```
_corr0 = [4, 2, 1, 4, 3, 5, 3, 3, 2, 1, 1, 3, 3, 5, 4, 4, 5, 1, 3, 3]
_copTM = [4, 1, 5, 4, 3, 3, 1, 4, 5, 3, 5, 1, 5, 5, 5, 1, 3, 3, 3, 3]
_corrTM = [True, False, False, True, True, False, False, False,
False, False, False, False, False, True, False, False, False, False,
True, True]


# Ex 1 : plusieurs versions

def corrige(copTM, corr0):
    corrTM = []
    for i in range(len(copTM)):
        corrTM.append(copTM[i] == corr0[i])
    return corrTM

def corrige(copTM, corr0):
    return [copTM[i] == corr0[i] for i in range(len(copTM)) ]

print("Ex1",corrige(_copTM, _corr0))



# Ex 2

def note(copTM, corr0):
    corrTM = corrige(copTM, corr0)
    n = 0
    for b in corrTM:
        if b :
            n += 1
    return n

print("Ex2", note(_copTM, _corr0))


# Ex 3

_p1 = {('Tom', 'Matt'): [4, 1, 5, 4, 3, 3, 1, 4, 5, 3, 5, 1, 5, 5, 5, 1,
3, 3, 3, 3],
('Lambert', 'Ginne'): [2, 4, 2, 2, 1, 2, 4, 2, 2, 5, 1, 2, 5, 5, 3, 1, 1,
1, 4, 4],
('Carl', 'Roth'): [5, 4, 4, 2, 1, 4, 5, 1, 5, 2, 2, 3, 2, 3, 3, 5, 2, 2,
3, 4],
('Kurt', 'Jett'): [2, 5, 5, 3, 4, 1, 5, 3, 2, 3, 1, 3, 4, 1, 3, 1, 3, 2,
4, 4],
('Ayet', 'Finzerb'): [4, 3, 5, 3, 2, 1, 2, 1, 2, 4, 5, 5, 1, 4, 1, 5, 4,
2, 3, 4]}

def notes_paquet(p1, corr0):
    d = {}
    for key in p1:
        d[key] = note(p1[key], corr0)
    return d

print("Ex3", notes_paquet(_p1, _corr0))
```

Ex 4 : Interdit, les clés ne doivent pas être mutable. (leur valeur ne doivent pas pouvoir changer)

Ex 5 : On pourra utiliser un identifiant unique par candidat, comme un numéro de candidat

Ex 6: ((('Tom','Matt'), 6), (('Lambert', 'Ginne'), 4), (('Kurt', 'Jett'), 4), {('Carl', 'Roth'): 2, ('Ayet', 'Finzerb'): 3})

Ex 7: Il retourne le noms des trois meilleurs élèves, ceux qui ont les meilleurs notes (1er a, 2nd b, 3ème c) et le reste des élèves dans d

Ex 8: d sera toujours un dictionnaire vide, les 3 entrés seront triées par ordre de note décroissante

Ex 9

```
def enigme(notes) :
    a = None
    b = None
    c = None
    d = {}
    for nom in notes :
        tmp = c
        if a == None or notes[nom] > a[1] :
            c = b
            b = a
            a = (nom, notes[nom])
        elif b == None or notes[nom] > b[1] :
            c = b
            b = (nom, notes[nom])
        elif c == None or notes[nom] > c[1] :
            c = (nom, notes[nom])
        else :
            d[nom] = notes[nom]
        if tmp != c and tmp != None :
            d[tmp[0]] = tmp[1]
    return (a, b, c, d)

def classement(d):
    l = []
    e = enigme(d)
    if e[0] is not None : l.append(e[0])
    if e[1] is not None : l.append(e[1])
    if e[2] is not None : l.append(e[2])
    while len(e[3]) != 0:
        e = enigme(e[3])
        if e[0] is not None : l.append(e[0])
        if e[1] is not None : l.append(e[1])
        if e[2] is not None : l.append(e[2])
    return l

print("Ex 9",classement({('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2, ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3}))
```

```
# Ex 10

l = [True, False, False, False, False, False, False]

def renote_express2(copcorr) :
    gauche = 0
    droite = len(copcorr) - 1
    while droite - gauche > 1 :
        milieu = (gauche + droite)//2
        if copcorr[milieu] :
            gauche = milieu + 1
        else :
            droite = milieu - 1
    print(gauche, droite)
    if copcorr[gauche] :
        return gauche + 1
    else :
        return gauche

print("Ex10", renote_express2(l))


# Ex 11

# renote_express = O(n)
# renote_express2 = O(log(n))


# Ex 12

# Mal écrite
```