1 PSet-6.py

```
import numpy as np
import matplotlib.pyplot as plt

fileName = "Final-Project"

# Grid points:
height = 200
width = 500

cylinder_diameter = 50
cylinder_radius = cylinder_diameter / 2
cylinder_center = [(height / 2), 100]

error_limit = 0.01      # 1% maximum change for convergence

U_inf = 2               # m/s uniform inflow
F = 1.9                 # over-relaxation factor
free_lid = 5*(10**(-4)) # free-lid streamfunction constant

# Constants picked for air around room temp
rho = 3000              # kg/m^3
c = 840                 # J/(kg*C)
h = 28                  # W/(m^2*C)  Convective Heat Transfer Coefficient
k = 5.2                 # W/(m*C)    Thermal Conductivity
alpha = k / (rho * c)   # m^2/s      Thermal Diffusivity
nu = 1.48 * 10**(-5)    # m^s/s      kinematic viscosity

h_1 = 9 * nu / U_inf #10 * nu / U_inf
h_2 = 9 * alpha / U_inf #10 * alpha / U_inf
h = min(h_1, h_2)

dt = (h / U_inf) / 2

```

```
# Create array and initialize to T-initial
omega = np.zeros((width, height))       # vorticity
psi = np.zeros((width, height))         # streamfunction


def in_circle(x, y):
    dist = np.sqrt((x - cylinder_center[0])**2 + (y - cylinder_center[1])**2)
    if (dist <= cylinder_radius):
        return True
    return False

def is_fixed(x, y):
    if (y == 0):
        return True
    if (y == (height - 1)):
        return True
    if (y == cylinder_center[1]):
        return True
    if in_circle(x, y):
        return True
    return False

for i in range(width):
    for j in range(height):
        if in_circle(i, j):
            psi[i, j] = 0
for i in range(width):
    psi[i, cylinder_center[1]] = 0
    psi[i, 0] = -free_lid
    psi[i, (height - 1)] = free_lid

# establish initial uniform gradient in psi
for i in range(width):
    for j in range(height):
        if not is_fixed(i, j):
```

```
            psi[i, j] = (2 * j / (height - 1)) * free_lid - free_lid


error_flag = True
while error_flag:
    large_error_term_found = False

    # Gauss-Seidel Iteration
    for i in range(1, width - 1):
        for j in range(1, height - 1):
            psi_old = psi[i, j]

            if not is_fixed(i, j):
                psi[i, j] = psi[i, j] + (F / 4) * (psi[i + 1, j] + psi[i - 1, j] + psi[i
   , j + 1] + psi[i, j - 1] - 4 * psi[i, j])

            if (i == 1):
                psi[0, j] = psi[i + 2, j]
            elif (i == (width - 2)):
                psi[i + 1, j] = psi[i, j]

            if not large_error_term_found and not is_fixed(i, j):
                error_term = abs(psi[i, j] - psi_old) / psi_old
                if (error_term <= error_limit):
                    error_flag = False
                else:
                    error_flag = True
                    large_error_term_found = True

# Print the data in the console (readable format)
# print(np.rot90(psi))

figNum = 1
fig = plt.figure(figNum)
plt.axes().set_aspect('equal')
```

```
data_graphable = np.flipud(np.rot90(psi))


num_streamlines = 31
max_streamline = np.max(data_graphable)
min_streamline = np.min(data_graphable)
contours_before = np.linspace(min_streamline, max_streamline, num=(num_streamlines + 3))
contours = contours_before[(contours_before != 0) & (contours_before != min_streamline)
    & (contours_before != max_streamline)]

plt.contour(data_graphable, levels = contours, colors = 'black', linestyles = 'solid')


plt.xlim(0, width)
plt.ylim(0, height)
plt.xticks(np.arange(0, width + 1, 50))
plt.yticks(np.arange(0, height + 1, 20))
plt.tick_params(top=True, right=True)

plt.style.use('grayscale')
heatmap = plt.pcolor(data_graphable)
plt.clim(np.amin(data_graphable), np.amax(data_graphable))

plt.savefig(fileName + "/images/" + fileName + "-Figure.png")
plt.show()
```