```python
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 14 17:39:43 2021
Curso: Laboratorio de Fisica 3
Proyecto: Generador de funciones
@author: Kenny Wu Wen C08592
"""
#Importar librerias
import matplotlib.pyplot as plt
import numpy as np
#Objeto generador de funciones
class generador_funciones():
    def __init__(self,Amplitud,frecuencia,tiempo,
                 tipo,cant_muestras,nombre='salida.csv'):
        self.name = nombre+".csv";
        self.A = Amplitud
        self.f = frecuencia
        self.T = 1/self.f
        self.tipo = tipo
        self.t_end = tiempo
        self.muestras = cant_muestras
        #Rango de tiempo a simular
        self.Trange = np.linspace(0, self.t_end, self.muestras)
        self.menu()

    def menu(self):
        if self.tipo == 'seno':
            self.sen()
        elif self.tipo == 'triangular':
            self.triag()
        elif self.tipo == 'coseno':
            self.cos()
        elif self.tipo == 'cuadrado':
            self.cuad()
        else:
            print('No existe funcion')

    def triag(self):
        #Valor inicial de cada periodo
        s=0
        #Valores Xs
        X = self.Trange
        y = []
        #Pendiente de las rectas
        m = 4*self.A/self.T
        #Dependiendo de x, graficar una de las tres rectas
        for x in X:
            if x >= self.T+s:
                s=x
            if x<self.T/4+s and x>=s:
                y.append( m*(x-s))
            elif x>=self.T/4+s and x<3*self.T/4+s:
                y.append(m*((self.T)/2+s - x))
            elif x>=3*self.T/4+s and x<self.T+s:
                y.append(m*(x-(self.T+s)))

        plt.plot(X,y)
        #Salvar la salida en un archivo llamado salida.csv
        np.savetxt(self.name,np.stack([self.Trange.T,np.array(y).T]).T,delimiter=';')
```

```
def sen(self):
    #Frecuencia angular
    omega = 2*np.pi*self.f
    y = self.A*np.sin(omega*self.Trange)
    plt.plot(self.Trange,y)
    np.savetxt(self.name,np.stack([self.Trange.T,y.T]).T,delimiter=';')

def cos(self):
    #Frecuencia angular
    omega = 2*np.pi*self.f
    y = self.A*np.cos(omega*self.Trange)
    plt.plot(self.Trange,y)
    np.savetxt(self.name,np.stack([self.Trange.T,y.T]).T,delimiter=';')

def cuad(self):
    #Valor inicial de cada periodo
    s = 0
    #Valores de salida
    y = []
        for x in self.Trange:
            if x<self.T/2+s:
                y.append(self.A)
            elif x>=self.T/2+s:
                y.append(-self.A)
            if x>= self.T+s:
                s=x

        plt.plot(self.Trange,y)
        np.savetxt(self.name,np.stack([self.Trange.T,np.array(y).T]).T,delimiter=';')

#Parametros
##Amplitudes
Amplitud_T = 2
Amplitud_C = 3
Amplitud_s = 5
Amplitud_cos = 1
A = [Amplitud_T, Amplitud_C, Amplitud_s, Amplitud_cos]
##Frecuencias
frecuencia_T = 0.5
frecuencia_C = 0.25
frecuencia_s = 0.4
frecuencia_cos = 1
f = [frecuencia_T, frecuencia_C, frecuencia_s, frecuencia_cos]
#Tiempo de simulacion
t = 5
#Numero de muestras a generar
m = 500
#Tipo de funciones
tipos = ["triangular","cuadrado","seno","coseno"]
#Parametros para la grafica
plt.xlabel("Tiempo(s)")
plt.ylabel("Amplitud")
plt.grid(1)
plt.title("Funciones generadas en función del tiempo")
#########
#Generador de funciones:
for (i,tipo) in enumerate(tipos):
    generador_funciones(A[i],f[i],t,tipo,m,tipo)
    #Leyenda para las curvas
    plt.legend(tipos)
```