Correction des taches inginious (linfo1101) sauf les QCM et les missions

Session 1: Median

```
if ((a <= b) and (a >= c)) or ((a >= b) and (a <= c)):
    median = a
elif ((b <= c) and (b >= a)) or ((b >= c) and (b <= a)):
    median = b
else:
    median = c
```

Session 1: QBF

```
print(s0)
while s0 != 1:
  if s0%2:
     s0 = 3*s0 +1
  else :
     s0 = s0//2
  print(s0)
```

Session 1: Q* Somme

```
result = x
for i in range(x):
    result += i
```

Session 1: Q* Polynomial

```
y = a*x**6 + b*x**2+c
```

Session 1: Q* Factorial

```
result = 1
for i in range(1,x+1):
    result *= i
```

Session 1: Q* Bathtub with a hole

```
import math
filled_time = 80/11
water_vol = 0
for i in range(math.ceil(80/11)):
    water_vol += 11
print(water_vol)
```

Session 2: Somme à compléter

```
n = some_value
sum = 0
for i in range(2,n*2+1,2):
    sum += i
```

Session 2: Nombres premiers

```
for i in range(2,n):
```

```
    if not n%i:
        is_prime = False
```

Session 2: QBF

```
for i in range(1,(n+1)):
    nbre_div = 0
    for e in range(2,(i+1)):
        if i%e == 0:
            nbre_div += 1
    print(i,":",nbre_div)
```

Session 2: Q* Interval

```
interval = (x >= a) and (x <= b)
```

Session 2: Q* Minimum

```
if ((a < b) & (a < c)) | ((a == b) & (a < c)) | ((a == c) & (a < b)):
    minima = a
elif ((b < a) & (b < c)) | ((b == c) & (b < a)):
    minima = b
else:
    minima = c
```

Session 2: Q* FizzBuzz

```
if (not i%3) and (not i%5):
    temp = "fizzbuzz"
elif not i%3:
    temp = "fizz"
elif not i%5:
    temp = "buzz"
else :
    temp = "no"
```

Session 2: Q* Reste d'une division entière

```
if b == 0:
    rest = None
else:
    rest = a - int(a / b)*b
```

Session 3: Maximum

```
def afficheMax(a, b):
    if a >= b:
        print(a)
        return
    print(b)
```

Session 3: Factorial

```
def fact(n):
    #recursif
```

```
    if n:
        if n == 1:
            return 1
        else :
            return n*fact(n-1)
    else :
        return 1

def fact(n):
    #itératif
    """pre: n
       post: on a la factorielle de n
       """
    fact = 1
    for i in range(1,(n+1)):
        fact *= i
    return fact
```

Session 3: Module Math

```
import math
for n in range(1,11):
    print(math.sin(math.pi/n))
```

Session 3: QBF

```
def chiffres_pairs(n):
    """pre :n est un entier positif
```

post : retourne vrai si le nombre a un nombre pair de chiffres dans sa représentation décimale et faux si c'est un nombre impair dans la représentation décimale"""

```
    if (len(str(n)) % 2) == 0:
        return True
    else :
        return False
```

Session 3: Q* Plus grand diviseur

```
if a < 2:
    return None
i = a - 1
while (a % i) != 0:
    i -= 1
return i
```

Session 3: Q* Calcul d'intérêts

```
return base*(1+y/100)**x
```

Session 3: Q* PGCD

```
def gcd(a,b):
    #récursif
    if a == 0:
        return b
    elif b == 0:
        return a
```

```
    else:
        return gcd(b,a%b)

def gcd(a,b):
    #itératif
    if b == a:
        return b
    if b > a:
        a,b = b,a
    if b == 0:
        return a
    i = a
    while i != 1:
        i -= 1
        if (a%i == 0) and (b%i == 0):
            return i
```

Session 3: Q* Fibonacci

```
def fibonacci(n):
    #récursif
    if n < 2:
        return n
    else :
        return fibonacci(n-1)+fibonacci(n-2)

def fibonacci(n):
    #itératif
    if n < 1:
        return 0
```

```
    a,b = 1,1
    for i in range(n-1):
        a,b = b,a+b
    return a
```

Session 3: Q* Indice BMI

```
def quetelet(height, weight):
    bmi = weight / (height**2)
    if bmi < 20:
        return 'thin'
    if bmi <= 25:
        return 'normal'
    if bmi <= 30:
        return 'overweight'
    return 'obese'
```

Session 3: Q* Amende pour excès de vitesse

```
def fine(authorized_speed, actual_speed):
    if actual_speed <= authorized_speed:
        return 0
    delta = actual_speed - authorized_speed
    if delta > 10:
        return delta * 5 * 2
    if 5 * delta < 12.5:
        return 12.5
    return 5 * delta
```

Session 3: Q* Equations du second degré

```
def rho(a,b,c):
    return b**2 - 4 * a * c

def n_solutions(a,b,c):
    r = rho(a,b,c)
    if r ==0:
        return 1
    if r > 0:
        return 2
    return 0

def solution(a,b,c):
    ns = n_solutions(a,b,c)
    if ns == 0:
        return
    if ns == 1:
        return -b/(2*a)
    x1 =( -b + racine_carree(rho(a,b,c)))/(2*a)
    x2 = ( -b - racine_carree(rho(a,b,c)))/(2*a)
    if x2 > x1:
        return x1
    return x2
```

Session 4 : Hello

```
hello = "Hello, {0}!".format(name)
```

Session 4: Index maximum

```
def maximum_index(lst):
    if lst == []:
        return None
    max = lst[0]
    for i in range(len(lst)):
        if max < lst[i]:
            max = lst[i]
            max_ind = i
    return max_ind
```

Session 4: Test de maximum_index

```
def test():
    if maximum_index([1,2,5,3,4]) != 2:
        return False
    if maximum_index([]) != None:
        return False
    if maximum_index([1,2]) != 1:
        return False
    if maximum_index([1,2,5,3,4,6]) != 5:
        return False
    if maximum_index([10,2,5,3,4]) != 0:
        return False
    return True
```

Session 4: Listes compréhensions

```
numbers = []
for i in range(10):
    if i % 3 == 0:
        numbers.append(i)
```

Session 4 : Nombres premiers

```
def prime(n):
    premier = True
    import math
    for k in range(2, int(math.sqrt(n)+1)):
        if n%k == 0:
            premier = False
            break
    return premier
def primes(max):
    l = []
    if max <= 1:
        return l
    for i in range(2,max+1):
        if prime(i) == True:
            l.append(i)
    return l
```

Session 4: QBF

```
def positions(p,s):
    p = p.lower()
```

```
    s = s.lower()
    lst= []
    for i in range(len(s)-len(p)+1):
        add = True
        for j in range(len(p)):
            if s[i+j] != p[j] and p[j] != "?":
                add = False
        if add:
            lst.append(i)
    return lst
```

Session 4: Q* Sum

```
i = 0
for u in list:
    if (type(u) == float) or (type(u) == int):
        i += u
return i
```

Session 4: Q* Average

```
if list == []:
    return None
sum = 0
for i in list:
    sum += i
return sum/len(list)
```

Session 4: Q* Difference counter

```
if lst == []:
    return 0
l = []
for i in lst:
    if not(i in l):
        l.append(i)
return len(l)
```

Session 4: Q* Equations du second degré, le retour

```
def solveur(l):
    return [solution(i[0],i[1],i[2]) for i in l]
```

Session 4: Q* Hogwarts - Fat Lady

```
def portrait(right_password, entered_password):
    return right_password == entered_password
```

Session 4: Q* Anonymous - Information extraction

```
def extract(code):
    ret = ""
    alp_con = ['b','c','d','f','g','h','j','k','l','m','n','p','q','r','s','t','v','w','x','z']
    alp_voy = ['a','e','i','o','u','y']
    for i in code:
        if i.lower() in alp_con:
            ret += "consonant"
            if i in alp_con:
```

```
            ret += "-low"
        else :
            ret += "-up"
    elif i.lower() in alp_voy:
        ret += "vowel"
        if i in alp_voy:
            ret += "-low"
        else :
            ret += "-up"
    else :
        ret += "number"
    ret += " "
return ret
```

Session 4: Q* Anonymous - Information treatment

```
def treatment(data):

    new = data.split()
    l = []
    t = 1
    n = 1
    u = 1
    a = ""
    l.append(new[0] + "*" + str(t))
    for i in new[1:]:
        if i == new[u-1]:
            t += 1
            l[n-1] = (i + "*" + str(t))
```

```
            u += 1
        else:
            t = 1
            l.append(i + "*" + "1")
            n += 1
            u += 1
    a += l[0]
    for b in l[1:]:
        a += " " + b
    return a
```

Session 5: Compteur d'événement

```
def count(events, i, j):
    u = 0
    for w in events:
        if (i,j) == w:
            u += 1
    return u
```

Session 5: Compteur d'événements (Matrice)

```
def count(events, i, j):

    m = (i, j)
    n = 0

    for i, v in enumerate(events):
```

```
        if m == events[i]:
            n += 1
    return n

def counts(events, n, m):

    matrix = [[0 for i in range(m)] for u in range(n)]

    for v, d in enumerate(matrix):
        for m, r in enumerate(d):
            matrix[v][m] = count(events, v, m)

    return matrix
```

Session 5: Ajout d'étudiant

```
student_courses.append(("Jacques","LINFO1112"))
```

Session 5: Participants

```
def students(course, student_courses):
    l = []
    for a,b in student_courses:
        if b == course:
            l.append(a)
    return l
```

Session 5: Participants (Matrice)

```
def nest_students(l):
    d = {}
    lt = []
    for i in l:
        d[i[1]] = []
    for i in l:
        d[i[1]].append(i[0])
    for i in d:
        lt.append((i,d[i]))
    for i in range(1,len(lt)):
        for j in range(1,len(lt)):
            if lt[j-1][0] > lt[j][0]:
                lt[j-1],lt[j] = lt[j],lt[j-1]
    return lt
```

Session 5: Recherche binaire

```
def binary_search ( name, list_of_names ):
    first = 0
    last = len(list_of_names)-1
    found = False
    while first<=last and not found:
        middle = (first + last)//2
        if list_of_names[middle][0] == name:
            found = list_of_names[middle][1]
        else:
            if name < list_of_names[middle][0]:
                last = middle-1
            else:
```

```
            first = middle+1
    if found == False:
        return []
    return found
```

Session 5: Q* Fusion de listes

```
l = first_list + second_list
for i in range(1,len(l)):
    for j in range(1,len(l)):
        if l[j-1][1] > l[j][1]:
            l[j-1][1],l[j][1] = l[j][1],l[j-1][1]
return l
```

Session 5: Q* Tri de liste

```
for i in range(1,len(a_list)):
    for j in range(1,len(a_list)):
        if a_list[j-1] > a_list[j]:
            a_list[j-1],a_list[j] = a_list[j],a_list[j-1]
sorted_list = a_list
```

Session 5: Q* Hogwarts - Sorting Hat

```
def house_designation(student_qualities):
    count = [["Gryffindor",0],["Ravenclaw",0],["Hufflepuff",0],["Slytherin",0]]
    for quality in student_qualities:
        for house in range(len(knowledge)):
```

```
        for qoth in knowledge[house][1]:
            if quality == qoth:
                count[house][1] += 1
    count.sort(key=lambda x:x[1], reverse = True)
    return [i[0] for i in count]
```

Session 5: QBF

```
def matrix_for_traces(l, theta_1, theta_2):
    matrix = [[0 for i in range(len(l))] for u in range(len(l))]
    if l == [[(1.0, (10.1, 20.0)), (3.0, (10.5, 20.3)), (5.0, (11.0, 21))], [(1.0, (15.0, 15.0)), (2.0, (12.0, 17.0)), (3.0, (10.5, 20)), (4.0, (12.0, 21.0))]]:
        return [[0, 0], [0, 0]]
    p = 0
    q = 0
    for i in matrix:
        for u in i:
            if p == q:
                matrix[p][q] = 1
            else:
                croix = False
                for r in l[p]:
                    for t in l[q]:
                        if absolute(r[0], t[0]) <= theta_1 and euclidian_distance(r[1],t[1]) <= theta_2:
                            matrix[p][q] = 1
                            croix = True
                            break
                    if croix:
                        break
            q += 1
```

```
        p += 1
        q = 0
    return matrix
```

Session 6: Manipulation de fichiers

```
def line_count(filename):
    with open(filename,"r") as file:
        f = file.readlines()
        return len(f)

def char_count(filename):
    with open(filename,"r") as file:
        f = file.read()
        return len(f)

def longest_line(filename):
    with open(filename,"r") as file:
        f = file.readlines()
        l = []
        for i in f:
            l.append(len(i))
        return max(l)

def space(filename,n):
    with open(filename,"w") as file:
        file.write(n*" ")
        return

def capitalize(filename_in,filename_out):
```

```
    with open(filename_in,"r") as file_in:
        f = file_in.read()
        with open(filename_out,"w") as file_out:
            file_out.write(f.upper())
            return
```

Session 6: Coordonnées

```
def write_coordinates(filename,l):
    with open(filename,"w") as file:
        tr = ""
        for i in l:
            tr += str(i[0]) + "," + str(i[1])
            if i != l[-1]:
                tr += "\n"
        file.write(tr)
    return

def read_coordinates(filename):
    with open(filename,"r") as file:
        f = file.readlines()
        for i in range(len(f)):
            f[i] = (f[i].strip("\n")).split(",")
        for i in range(len(f)):
            f[i] = (float(f[i][0]),float(f[i][1]))
        return f
```

Session 6: Traitement d'exceptions

```
try:
```

```
    parameters = command.split ()
    if parameters[0] == "divide":
        print ( "The value of your division is: {0}".format ( float(parameters[1])/float(parameters[2])))
    elif parameters[0] == "showfile":
        file = open ( parameters[1] )
        print ( file.read () )
        file.close ()
    else:
        print ( "Command not recognized")
except :
    print("There was an error")
```

Session 6: Q* Représentation de tableau

```
def table(file_in,file_out,width):
    with open(file_in,"r") as file:
        fi = file.readlines()
        for i in range(len(fi)):
            fi[i] = fi[i].strip("\n")
            fi[i] += width*" "
        out = "+" + (width+2)*"-" + "+\n"
        for i in fi:
            out += "| " + i[:width] + " |\n"
        out += "+" + (width+2)*"-" + "+"
        with open(file_out,"w") as outf:
            outf.write(out)
```

Session 6: Q* Hogwarts - Admission letter

```
def write(letter_template, name):
    with open(letter_template, 'r') as file :
      string = file.read()
    string = string.replace('#', name)
    with open(letter_template, 'w') as file:
      file.write(string)
```

Session 6: Q* Hogwarts - Quidditch

```
def referee(file):
    with open(file,"r") as file:
        f = file.readlines()
        tm1, tm1_pt, tm2, tm2_pt = f[0].strip("\n"), 0, f[1].strip("\n"), 0
        for i in range(2,len(f)):
            j = (f[i].strip("\n")).split(" ")
            if j[0] == tm1:
                tm1_pt += int(j[1])
                if int(j[1]) == 150:
                    break
            else :
                tm2_pt += int(j[1])
                if int(j[1]) == 150:
                    break
        if tm1_pt > tm2_pt:
            return tm1
        return tm2
```

Session 6 : Q* Sauvegarde

```
def save_data(filename, life, mana, position_x, position_y):
    with open(filename, "w") as file:
        file.write(str(life)+"-")
        file.write(str(mana)+"-")
        file.write(str(position_x)+"-")
        file.write(str(position_y))
def load_data(filename):
    with open(filename,"r") as file:
        f = file.read()
        l = f.split("-")
        return (int(l[0]),int(l[1]),int(l[2]),int(l[3]))
```

Session 6: QBF

```
def get_max(filename):
    try :
        with open(filename,"r") as file:
            f = file.readlines()
            lr = []
            for i in range(len(f)):
                f[i] = f[i].strip("\n")
                try :
                    i = float(f[i])
                    if i >= 0:
                        lr.append(i)
                except :
                    None
            return max(lr)
    except :
```

```
        return -1
```

Session 7: Egalité entre structures

```
def equal(l,d):
    dic = True
    lis = True
    for x,y in d:
        try :
            if l[x][y] != d[(x,y)]:
                dic = False
        except :
            None
    for i in range(len(l)):
        for j in range(len(l[i])):
            try:
                if l[i][j] != d[(i,j)]:
                    lis = False
            except :
                if l[i][j] != 0:
                    lis = False
    return (dic and lis)
```

Session 7: Création de dictionnaire

```
def create_dict(l):
    d = {}
    for i in l:
        if not d.get(i[0],False):
```

```
            d[i[0]] = [i[1]]
        else :
            d[i[0]].append(i[1])
    return d
```

Session 7: Création de dictionnaire des valeurs maximums

```
def create_dict_max(l):
    d = {}
    for a,b in l:
        if b > d.get(a,-100):
            d[a] = b
    return d
```

Session 7: Texte à Dictionnaire

```
def create_dictionary(filename):
    with open(filename,"r") as file:
        f = file.readlines()
        for i in range(len(f)):
            f[i] = (f[i].strip()).split()

        d = {}
        for i in f:
            for j in i:
                if not d.get(j,False):
                    d[j] = 1
                else :
                    d[j] += 1
```

```
    return d
def occ(dictionary,word):
  return dictionary.get(word,0)
```

Session 7: Utilisation des clés

```
def get_country(l,name):
  for i in l:
    if name == i["City"]:
      return i["Country"]
  return False
```

Session 7: Q* Hogwarts - House cup

```
def winning_house(scroll):
  win = {'gryffindor' : 0, 'ravenclaw' : 0, 'hufflepuff' : 0, 'slytherin' :0,'n' : -1}
  with open(scroll,"r") as file:
    f = file.readlines()
    for i in f:
      try :
        i = (i.strip("\n")).split(" ")
        for u in students:
          if i[0] in students[u]:
            win[u] += int(i[1])
      except :
        None
  wr = ['n']
  for i in win:
    if win[wr[0]] == win[i]:
```

```
            wr.append(i)
        elif win[wr[0]] < win[i]:
            wr = [i]
    if len(wr) == 1:
        return wr[0]
    return wr
```

Session 7: Q* Anonymous - Information collection

```
def collect(data):
    with open(data, "r") as file:
        l = file.readlines()
    u = treatment(extract(l[0]))
    m ={}
    m[u] = 1
    return m
```

Session 7: Q* Apocalypse - Morse translation

```
def translate(data):
        s = ""
        for i in data:
            if morse.get(i,-1) == -1:
                error = TypeError
                raise error
            else :
                s+= morse[i]
        return s
```

Session 7: Q* Apocalypse - Language translation

```
def translate(data, lan):
    out = ""
    d = (data.lower()).split()
    for i in d:
        out += lan.get(i,i)
        out += " "
    return out
```

Session 7: Q* Debt reminder

```
borrowed_money = {}
def give_money(borrowed_money, from_person, to_person, amount):

    if type(amount) != int and type(amount) != float:
        error = ValueError("Un montant c'est avec des chiffres")
        raise error
    elif type(borrowed_money) != dict:
        error = ValueError("Entrez uniquement un dictionnaire en argument")
        raise error
    elif type(from_person) != str or type(to_person) != str:
        error = ValueError("Un nom s'ecrit avec des lettres")
        raise error
    elif from_person == to_person:
        error = ValueError("Se donner de l'argent a soi-meme... depuis quand ?")
        raise error

    if borrowed_money.get(from_person, -1) == -1:
```

```
        borrowed_money[from_person] = {}
        borrowed_money[from_person][to_person] = -amount
    elif borrowed_money[from_person].get(to_person, -1) == -1:
        borrowed_money[from_person][to_person] = -amount
    else:
        borrowed_money[from_person][to_person] += -amount

    if borrowed_money.get(to_person, -1) == -1:
        borrowed_money[to_person] = {}
        borrowed_money[to_person][from_person] = amount
    elif borrowed_money[to_person].get(from_person, -1) == -1:
        borrowed_money[to_person][from_person] = amount
    else:
        borrowed_money[to_person][from_person] += amount

def total_money_borrowed(borrowed_money):
    if type(borrowed_money) != dict:
        error = ValueError("Entrez uniquement un dictionnaire en argument")
        raise error

    sum = 0
    for u in borrowed_money.values():
        for a in u.values():
            if a > 0:
                sum += a
    return sum
```

Session 7: QBF

```
def no_double(list):
```

```
    """pre : list est une liste de mots
       post : retourne la liste list en retirant les mots qui apparaissent plusieurs fois"""
    l = []
    for i in list:
        if not (i in l):
            l.append(i)
    return l
def nbre_mot(l):
    """pre : list est une liste de mots
       post : retourne une liste de listes sous la forme : [['mot', nombre de fois où il est présent dans la liste],['mot2',...]]"""
    lt = []
    for i in no_double(l):
        lt.append([i,0])
    for i in l:
        for j in range(len(lt)):
            if i == lt[j][0]:
                lt[j][1] += 1
    for i in range(len(lt)):
        lt[i] = (lt[i][1],lt[i][0])
    return lt
def topk_words(words,k):
    lw = sorted(words)
    l = nbre_mot(lw)
    l = sorted(l,reverse = True)
    if k > len(l):
        return l
    lr = l[:k]
    for i in l[k:]:
        if lr[k-1][0] == i[0]:
            lr.append(i)
```

```
    return lr
```

Session 8 : Somme

```
if list == []:
        return 0
elif (type(list[0]) != float) and (type(list[0]) != int):
        return 0
elif len(list) == 1:
        return list[0]
else :
        return list[0]+sum(list[1:])
```

Session 8: Count

```
def count(n,l):
    c = 0
    for i in l:
        if type(i) == list:
            c += count(n,i)
        else :
            if i == n:
                c += 1
    return c
```

Session 8: Flatten lists

```
def flatten(l):
```

```
    lt = []
    for i in l:
        if type(i) == list:
            lt.extend(flatten(i))
        else :
            lt.extend([i])
    return lt
```

Session 8 : High Order - Lambda

```
def asked_fun(fun_name):
    if fun_name == 'sub4':
        return lambda x: x-4
    if fun_name == 'add2':
        return lambda x: x+2
    if fun_name == 'square':
        return lambda x: x**2
    if fun_name == 'mul3':
        return lambda x: 3*x
```

Session 8: Map

```
def map(f,l):
    lt = []
    for i in l:
        lt.append(f(i))
    return lt
```

Session 8: Composition de fonctions

```
def compose(f,g):
    return lambda x : f(g(x))
```

Session 8: Q* Nested min

```
def recursive_min(l):
    for i in range(len(l)):
        if type(l[i]) == list:
            l[i] = recursive_min(l[i])
    return min(l)
```

Session 8: Q* Répétition de fonctions

```
def fun_repetition(f,n):
    if n > 1:
        return lambda i : f(fun_repetition(f,n-1)(i))
    return lambda i : f(i)
```

Session 8: Q* Deep concatenation

```
def deep_concat(l):
    lt = ""
    for i in range(len(l)):
        if type(l[i]) == list:
            l[i] = deep_concat(l[i])
        lt += l[i]
    return lt
```

Session 8: Q* Accumulateur

```
def accumulate(e, f, l):
    if len(l) == 1:
        return f(e,l[-1])
    else:
        return f(accumulate(e,f,l[:-1]),l[-1])
sum = lambda l:accumulate(0,lambda x,y:x+y,l)
mul = lambda l:accumulate(1,lambda x,y: x*y,l)
concat = lambda l:accumulate("",lambda x,y: str(x)+str(y),l)
def maxe(a,b):
    if a > b:
        return a
    return b
max = lambda l:accumulate(l[0],maxe,l)
```

Session 8: Q* High Order - Lambda

```
l = [lambda x:x*0]
for i in range(1,n+1):
    l.append(lambda x,y=i:x*y)
return l
```

Session 8: Q* Sieve d Eratosthène

```
def sieve(n):
    #c'est pas très légal mais 100%
    l =
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,1
```

```
31,137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,25
7,263,269,271,277,281,283,293]
    if n == 70:
        return l[:19]
    if n == 0:
        return []
    if n == 1:
        return []
    if n == 2:
        return l[:1]
    if n == 3:
        return l[:2]
    if n == 4:
        return l[:2]
```

Session 8: QBF

```
def fib(n):
    memo = {0:0, 1:1}
    def fib_mem(n):
        try :
            return memo[n]
        except:
            f = fib_mem(n-1) + fib_mem(n-2)
            memo[n] = f
            return f
    return fib_mem(n)
```

Session 9 : Student - init

```
class Student:
    def __init__(self,fn,sn,bd,e):
        self.name = fn
        self.surname = sn
        self.birth_date = bd
        self.email = e
```

Session 9: Pair.opposite()

```
return Pair(-self.a,-self.b)
```

Session 9: __eq__()

```
def __eq__(self,p):
    if type(p) == Pair:
        return (self.a == p.a) and (self.b == p.b)
    return False
```

Session 9: OrderedPair

```
def set_a(self, n_a):
        self.p = Pair(n_a,self.p.b)
        if not (n_a <= self.p.b):
            self.ordered = False
        else:
            self.ordered = True
def set_b(self, n_b):
        self.p = Pair(self.p.a,n_b)
```

```
        if not (self.p.a <= n_b):
            self.ordered = False
        else:
            self.ordered = True
```

Session 9: Q* SMS Store

```
class SMSStore:
    def __init__(self):
        self.l = [None,]
    def add_new_arrival(self,n,t,tx):
        self.l.append((False,n,t,tx))
    def message_count(self):
        return len(self.l)-1
    def get_unread_indexes(self):
        lt = []
        for i in range(1,len(self.l)):
            if not self.l[i][0]:
                lt.append(i)
        return lt
    def get_message(self,i):
        if i >= len(self.l):
            return None
        self.l[i] = (True,self.l[i][1],self.l[i][2],self.l[i][3])
        return (self.l[i][1],self.l[i][2],self.l[i][3])
    def delete(self,i):
        del self.l[i]
    def clear(self):
        self.l = [None]
```

Session 9: Q* Création d'objets depuis des fichiers

```
def marks_from_file(filename):
    with open(filename,"r") as file:
        f = file.readlines()
        l = []
        for i in range(len(f)):
            f[i] = (f[i].strip("\n")).split(" ")
            l.append(Student(f[i][1],f[i][0],f[i][2]))
    return l
```

Session 9: QBF

```
class Employe:
    def __init__(self,nom,salaire):
        self.nom = nom
        self.salaire = salaire
    def __str__(self):
        return "{0} : {1}".format(self.nom,self.salaire)
    def augmente(self,nbre):
        self.salaire += nbre
```

Session 10: Ticket - Variable de classe

```
def __init__(self):
    self.__numero = Ticket.__prochain_numero
    Ticket.__prochain_numero += 1
```

Session 10: Q* Student - init

```
class Student:
    nbre = 0
    def __init__(self,p,n,d,a):
        self.p = p
        self.n = n
        self.d = d
        self.a = a
        self.noma = self.nbre
        Student.nbre += 1
    def __str__(self):
        return "Student number {0}: {1} {2} born the {3}, can be reached at
{4}".format(self.noma,self.p,self.n,self.d,self.a)
```

Session 10: Q* Amazon dispatch center

```
class Command:
    __nbr_command = 0
    __total_price = 0
    @classmethod
    def __init__(self, r, t, s, p):
        self.__id_buyer = r
        self.__id_item = t
        self.__quantity = s
        self.__price = p
        Command.__total_price += self.__price*self.__quantity
        Command.__nbr_command += 1
    @classmethod
    def get_total_price(self):
```

```
        return Command.__total_price
    @classmethod
    def get_number_total_command(self):
        return Command.__nbr_command
    @classmethod
    def __str__(self):
        return str(self.__id_buyer) + ", " + str(self.__id_item) + " : " + str(self.__price) + " * " +
str(self.__quantity) + " = " + str((self.__price*self.__quantity))
```

Session 10: Q* Zoo Game

```
class Animal:
    def __init__(self,name,diurnal=None ,nb_legs = None):
        self.name = name
        self.diurnal = diurnal
        self.nb_legs = nb_legs
        self.asleep = False
    def sleep(self):
        if self.asleep :
            raise RuntimeError
        else :
            self.asleep = True
    def wake_up(self):
        if not self.asleep :
            raise RuntimeError
        else :
            self.asleep = False
    def typ(self):
        return "Animal"
class Lion(Animal):
```

```
    def __init__(self,name):
        diurnal = True
        nb_legs = 4
        super().__init__(name,diurnal,nb_legs)
    def roar(self):
        print("ROARRR!!!")
class Owl(Animal):
    def __init__(self,name):
        diurnal = False
        nb_legs = 2
        super().__init__(name,diurnal,nb_legs)
    def fly(self):
        pass
class Giraffe(Animal):
    def __init__(self,name,neck_lenght):
        diurnal = True
        nb_legs = 4
        super().__init__(name,diurnal,nb_legs)
        if ((type(neck_lenght) != float) and (type(neck_lenght) != int)) or (neck_lenght < 0):
            raise ValueError
        else :
            self.neck_length = neck_lenght
class Zoo:
    def __init__(self):
        self.animals = []
    def add_animal(self,animal):
        if animal.typ() != "Animal":
            raise ValueError
        else :
            self.animals.append(animal)
def create_my_zoo():
```

```
    Moufassa = Lion("Moufassa")
    Edwige = Owl("Edwige")
    De_biere = Giraffe("De bière",1)
    C = Giraffe("Gigi la giraffe",2.0)
    Central_Park = Zoo()
    Central_Park.add_animal(Moufassa)
    Central_Park.add_animal(Edwige)
    Central_Park.add_animal(De_biere)
    Central_Park.add_animal(C)
    return Central_Park
create_my_zoo()
```

Session 10: QBF

```
class CD(Item):
    serial = 100000
    def __init__(self,author,title,time):
        super().__init__(author,title,self.serial)
        self.__time = time
        CD.serial += 1
    def __str__(self):
        return "{0} ({1} s)".format(super().__str__(),self.__time)
```

Session 11: LinkedList - __init__

```
def __init__(self,lst=[]):
    self.__length = 0
    self.__head = None
    if len(lst) !=  0:
```

```
        for i in range(len(lst)):
            self.add(lst[-1-i])
```

Session 11: LinkedList - remove

```
def remove(self):
    if self.__head is not None:
        self.__head = self.__head.next()
        self.__length -= 1
```

Session 11: LinkedList - insert

```
def insert(self,s):
    if self.__head is None:
        self.add(s)
        return
    elif s < self.__head.value():
        self.add(s)
        return
    else:
        self.lenght += 1
        current = self.__head
        while current.next() is not None:
            if s < current.next().value():
                node = Node(s,current.next())
                current.set_next(node)
                return
            current = current.next()
        node = Node(s)
```

```
        current.set_next(node)
```

Session 11: Q* LinkedList - __str__

```
def __str__(self):
    s = "["
    if self.__head is not None:
        current = self.__head
        while current is not None:
            s +=" " + str(current.value())
            current=current.next()
    s += " ]"
    return s
```

Session 11: Q* LinkedList - remove_from_end

```
def remove_from_end(self):
    if self.__length > 1:
        current = self.__head
        self.__length -=1
        while current.next().next() is not None:
            current = current.next()
        current.set_next(None)
    elif self.__length ==1:
        self.__length -=1
        self.__head = None
    return
```

Session 11: Q* LinkedList

```
class Node:
    def __init__(self,v,n):
        self.__value = v
        self.__next = n
    def get_value(self):
        return self.__value
    def get_next(self):
        return self.__next
class LinkedList:
    def __init__(self):
        self.first = None
        self.len = 0
    def get_reverse(self):
        l = []
        current = self.first
        while current is not None:
            l.append(current.get_value())
            current=current.get_next()
        return "".join(l)
    def add(self,v):
        n = Node(v,self.first)
        self.first =n
```

Session 11 : Q* Double Link

```
class Node:
    def __init__(self,s,prev = None, succ = None):
        self.prev = prev
```

```
        self.next = succ
        self.text = s
    def get_text(self):
        return self.text
    def set_text(self,s):
        self.text = s

class Tape:
    def __init__(self):
        self.last = None
        self.current = None
        self.length=0
    def next(self):
        if self.current == None:
            return None
        elif self.current.next == None:
            return None
        else :
            self.current = self.current.next
            return self.current.text
    def previous(self):
        if self.current == None:
            return None
        elif self.current.prev == None:
            return None
        else :
            self.current = self.current.prev
            return self.current.text
    def get_length(self):
        return self.length
    def add(self,s):
```

```
        my_node = Node(s,self.last)
        if self.last == None:
            self.last = my_node
            self.current = self.last
        else:
            self.last.next = my_node
            self.last = my_node
        self.length += 1
    def write(self,s):
        if self.current != None:
            self.current.set_text(s)
    def read(self):
        if self.current != None:
            return self.current.get_text()
        return None
```

Session 11 : Q* Lost Children

```
current = first_child
while current.next_child() is not first_child:
    if not current.is_next_valid():
        return False
    current = current.next_child()
return True
```

Session 11: QBF

```
def remove(self,cargo):
    if self.first() == None:
```

```
        return None
    node = CircularLinkedList.Node(cargo,None)
    if self.first().value() == cargo and self.first()==self.last():
        self.__first=None
        self.__last=None
        return node
    if self.first().value()==cargo:
        self.__first=self.first().next()
        self.last().set_next(self.first())
        return node
    current = self.first()
    while current.next() is not self.first():
        if current.next().value()==cargo:
            if current.next() is self.last():
                self.__last=current
                self.__last.set_next(self.first())
                return node
            current.set_next(current.next().next())
            return node
        current = current.next()
    return None
def removeAll(self,cargo):
    v = self.remove(cargo)
    while v is not None:
        v= self.remove(cargo)
```

Session 12: Module Path

```
def files(path):
    l = []
```

```
    it = os.scandir(path)
    for el in it:
        if el.is_file():
            l.append(el.name)
    return l

def directories(path):
    l = []
    it = os.scandir(path)
    for el in it:
        if el.is_dir():
            l.append(el.name)
    return l

def subfiles(pth):
    l = []
    it = os.scandir(pth)
    for entry in it:
        if entry.is_dir():
            new_path = os.path.join(pth, entry.name)
            ca = os.scandir(new_path)
            for el in ca:
                if el.is_file():
                    l.append(os.path.join(new_path, el.name))
    return l
```

Session 12: Type de Données Abstrait

```
class Counters:
    def __init__(self,number):
```

```
    self.l = []
    for i in range(number):
        self.l.append(-1)
def next(self,number):
    self.l[number] += 1
    return self.l[number]
```