```python
#! python3.7
# -*- coding: utf-8 -*-
from numpy import zeros, linspace, cos, pi
from matplotlib.pyplot import style, figure, axes
from celluloid import Camera

# Набор команд, за счёт которых анимация строится в отдельном окне
from IPython import get_ipython
get_ipython().run_line_magic('matplotlib', 'qt')

# Определение функции, задающей начальное условие
def u_init_0(x) :
    if x >= 4 and x <= 6:
        u_init_0 = cos(pi/2*(x - 5))
    else:
        u_init_0 = 0.
    return u_init_0

# Определение функции, задающей второе начальное условие
def u_init_1(x) :
    u_init_1 = 0.
    return u_init_1

# Определение функции, задающей левое граничное условие
def u_left(t) :
    u_left = 0.
    return u_left

# Определение функции, задающей правое граничное условие
def u_right(t) :
    u_right = 0.
    return u_right

def g(x,t):
    g = 0.
    return g

# Определение входных данных задачи
a = 0.; b = 10.
t_0 = 0.; T = 20.0
c = 1.

# Определение числа интервалов пространственно-временной сетки,
# на которой будет искаться приближённое решение
N = 500; M = 1000

# Определение сетки по пространству
h = (b - a)/N; x = linspace(a,b,N+1)
# Определение сетки по времени
tau = (T - t_0)/M; t = linspace(t_0,T,M+1)

# Выделение памяти под массив сеточных значений решения УЧП
# В строке с номером m этого массива будут храниться сеточные значения решения,
# соответствующие моменту времени t_m
u = zeros((M + 1,N + 1))

# Задание начального условия (на начальном временном слое)
for n in range(N + 1) :
    u[0,n] = u_init_0(x[n])

# Пересчёт сеточных значений в момент времени t_1
# чрез функции u_init_0(x) и u_init_1(x)
for n in range(N + 1):
    u[1,n] = u_init_0(x[n]) + tau*u_init_1(x[n])

# Задание граничных условий
for m in range(1,M + 1) :
    u[m,0] = u_left(t[m])
    u[m,N] = u_right(t[m])
```

```
70
71  # Реализация поиска приближённого решения
72  # с помощью явной разностной схемы
73  for m in range(1,M) :
74      # Вычисление решения на новом временном слое t_{m+1}
75      for n in range(1,N) :
76          u[m + 1,n] = -u[m-1,n] + 2*u[m,n]  + c**2*tau**2/h**2*(u[m,n+1] - 2*u[m,n] +
        u[m,n-1]) + tau**2*g(x[n],t[m])
77
78  # Анимация отрисовки решения
79  style.use('dark_background')
80  fig = figure()
81  camera = Camera(fig)
82  ax = axes(xlim=(a,b), ylim=(-2.,2.))
83  ax.set_xlabel('x'); ax.set_ylabel('u')
84  # Отрисовываем только каждый 2-й кадр
85  for m in range(0,M + 1,2) :
86      # Отрисовка решения в момент времени t_m
87      ax.plot(x,u[m], color='y', ls='-', lw=2)
88      camera.snap()
89  animation = camera.animate(interval=15, repeat=False, blit=True)
90
91  # Листинг программы, реализущей решение линейного уравнения
92  # гиперболического типа, описывающего колебание струны
93  # с помощью разностной схемы
```