```
# ================ #
# PSI* / 2016-2017  #
# DS du 1er octobre #
# ================ #

## Exercice N°1

def ListCreate(n):
    L = []
    for i in range(n):
        L += list(range(i+2))
    return(L)

## Exercice N°2

def ListContent(L):
    # Initialisation des variables
    LNUM, LSTRING = [], []
    ni, nf = 0, 0 # nombre d'entiers et flottants
    # Annalyse de la liste L
    for e in L:
        if type(e) == int:
            LNUM.append(e)
            ni += 1
        elif type(e) == float:
            LNUM.append(e)
            nf += 1
        else:
            LSTRING.append(e)
    # Affichage des informations de synthèse
    print('Votre liste contient :')
    print(str(len(LSTRING)) + ' chaîne(s) de caractères')
    print(str(ni) + ' entier(s)')
    print(str(nf) + ' flottant(s)')
    # Renvoi des sous-listes
    return (LNUM,LSTRING)

## Exercice N°3

# importation requise pour la méthode rotation de la classe point
from math import cos, sin

class point:
    def __init__(self,x,y,s):
        self.name = s # de type str
        self.abs  = x # de type float
        self.ord  = y # de type float

    def __repr__(self):
        return(self.name + '(' + str(self.abs) + ',' + str(self.ord) + ')')

    def milieu(self,P):
        return(point(0.5*(self.abs + P.abs),0.5*(self.ord + P.ord),'m' + self.name +
P.name))

    def translation(self,v):
        return(point(self.abs + v.abs,self.ord + v.ord,'t' + self.name))

    def rotation(self,C,alpha):
        return(point(C.abs + (self.abs - C.abs) * cos(alpha) - (self.ord - C.ord) *
sin(alpha),C.ord + (self.abs - C.abs) * sin(alpha) + (self.ord - C.ord) *
cos(alpha),'r' + self.name))

class vecteur:
    def __init__(self,x,y,s):
        self.name = s # de type str
```

```
        self.abs  = x # de type float
        self.ord  = y # de type float

    def __repr__(self):
        return(self.name + '(' + str(self.abs) + ',' + str(self.ord) + ')')

    def norme(self):
        return((self.abs**2 + self.ord**2)**0.5)

    def __neg__(self):
        return(vecteur(-self.abs,-self.ord,'-' + self.name))

    """
    Pour __add__ et __radd__, il convenait d'envisager des calculs tels u+v
    (addition de deux vecteurs, qui donne un vecteur) mais aussi u+A
    (dans __add__) ou A+u (dans __radd__), addition d'un point et d'un vecteur
    qui donne bien sûr un point (translaté).
    """
    def __add__(self,V):
        if type(V) == vecteur:
            return(vecteur(self.abs + V.abs,self.ord + V.ord,'s' + self.name +
V.name))
        else:
            return(point(self.abs + V.abs,self.ord + V.ord,'t' + V.name + self.name))

    def __radd__(self,V):
        return(self + V)
```