Exercice

1. L'égalité sert à affecter une valeur à une variable et la double égalité sert à tester si deux variables sont égales.

```
>>>a = 3 # a prend la valeur 3
>>>a
3
>>>a == 4 # renvoie un booléen : True ou False
False
```

2. On a les deux techniques suivantes :

```
>>>import math as m
>>>m.sqrt(5)
2.23606797749979
>>>5 ** ( 1 / 2)
2.23606797749979
```

3. Cela renvoie $True$. En effet $3.2 != 3$ est vrai et $7 \% 3 == 1$ est vrai aussi car le reste de la division euclidienne de 7 par 3 vaut 1.
4. On a par exemple :

```
>>>L = [2, 4, 6]
>>>L[-1]
6
>>>L[len(L) - 1]
6
```

5. (a) Avec une boucle for :

```
L = []
for i in range(1, 1000):
    if i % 2 == 1: # test de l'imparité de i
        L.append(i)
```

(b) Avec une boucle $while$:

```
M = []
i = 1
while i < 1000:
    M.append(i)
    i = i + 2
```

(c) Avec une liste en compréhension :

```
N = [ 2 * i + 1 for i in range(500)]
```

6. Lors du premier passage dans la boucle la commande *return* stoppe l'exécution de la fonction et la valeur 0 est renvoyée.
7. La liste L vaut [0, 1, 2]. Pour chaque valeur de i entre 0 et 2 toutes les valeurs de j entre 0 et 2 sont parcourues, ce qui donne :

```
00
01
02
10
11
12
20
21
22
```

8. Voici les résultats, il faut juste suivre pas à pas les différentes conditions selon les valeurs de n :

```
>>>[test2(k) for k in range(10)]
[0, 0, 2, 3, 4, 3, 3, 6, 7, 8]
```

9. Voici une fonction réalisant ceci. Si la liste est vide, on ne rentre pas dans la boucle et le programme renvoie 1, ce qui est cohérent car le produit vide vaut 1.

```
def mul(L):
    """renvoie le produit des éléments de la liste L"""
    p = 1
    for i in range(len(L)):
        p = p * L[i]
    return(p)
```

10. Cela renvoie une erreur car a est une variable locale qui n'existe qu'au sein de la fonction tandis que l'affichage se fait en dehors de la fonction :

```
NameError: name 'a' is not defined
```

11. On a :

```
>>>L = [4, 8]
>>>L.append(6)
>>>L
[4, 8, 6]
>>>L + [5]
[4, 8, 6, 5]
```

12. Voici une fonction à l'aide d'une boucle *for* :

```
def rev(ch):
    """écrit la chaine de caractères dans l'autre sens"""
    ch2 = "" # la nouvelle chaine de caractère, vide initialement
    for i in range(0, len(ch)):
        ch2 = ch2 + ch[len(ch) - i - 1] # on ajoute les éléments en partant de la fin
    return(ch2)
```

13. Pour cette question, je vous renvoie au chapitre 1.
14. Nous avons vu cette fonction classique en exercice dans le chapitre 1 :

```
def comptage(L):
    """fonction qui renvoie un dictionnaire permettant de connaitre le nombre d'occurrences des éléments
     de la liste L"""
    d = {} # création d'un dictionnaire vide
    for i in L:
        if i in d:
            d[i] += 1 # si l'élément est déjà une clé, on ajoute 1
        else:
            d[i] = 1 # sinon on crée l'entrée correspondante dans le dictionnaire
    return(d)
```

15. On reprend l'algorithme déjà étudié :

```
def tri_bulles (L):
    """ordonne la liste L dans l'ordre croissant en utilisant le tri bulle"""
    n =len(L) # la longueur de la liste
    trier = False # cette variable prend la valeur True lorsque l'on fait un parcours sans échange
    while trier == False:
        trier = True # pour l'instant, on n'a pas fait d'échange
        for i in range(n - 1): # on parcourt la liste
            if L[i] > L[i + 1]:
                L[i], L[i +1] = L[i + 1], L[i] # on échange les paires mal rangées
                trier = False # on indique que la liste n'est pas triée car on a fait un échange
    return(L) # quand on sort de la boucle, la liste est triée
```

D'après le cours, on sait que la complexité du tri bulle est $O(n^2)$ où n est la longueur de la liste.

16. Voici cette fonction classique :

```
def mini(L):
    """renvoie le minimum et un indice de ce minimum"""
    if L == []:
        return(None) # si la liste est vide, on ne renvoie rien
    else:
        indice = 0
        mini = L[0] # initialisation de l'indice et du minimum
        n = len(L)
        for i in range(1, n):
            a = L[i]
            if a < mini: # si l'on trouve un élément plus petit, c'est le nouveau minimum
                indice = i
                mini = a
    return((mini, indice))
```

17. On utilise, par exemple, les fonctions qui existent dans le module *numpy* :

```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(8, 11, 10000)
Y = np.sqrt(X) + X ** 2 + 1
plt.plot(X, Y)
plt.show()
```

18. Voici une fonction qui réalise ceci :

```
import random as rd

def mel(L):
    """melange la liste L"""
    M = [] #la nouvelle liste
    while L != []:
        i = rd.randint(0, len(L) - 1) #choix d'un indice au hasard
        M.append(L[i]) #on ajoute l'élément à M
        L.pop(i) # on le supprime dans L
    return(M)
```

Problème

1. (a) On propose la fonction suivante :

```
def minmax(L):
    """renvoie la taille minimale et la taille maximale"""
    tmax = 0 # initialisation des variables taille max et taille min
    tmin = 3000
    for i in L: # on parcourt les éléments de la liste
        if i > tmax:
            tmax = i
        if i < tmin:
            tmin=i
    return((tmin,tmax))
```

(b) On propose la fonction suivante :

```
def moy(L):
    """renvoie la moyenne des valeurs de la liste L"""
    S = 0
    for i in L:
        S = S + i # on somme les éléments de la liste
    return(S / len(L))
```

(c) On propose la fonction suivante :

```
def sup(L, t):
    """renvoie le nombre d'éléments de la liste L supérieurs à t"""
    nb = 0
    for i in L:
        if i > t:
            nb = nb + 1 # si l'on trouve plus grand que t, on incrémente le compteur
    return(nb)
```

(d) On utilise ici deux boucles imbriquées :

```
def idem(L):
    """teste si deux éléments de L sont identiques"""
    for i in range(len(L)):
        for j in range(i + 1, len(L)): # on commence à i+1 pour ne pas prendre deux fois le même
 élément
            if L[i] == L[j]:
                return(True) # ce return stoppe la fonction
    return(False)
```

(e) La fonction *set* permet de transformer la liste L en ensemble, c'est-à-dire que les doublons sont éliminés. Le test $set(L) == len(L)$ permet par conséquent de savoir si la liste L ne contient pas d'élément en double. Finalement, cette fonction renvoie $True$ si deux personnes ont la même taille, $False$ sinon.

2. (a) On propose la fonction suivante :

```
def pivot(L, i):
    """renvoie le nombre d'éléments inférieurs et supérieurs à celui d'indice i"""
    a = 0
    b = 0
    p = L[i] # pour ne pas le recalculer à chaque fois
    for j in range(len(L)):
        if L[j] < p:
            a = a + 1
        elif L[j] > p:
            b = b + 1
        # quand j = i, a et b ne seront pas incrémentés
    return((a, b))
```

(b) La médiane est l'élément de la liste se trouvant exactement au milieu. La liste étant de longueur impaire, il n'y a pas d'ambiguité.

```
def mediane(L):
    """renvoie la médiane de la liste L qui est de longueur 49"""
    for i in range(49):
        a, b = pivot(L, i)
        if a == 24: # lorsque a = 24 et donc b = 24, on a la médiane
            return(L[i])
```

La fonction *pivot* réalise au maximum $2n$ comparaisons. En effet, chaque passage dans la boucle for donne lieu à une ou deux comparaisons. La fonction médiane fait appel au maximum n fois à la fonction *pivot* et réalise aussi n comparaisons en testant si a vaut 24. Ainsi, il y a au maximum $2n^2 + n$ comparaisons. La complexité est en $O(n^2)$.

(c) Afin de trouver la médiane, on peut aussi envisager de trier la liste, avec un tri bulle par exemple, et prendre l'élément $L[24]$.

3. On commence par trier la liste afin de la partager ensuite comme demandé. Pour la trier, on peut utiliser un tri bulle.

```
def tri_bulles (L):
    """ordonne la liste L dans l'ordre croissant en utilisant le tri bulle"""
    n =len(L)
    trier = False
    while trier == False:
        trier = True
        for i in range(n - 1):
            if L[i] > L[i + 1]:
                L[i], L[i +1] = L[i + 1], L[i]
                trier = False
    return(L)

def partage(L):
    """partage la liste L en trois sous-listes selon l'ordre croissant en utilisant le tri bulle"""
    L = tri_bulles (L)
    L1 = []
    L2 = []
    L3 = []
    for i in range(49): # la liste est triée, il s'agit juste de placer chaque tier dans L1, L2 ou L3
        if i < 16:
            L1.append(L[i])
        elif i > 32:
            L3.append(L[i])
        else:
            L2.append(L[i])
    return(L1, L2, L3)
```

4. (a) De façon classique :

```
def Ltailles (d):
    """renvoie la liste des tailles qui sont les valeurs du dictionnaire d"""
    Lt = []
    for i in d: # on parcourt les clés
        Lt.append(d[i])
    return(Lt)
```

(b) On propose la fonction suivante :

```
def Lnoms(d):
    """renvoie la liste des noms qui sont les clés du dictionnaire"""
    Lnoms = []
    for i in d:
        Lnoms.append(i)
    return(Lnoms)
```

(c) On utilise notamment la fonction *partage* de la question 3.

```
def tiers(d):
    """partage la classe en trois tiers selon la taille """
    Lt = Ltailles(d) # on récupère la liste des tailles
    L1, L2, L3 = partage(Lt) # on partage les tailles en trois tiers
    N1, N2, N3 = [], [], []
    for i in d: # on regarde dans quelle liste mettre le nom selon sa taille
        if d[i] in L1:
            N1.append(i)
        elif d[i] in L2:
            N2.append(i)
        else:
            N3.append(i)
    return((N1, N2, N3))
```

(d) On utilise la fonction *tiers* précédente :

```
def queltiers(d, nom):
    """renvoie le tiers d'appartenance de l'étudiant"""
    N1, N2, N3 = tiers(d)
    if nom in N1:
        return(1)
    elif nom in N2:
        return(2)
    else:
        return(3)
```

(e) Il manque les guillemets autour de $BELLON - WANG$ pour avoir une chaine de caractères comme précisé dans l'énoncé...