```
"""
Author   : Abraham Flores
File     : HW2.py
Language : Python 3.5
Created  : 2/22/2018
Edited   : 2/26/2018

San Digeo State University
MTH 693b : Computational Partial Differential Equations

Strikwerda 3.4.1 : Boundary Conditions

One Way Wave Equation
    U_t + U_x = 0
    x = [0,1]
    t = [0,6.3]

    U_0(x) = Alpha*e^(-beta*(x-delta)^2)
    Alpha = 5
    beta = 100
    delta = .5
    U(t,-1) = 0
    U(t,3) = U(t,3-h)

    h = 1/40
    lambda = .95

    A. u(t,0) = 0;                      u(t,1) = 2u(t,1-h) - u(t,1-2h)
    B. u(t,0) = 0;                      u(t,1) = 0
    C. u(t,0) = 2u(t,h) - u(t,2h);      u(t,1) = u(t-k,1-h)
    D  u(t,0) = 0;                      u(t,1) = u(t-k,1-h)

"""
import os,glob
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

#Generates intial value function
def intial_foo(x):
    return 5*np.exp(-100*(x-.5)**2)

def plot(x,U,time,title,annotation,fileLoc):
    sns.set(font_scale = 2)
    sns.set_style("darkgrid", {"axes.facecolor": ".9"})
    fig,ax = plt.subplots()
    fig.set_size_inches(16,10)
    plt.plot(x,U,linewidth=3.0,label="t = "+ str(round(time,3)),color="r")
    plt.axis([0, 1, 0, 8])
    plt.xlabel('x (Spatial)')
    plt.ylabel('U(x,t)')
    plt.title(title)
    ax.annotate(annotation[0],xy=(0,0) ,xytext=(.05, 7.5))
    ax.annotate(annotation[1],xy=(0,0) ,xytext=(.05, 6.5))

    plt.legend()
    plt.savefig(fileLoc+".png")
    plt.close()

def makeGif(gifName):
    os.chdir('Figures')
    #Create txt file for gif command
    fileList = glob.glob('*.png') #star grabs everything,
    fileList.sort()
    #writes txt file
    file = open('FileList.txt', 'w')
    for item in fileList:
        file.write("%s\n" % item)
    file.close()

    os.system('convert -delay 10 @FileList.txt ' + gifName + '.gif')
    os.system('del FileList.txt')
    os.system('del *.png')
```

```
    os.chdir('..')

#A.
# u(t,0) = 0
# u(t,1) = 2u(t,1-h) - u(t,1-2h)
def A(h,Lamb):
    #generate array of intial values at t = 0
    X = np.arange(0,1+h,h)
    temp = []
    for dx in X:
        temp.append(intial_foo(dx))

    prev_ = np.array(temp)
    #Need first step from FTCS
    current_ = .5*Lamb*(np.roll(prev_,1)-np.roll(prev_,-1)) + prev_

    #Boundary Conditions
    current_[-1] = 2*current_[-2] - current_[-3]
    current_[0]  = 0

    steps = int(3.14/(Lamb*h)) + 2
    for time in range(steps):
        #plot
        title = "BC: A"
        str_time = '0'*(4-len(str(time)))+str(time)
        outFile = "Figures\LF" + str_time
        BC = ["u(t,0) = 0","u(t,1) = 2u(t,1-h) - u(t,1-2h)"]
        plot(X,prev_,time*Lamb*h,title,BC,outFile)

        #implement Scheme
        next_ = Lamb*(np.roll(current_,1)-np.roll(current_,-1)) + prev_

        #Boundary Conditions
        next_[-1] = 2*next_[-2] - next_[-3]
        next_[0]  = 0

        prev_ = current_
        current_ = next_

    #makeGif
    makeGif("BC_a")
    return 0

#B.
# u(t,0) = 0
# u(t,1) = 0

def B(h,Lamb):
    #generate array of intial values at t = 0
    X = np.arange(0,1+h,h)
    temp = []
    for dx in X:
        temp.append(intial_foo(dx))

    prev_ = np.array(temp)
    #Need first step from FTCS
    current_ = .5*Lamb*(np.roll(prev_,1)-np.roll(prev_,-1)) + prev_

    #Boundary Conditions
    current_[-1] = 0
    current_[0]  = 0

    steps = int(3.14/(Lamb*h)) + 2
    for time in range(steps):
        #plot
        title = "BC: B"
        str_time = '0'*(4-len(str(time)))+str(time)
        outFile = "Figures\LF" + str_time
        BC = ["u(t,0) = 0","u(t,1) = 0"]
        plot(X,prev_,time*Lamb*h,title,BC,outFile)

        #implement Scheme
        next_ = Lamb*(np.roll(current_,1)-np.roll(current_,-1)) + prev_
```

```
        #Boundary Conditions
        next_[-1] = 0
        next_[0]  = 0

        prev_ = current_
        current_ = next_

    #makeGif
    makeGif("BC_b")
    return 0

#C.
# u(t,0) = 2u(t,h) - u(t,2h)
# u(t,1) = u(t-k,1-h)
def C(h,Lamb):
    #generate array of intial values at t = 0
    X = np.arange(0,1+h,h)
    temp = []
    for dx in X:
        temp.append(intial_foo(dx))

    prev_ = np.array(temp)
    #Need first step from FTCS
    current_ = .5*Lamb*(np.roll(prev_,1)-np.roll(prev_,-1)) + prev_

    #Boundary Conditions
    current_[-1] = prev_[-2]
    current_[0]  = 2*current_[1] - current_[2]

    steps = int(3.14/(Lamb*h)) + 2
    for time in range(steps):
        #plot
        title = "BC: C"
        str_time = '0'*(4-len(str(time)))+str(time)
        outFile = "Figures\LF" + str_time
        BC = ["u(t,0) = 2u(t,h) - u(t,2h)","u(t,1) = u(t-k,1-h)"]
        plot(X,prev_,time*Lamb*h,title,BC,outFile)

        #implement Scheme
        next_ = Lamb*(np.roll(current_,1)-np.roll(current_,-1)) + prev_

        #Boundary Conditions
        next_[-1] = current_[-2]
        next_[0]  = 2*next_[1] - next_[2]

        prev_ = current_
        current_ = next_

    #makeGif
    makeGif("BC_c")
    return 0

#D
#u(t,0) = 0
#u(t,1) = u(t-k,1-h)
def D(h,Lamb):
    #generate array of intial values at t = 0
    X = np.arange(0,1+h,h)
    temp = []
    for dx in X:
        temp.append(intial_foo(dx))

    prev_ = np.array(temp)
    #Need first step from FTCS
    current_ = .5*Lamb*(np.roll(prev_,1)-np.roll(prev_,-1)) + prev_

    #Boundary Conditions
    current_[-1] = prev_[-2]
    current_[0]  = 0

    steps = int(3.14/(Lamb*h)) + 2
    for time in range(steps):
```

```
        #plot
        title = "BC: D"
        str_time = '0'*(4-len(str(time)))+str(time)
        outFile = "Figures\LF" + str_time
        BC = ["u(t,0) = 0","u(t,1) = u(t-k,1-h)"]
        plot(X,prev_,time*Lamb*h,title,BC,outFile)

        #implement Scheme
        next_ = Lamb*(np.roll(current_,1)-np.roll(current_,-1)) + prev_

        #Boundary Conditions
        next_[-1] = current_[-2]
        next_[0]  = 0

        prev_ = current_
        current_ = next_

    #makeGif
    makeGif("BC_d")
    return 0

if __name__ == '__main__':
    h = 1.0/40
    L = 0.95
    A(h,L)
    B(h,L)
    D(h,L)

#Change y-limit on plot to -8 : 8 to see error. annotate 6.5->5, 7.5->7
    #C(h,L)


"""
Report:
    The final boundary condition (D) is the only one that does not fall
    victim to the oscillations of leapfrog. D. makes use of leapfrog stability
    for the right boundary and trivialy ignores the left. As the solution is
    propagating to the right. All other conditions either amplfiy the parsitic
    solution (A) or amplfify error due to the oscillations (B(right),C(left))

"""
```