```
import matplotlib.pyplot as pl
import numpy as np
pl.close()

## Q1. Methode Euler Explicite (3 pts)
g=9.81
m=80
# Version sans numpy

def euler(t0,tf,z0,C,h):
    t = t0
    u = 0
    T=[t0]
    L=[0]
    while t<=tf:
        u = C*h/m*u**2-g*h+u
        t = t + h
        T.append(t)
        L.append(u)
    return T,L

# Version avec numpy

def euler(t0,tf,z0,C,h):
    n=int((tf-t0)/h)
    T = np.arange(t0,tf+h,h)
    L = np.zeros(n+1)
    for k in range(1,n+1):
        L[k] = C*h/m*L[k-1]**2-g*h+L[k-1]
    return T,L

## Q2. Méthode de Simpson (1 pt)

def altitude(t0,tf,z0,C,h):
    T,V=euler(t0,tf,z0,C,h)
    N=len(T)-1
    somme=0
    for k in range(0,N-1):
        somme+=V[k]+4*V[k+1]+V[k+2]
    return h/6*somme+z0

## Q3.a) Altitude finale de FB suivant C  (1 pt)

def altitudeFB(C):
    return altitude(0,4*60+19,39000,C,1)

## Q3.b) Tracé  (2 pts)

pl.figure()
liste_C=np.arange(0.01,.5,0.001)
liste_altitude=[altitudeFB(C) for C in liste_C]
pl.plot(liste_C,liste_altitude)
pl.grid()
pl.title("Evolution de l'altitude finale en fonction de C")
pl.xlabel("C")
pl.ylabel("Altitude")
pl.show()


## Q3.c) Dichotomie  (3 pts)
```

```python
def coefficient():
    # On résout numériquement, par dichotomie, l'équation altitudeFB(C)=2500. On choisit une pre
    a,b=0.01,0.5
    m=(a+b)/2
    while b-a > 1e-5 :
        if altitudeFB(m)==2500:
            return m
        elif altitudeFB(m)<2500:
            a=m
        else:
            b=m
        m=(a+b)/2
    return m

# Test (non demandé):
print(coefficient())

## Q4. Fonction de formalisation  (2 pts)

def donnees_ligne(chaine):
    c=""
    for x in chaine:
        if x=='\t':
            c=c+','
        elif x!='\n':
            c=c+x
    return c

## Q5  (3 pts)

f= open('releve.txt','r')

f.readline()

Ltemps=[]
Lvitesse=[]
Laltitude=[]

for ligne in f:
    format=donnees_ligne(ligne)
    Ltemps.append(int(format[0]))
    Lvitesse.append(float(format[1]))
    Laltitude.append(float(format[2]))

##
# 4 lignes ici superflues car on voit que le pas de temps est 1 et on
"""tR0=min(Ltemps)    # temps minimal du relevé
tRf=max(Ltemps)    # temps maximal du relevé
taille=len(Ltemps) # nombre de relevés
hR=(tRf-tR0)/taille # pas du relevé des temps
"""

Z = [altitude(0,4*60+19,39000,C,1) for t in Ltemps]

#Ligne de code attendue  (1 pt) :

print(max([abs(Z[i]-Laltitude[i]) for i in range(len(Z))]))


### Exercice 2
```

```
"""
Created on Thu Nov 29 18:04:38 2018

@author: chabe
"""
# Q1 (5 pts)

from fichier_pile import *

def calc_npi(exp):
    p = pile_vide()
    for e in exp:
        if e == '+' or e == '*':
            x = depiler(p)
            y = dépiler(p)
            if e =='+':
                z=x+y
            else:
                z=x*y
        else:
            empiler(p,e)
    return depiler(p)
print (calc_npi([3, 4, '+', 5, '*']))
    #résultat: 35

# Q2 :  2.5 pts

# Q3 : 0.5 pt


### Exercice 3 (4 pts)

"""
Created on Fri Nov 30 14:26:54 2018

@author: chabe
"""
from fichier_pile import *
def gestion_pile(pile,appel):
    i=0
    k=0
    if appel == 0: #cas appel etage 0 prioritaire
        empile(pile,appel) #LIFO car priorité
    elif appel!=0 and (appel- position)*sens>0: #cas appel en cours de mouvement
        empile(pile,appel) #appel reste au sommet pour s'arrêter à l'étage en cours
    else: #Cas classique d'appel non prioritaire
        pile_temp=[]
        for i in range(len(pile)):
            empile(pile_temp,depile(pile)) # à ce stade, pile_temp contient les éléments de pile
        empile(pile,appel) # à ce stade, pile contient appel.
        for k in range(len(pile_temp)):
            empile(pile, depile(pile_temp))# reconstruction de la pile initiale, mais avec en p
```