```python
import context
import numpy as np
from cr507.utils import plt_set
import matplotlib.pyplot as plt
from collections import namedtuple


class Approximator:

    #############################################
    # initialize condtions
    #############################################
    def __init__(self, valueDict):
        """
        Create the grid and initial conditions
        """
        ##  Defined conditions from dictonary
        self.__dict__.update(valueDict)

        ## Define number of time steps
        nsteps = (self.gridx - 300) / (self.u0 * self.dt / self.dx)
        nsteps = np.arange(0,nsteps)
        self.nsteps = nsteps

        ## Calculate the Courant number
        cr = self.u0 * self.dt / self.dx
        self.cr = cr

        ## Create initial concentration anomaly
        #  distribution in the x-direction
        conc = np.zeros(self.gridx)
        conc[100:151] = np.linspace(0.,self.cmax,51)
        conc[150:201] = np.linspace(self.cmax, 0.,51)
        conc[20:41] = np.linspace(0., -0.5 * self.cmax, 21)
        conc[40:61] = np.linspace(-0.5 * self.cmax, 0., 21)
        self.Pj = np.array(conc)

        ## Define the ideal exact final solution
        cideal = np.zeros(self.gridx)
        cideal[800:851] = np.linspace(0., self.cmax,51)
        cideal[850:901]  = np.linspace(self.cmax, 0., 51)
        cideal[720:741]  = np.linspace(0., -0.5 * self.cmax, 21)
        cideal[740:761]  = np.linspace(-0.5 * self.cmax, 0., 21)
        self.cideal = np.array(cideal)

    #############################################
    # spatial discretization methods
    #############################################
    def centdif(self):
        """
        Centered difference spatial approximation
        """
        # print(self.Pj[50],"centdif start")
        Pj = -self.u0 * ((np.roll(self.Pj,-1) - np.roll(self.Pj,1)) / (2 * self.dx))

        # print(Pj[50],"centdif end")
        return Pj

    def backdif(self):
```

```python
        """
        Backward difference spatial approximation
        """
        # print(self.Pj[50],"backdif start")
        Pj = -self.u0 * ((self.Pj - np.roll(self.Pj,1)) / (self.dx))

        # print(Pj[50],"backdif end")
        return Pj

    ############################################
    # time discretization methods
    ############################################
    def forward(self):
        Pj_OG = self.Pj

        Pjn_1 = []
        for n in range(len(self.nsteps)):
            Pj = self.Pj
            Pn = Pj + self.dt * self.backdif()
            self.Pj = Pn

            Pjn_1.append(Pn)

        Pjn_1 = np.array(Pjn_1)
        print(Pjn_1.shape)
        self.Pj = Pj_OG

        return Pjn_1



    def rk3(self):
        """
        Runge-Kutta 3rd order Centred in Space
        """
        Pj_OG = self.Pj

        Pjn_1 = []

        for n in range(len(self.nsteps)):

            Pj = self.Pj
            # print(Pj[50], "Pj var")
            P_str = Pj + (self.dt/3) * self.centdif()
            # print(P_str[50], 'P_str')

            self.Pj = P_str
            # print(self.Pj[50], 'self Pj should be Pjstr')

            P_str_str  = Pj + (self.dt/2) * self.centdif()
            # print(P_str_str[50], 'P_str_str')

            self.Pj = P_str_str
            # print(self.Pj[50], 'self Pj should be Pj_str_str')

            Pn  = Pj + self.dt * self.centdif()
            Pn = np.array(Pn)
            # print(Pn[50], "Pn pre append")
            Pjn_1.append(Pn)

```

```python
            self.Pj = Pn
            # print(self.Pj[50], "self Pj or Pn")

        Pjn_1 = np.array(Pjn_1)
        self.Pj = Pj_OG

        return Pjn_1

    def plot_functions(self, method):
        if method == 'Initial':
            fig, ax = plt.subplots(1,1, figsize=(12,4))
            fig.suptitle('HW7 Initial concentration', \
                fontsize= plt_set.title_size, fontweight="bold")
            ax.plot(self.xx, self.Pj, color = 'blue', \
                 label = "Initial concentration", zorder = 9)
            ax.set_xlabel('Grid Index (i)', fontsize = plt_set.label)
            ax.set_ylabel('Quantity', fontsize = plt_set.label)
            ax.xaxis.grid(color='gray', linestyle='dashed')
            ax.yaxis.grid(color='gray', linestyle='dashed')
            ax.set_ylim(-10,15)
            ax.legend()
            plt.show()

        elif method == 'Final':
            fig, ax = plt.subplots(1,1, figsize=(12,4))
            fig.suptitle('HW7 Final Ideal', \
                fontsize= plt_set.title_size, fontweight="bold")
            ax.plot(self.xx, self.Pj, color = 'blue', \
                label = "Initial concentration", zorder = 9)
            ax.plot(self.xx,self.cideal, color = 'red', \
                label = "Final Ideal", zorder = 8)
            ax.set_xlabel('Grid Index (i)', fontsize = plt_set.label)
            ax.set_ylabel('Quantity', fontsize = plt_set.label)
            ax.xaxis.grid(color='gray', linestyle='dashed')
            ax.yaxis.grid(color='gray', linestyle='dashed')
            ax.set_ylim(-10,15)
            ax.legend()
            plt.show()

        elif method == 'RK3':
            fig, ax = plt.subplots(1,1, figsize=(12,4))
            fig.suptitle("Runge-Kutta 3rd order Centred in Space  CR: 0.5", \
                fontsize= plt_set.title_size, fontweight="bold")
            ax.plot(self.xx, self.Pj, color = 'blue', \
                 label = "Initial concentration", zorder = 10)
            ax.plot(self.xx,self.cideal, color = 'red', \
                 label = "Final Ideal", zorder = 8)
            Prk3 = self.rk3()
            ax.plot(self.xx,Prk3.T[:,-1], color = 'green', \
            label = "RK3", zorder = 9)
            ax.set_xlabel('Grid Index (i)', fontsize = plt_set.label)
            ax.set_ylabel('Quantity', fontsize = plt_set.label)
            ax.xaxis.grid(color='gray', linestyle='dashed')
            ax.yaxis.grid(color='gray', linestyle='dashed')
            ax.set_ylim(-10,15)
            ax.legend()
            plt.show()

        elif method == 'FTBS':
            fig, ax = plt.subplots(1,1, figsize=(12,4))
```

```python
            fig.suptitle("Forward in time, Backward in space  CR: 0.5", \
                 fontsize= plt_set.title_size, fontweight="bold")
            ax.plot(self.xx, self.Pj, color = 'blue',  \
                label = "Initial concentration", zorder = 10)
            ax.plot(self.xx,self.cideal, color = 'red', \
                label = "Final Ideal", zorder = 8)
            Ftbs = self.forward()
            print(Ftbs.shape)
            ax.plot(self.xx,Ftbs.T[:,-1], color = 'green', \
                label = "FTBS", zorder = 9)
            ax.set_xlabel('Grid Index (i)', fontsize = plt_set.label)
            ax.set_ylabel('Quantity', fontsize = plt_set.label)
            ax.xaxis.grid(color='gray', linestyle='dashed')
            ax.yaxis.grid(color='gray', linestyle='dashed')
            ax.set_ylim(-10,15)
            ax.legend()
            plt.show()

        else:
            pass


        return
```