:', Parenthesage_complexe(chaine))


"""Exercice 5"""


def tri_bulle(L):
    """
    Trie la liste L par ordre croissant, en place.
    Keyword arguments:
    L -- Liste d'elements comparables via >
    """
    inchange = False   # Permet de detecter un passage sans permutations.
                       # Au départ on veut entrer dans la boucle while
    while (inchange is False):
        inchange = True  # A priori il n'y a pas eu de permutation.
        for i in range(len(L)-1):
            if L[i] > L[i+1]:
                L[i], L[i+1] = L[i+1], L[i]
                inchange = False  # une permutation a eu lieu
```