exercise03

May 28, 2025

1 Exercise 01

[1]:
```
import re
import csv

# Regular expression pattern
pattern = re.compile(r"""
    ^(?P<id>\d{5})\s+
    (?P<from>\d{8})\s+
    (?P<to>\d{8})\s+
    (?P<height>\d+)\s+
    (?P<geo_width>-?\d+\.\d+)\s+
    (?P<geo_length>-?\d+\.\d+)\s+
    (?P<name>[^\d]+?)\s{2,}
    (?P<state>\S+)
""", re.VERBOSE)

lines = []
with open('data03/zehn_min_rr_Beschreibung_Stationen.txt', 'r',␣
 ↪encoding='utf8', errors='ignore') as file:
    for line in file:
        lines.append(line.strip())

with open("data03/data_stations.csv", "w", newline="") as csvfile:
    fieldnames = 'id;from;to;height;geo_width;geo_length;name;state'.split(';')
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()

    for line in lines[1:]:
        match = pattern.match(line)
        if match:
            writer.writerow(match.groupdict())
```

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
```

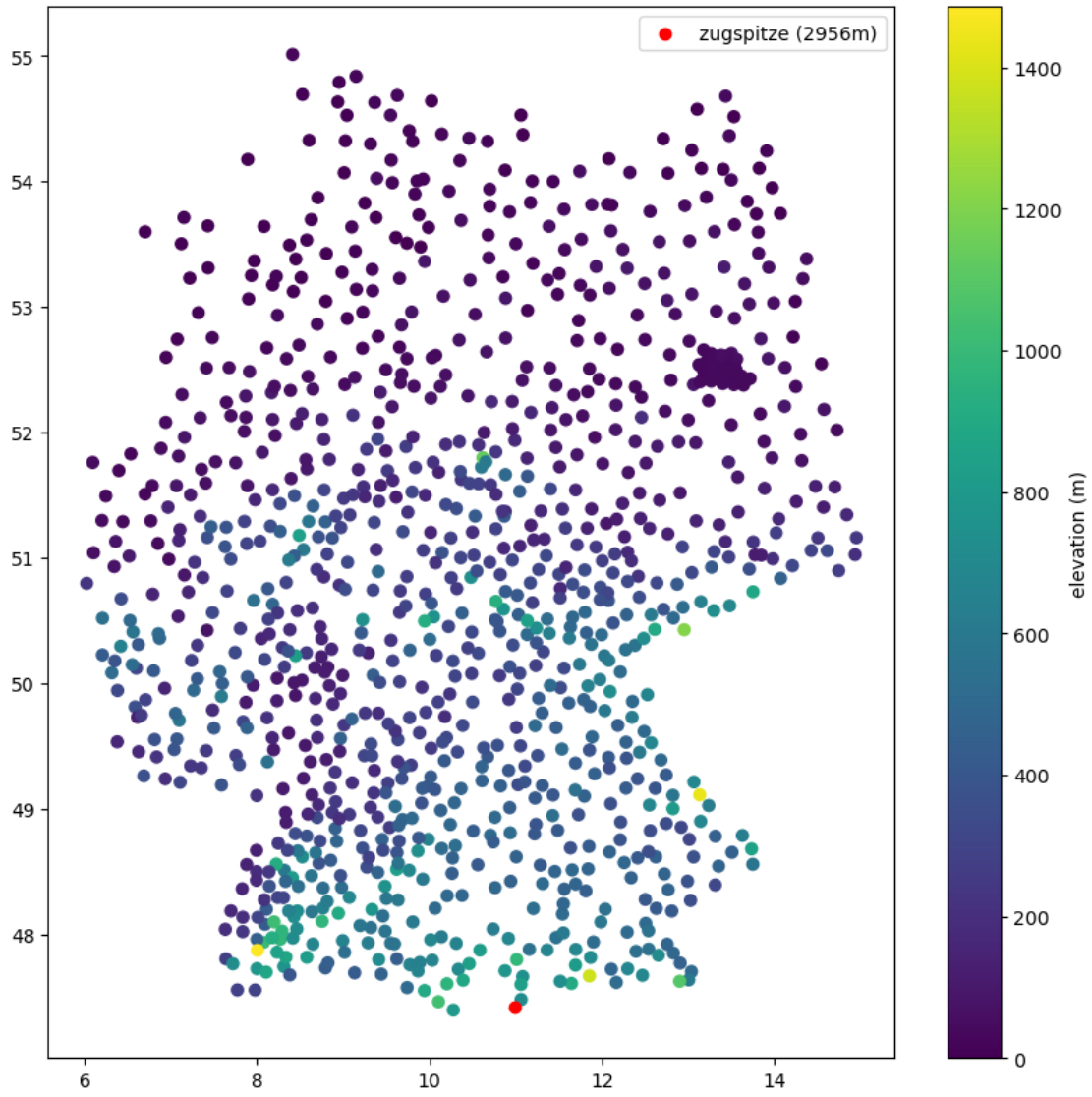
```
from numpy import load
import scipy.interpolate as interp
```

[3]:
```
df = pd.read_csv('data03/data_stations.csv')
df['id'] = df['id'].astype(np.int64)

zugspitze = df[df['height']>2000]
df = df[df['height'] < 2000]

fig, ax = plt.subplots(figsize=(10, 10))
f = ax.scatter(df['geo_length'], df['geo_width'], c=df['height'],␣
 ↪vmin=df['height'].min(), vmax=df['height'].max())
ax.scatter(zugspitze['geo_length'], zugspitze['geo_width'], c='red',␣
 ↪label='zugspitze (2956m)')
ax.legend()

cbar = fig.colorbar(f, label='elevation (m)')
```



[4]:
```
df_rain = pd.read_csv('data03/10min_processed.csv')
df_rain = df_rain.astype({'stationid':np.int32, 'date':str, 'rain':np.float64})
df_rain['date'] = pd.to_datetime(df_rain['date'], format="%Y%m%d%H%M")
df_rain['hour'] = df_rain['date'].map(lambda x: x.hour)
df_rain['minute'] = df_rain['date'].map(lambda x: x.minute)
df_rain['rain'] = df_rain['rain'].replace(-999.0, np.nan)
df_rain

rain_sums = df_rain.groupby('stationid')['rain'].sum().reset_index()
rain_sums.columns = ['id', 'rain_sum']

station_hourly_rain = df_rain.groupby(['stationid', 'hour'])['rain'].sum().
 ↪reset_index()
```

```
station_hourly_pivot = station_hourly_rain.pivot(index='stationid',␣
 ↪columns='hour', values='rain')
station_hourly_pivot.columns = [f'hour_{int(col)}' for col in␣
 ↪station_hourly_pivot.columns]
station_hourly_pivot = station_hourly_pivot.reset_index()
df_station_hours_rain = pd.merge(rain_sums, station_hourly_pivot, left_on='id',␣
 ↪right_on='stationid').drop(columns='stationid')
df_final = pd.merge(df, df_station_hours_rain, left_on='id', right_on='id').
 ↪fillna(0)
df_final
```

[4]:

	id	from	to	height	geo_width	geo_length	\
0	44	20070208	20240422	44	52.9336	8.2370	
1	53	20050831	20240422	60	52.5850	13.5634	
2	73	20070213	20240422	374	48.6183	13.0620	
3	78	20041010	20240422	64	52.4853	7.9125	
4	87	20041019	20240422	158	51.0950	11.0479	
…	…	…	…	…	…	…	
1041	19172	20200820	20240422	48	54.0246	9.3880	
1042	19207	20230330	20240422	16	53.8178	12.0645	
1043	19299	20210322	20240422	463	49.8713	11.7883	
1044	19897	20231231	20240422	37	52.5040	13.4550	
1045	19898	20231231	20240422	39	52.4970	13.2820	

	name	state	rain_sum	hour_0	\
0	Groenkneten	Niedersachsen	5.51	0.00	
1	Ahrensfelde	Brandenburg	2.61	1.16	
2	Aldersbach-Kramersepp	Bayern	5.97	0.00	
3	Alfhausen	Niedersachsen	5.44	0.00	
4	Alperstedt	Thringen	1.68	0.33	
…	…	…	…	…	
1041	Wacken	Schleswig-Holstein	1.69	0.00	
1042	Glzow-Przen	Mecklenburg-Vorpommern	0.00	0.00	
1043	Speichersdorf	Bayern	6.46	0.31	
1044	Berlin-Friedrichshain-Nord	Berlin	1.90	0.90	
1045	Berlin-Halensee	Berlin	1.80	1.00	

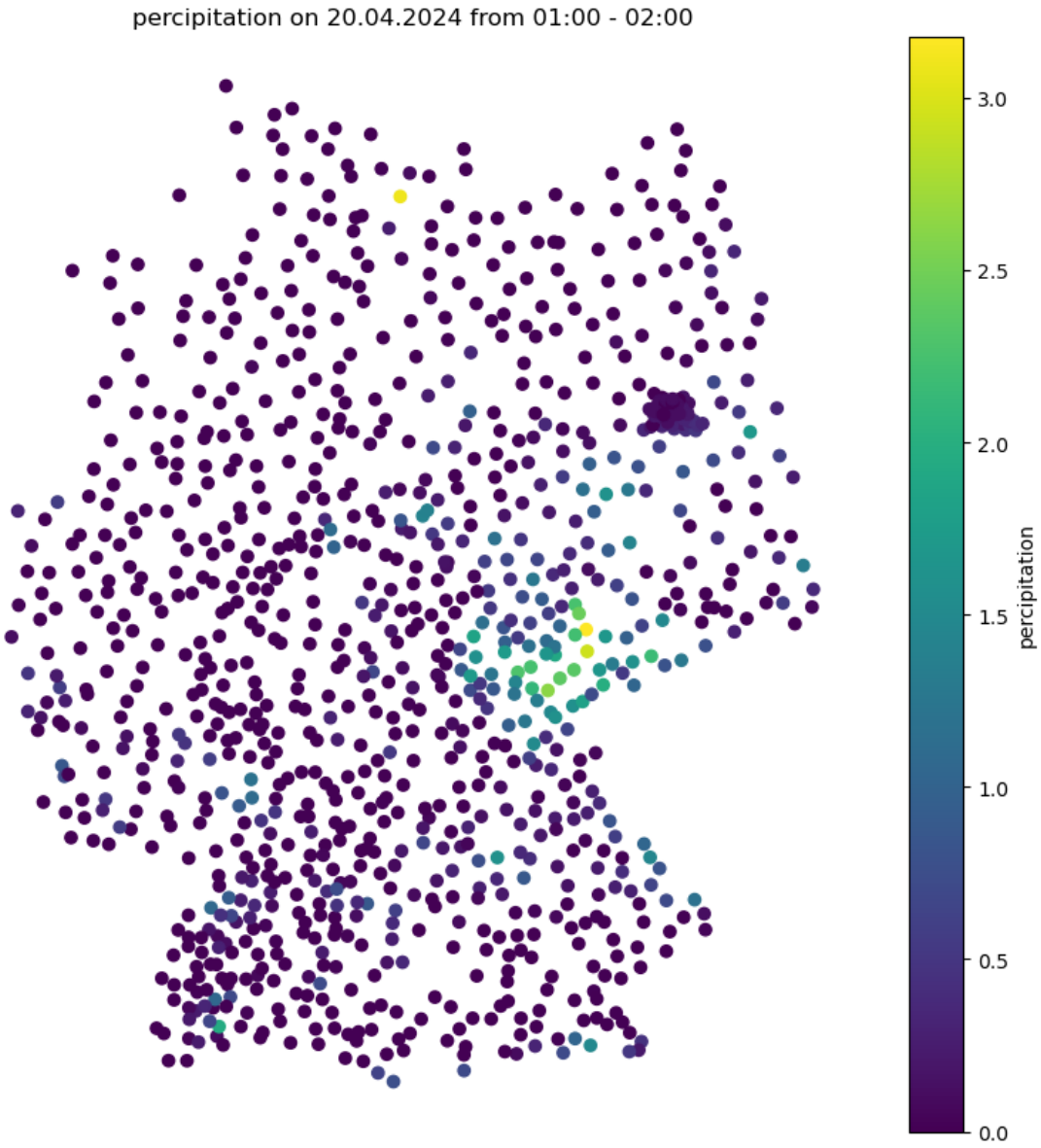
	…	hour_14	hour_15	hour_16	hour_17	hour_18	hour_19	hour_20	\
0	…	1.29	0.40	0.00	0.0	0.00	2.65	0.11	
1	…	0.00	0.00	0.00	0.0	0.00	0.00	0.00	
2	…	1.43	0.00	0.00	0.0	0.23	0.00	0.00	
3	…	0.92	0.16	0.00	0.0	0.01	2.77	0.05	
4	…	0.00	0.00	0.00	0.9	0.01	0.00	0.00	
…	…	…	…	…	…	…	…	…	
1041	…	0.00	0.00	0.00	0.0	0.00	0.23	0.00	
1042	…	0.00	0.00	0.00	0.0	0.00	0.00	0.00	
1043	…	0.27	0.14	0.08	0.0	0.00	0.00	0.00	

```
1044  …     0.00     0.00     0.00      0.0     0.00     0.00     0.00
1045  …     0.00     0.00     0.00      0.0     0.00     0.00     0.00

      hour_21  hour_22  hour_23
0        0.00     0.00     0.00
1        0.00     0.00     0.00
2        0.00     0.00     0.00
3        0.00     0.00     0.00
4        0.00     0.00     0.00
…         …        …        …
1041     0.41     0.59     0.46
1042     0.00     0.00     0.00
1043     0.29     0.14     0.66
1044     0.00     0.00     0.00
1045     0.00     0.00     0.00

[1046 rows x 33 columns]
```

[5]:
```
fig, ax = plt.subplots(figsize=(10, 10))
f = ax.scatter(df_final['geo_length'], df_final['geo_width'],␣
 ↪c=df_final['hour_2'], vmin=df_final['hour_2'].min(), vmax=df_final['hour_2'].
 ↪max())
ax.set_title('percipitation on 20.04.2024 from 01:00 - 02:00')
ax.axis('off')
cbar = fig.colorbar(f, label='percipitation')
```



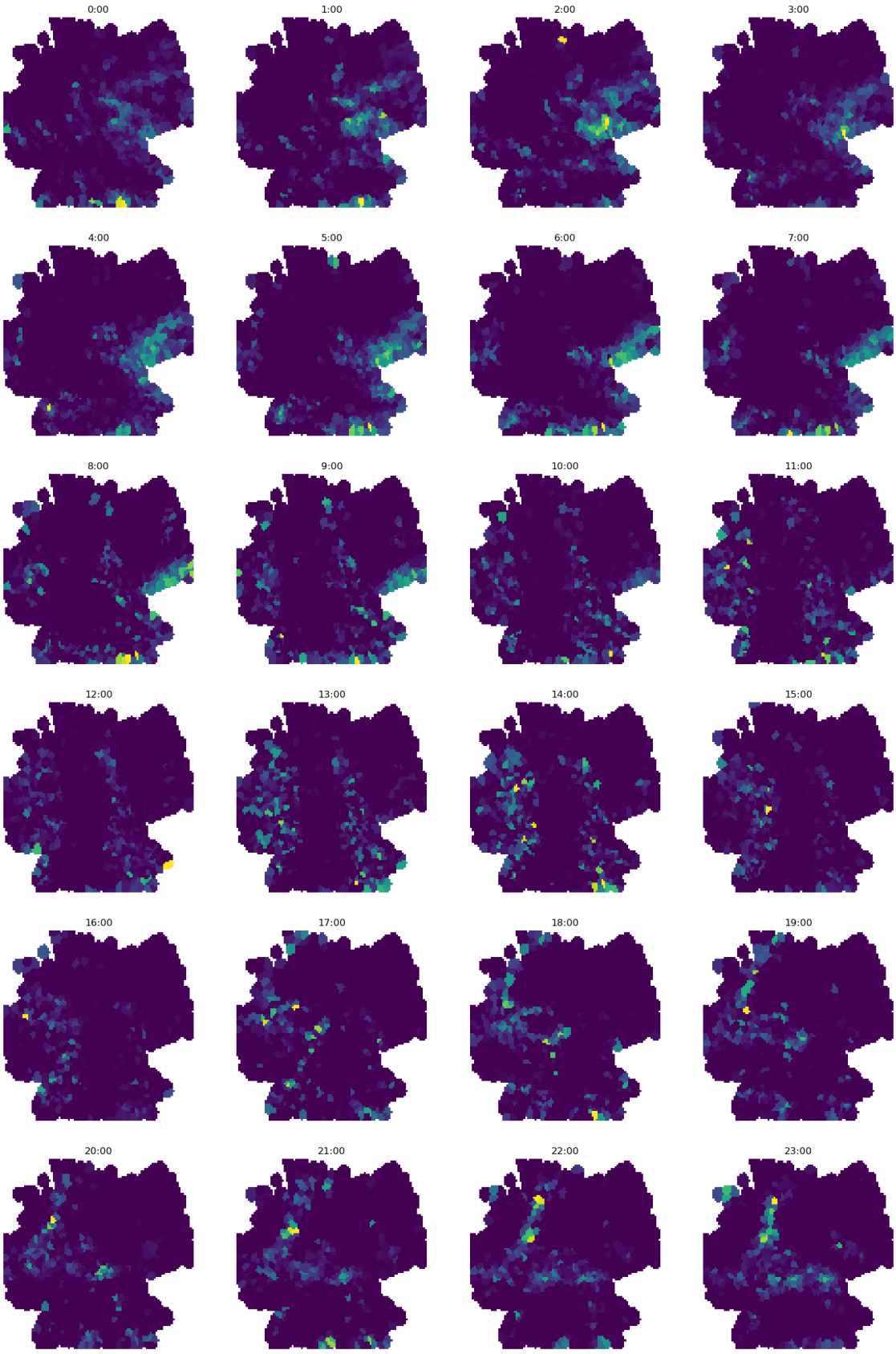
[6]:
```
data = load('data03/griddata.npz')
geolat = data['geolat']
geolong = data['geolong']
ind = data['ind']
del data
```

[8]:
```
interp_points = []
for i in range(100):
    for j in range(100):
        if ind[i][j]:
            interp_points.append([geolong[0][j], geolat[i][0]])
```

```
grid = np.array(interp_points)
sample_points = np.array(df_final[['geo_length', 'geo_width']])
```

[9]:
```
def fill_and_reshape(xs):
    index = 0
    res = []
    for i in range(100):
        for j in range(100):
            if ind[i][j]:
                res.append(xs[index])
                index += 1
            else:
                res.append(np.nan)
    return np.flipud((np.array(res)).reshape(100,100))
```

[10]:
```
fig, ax = plt.subplots(6, 4, figsize=(20, 30))
for h in range(24):
    axi = ax[h // 4][h % 4]
    values = np.array(df_final[f'hour_{int(h)}'])
    res = interp.griddata(sample_points, values, grid, method='nearest')
    res = fill_and_reshape(res)
    axi.imshow(res, cmap='viridis')
    axi.set_title(f'{h}:00')
    axi.axis('off')
plt.show()
```


0:00
1:00
2:00
3:00
4:00
5:00
6:00
7:00
8:00
9:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00

[10]:
```python
import numpy as np
import plotly.graph_objects as go

# vertices
points = np.array(
    [[0,0,0],[0,1,0],[0,1,1],[0,0,1],
     [1,0,0],[1,1,0],[1,1,1],[1,0,1]],
    dtype=float
)

triangles = np.array([
    [0, 1, 5], [0, 5, 4],
    [3, 7, 6], [3, 6, 2],
    [0, 3, 2], [0, 2, 1],
    [4, 5, 6], [4, 6, 7],
    [0, 4, 7], [0, 7, 3],
    [1, 2, 6], [1, 6, 5]
])

fig_cube = go.Figure(
    data=go.Mesh3d(
        x=points[:,0], y=points[:,1], z=points[:,2],
        i=triangles[:,0], j=triangles[:,1], k=triangles[:,2],
        color="royalblue", opacity=0.5, flatshading=True
    )
)
fig_cube.update_layout(title="Unit cube - 12-triangle surface mesh")
fig_cube.show()
```

[11]:
```python
def disk_mesh(n_r=15, n_th=40):
    """Return (points, triangles) approximating the unit disk z=0."""
    r = np.linspace(0, 1, n_r)                  # radii
    th = np.linspace(0, 2*np.pi, n_th, endpoint=False)  # angles
    R, T = np.meshgrid(r, th, indexing="ij")    # rectangular grid

    # polar → cartesian (z = 0)
```

```
    x = (R*np.cos(T)).ravel()
    y = (R*np.sin(T)).ravel()
    z = np.zeros_like(x)

    pts = np.column_stack([x, y, z])

    # connectivity: two triangles per quad in (r, ) grid
    tri_list = []
    for ir in range(n_r-1):
        for it in range(n_th):
            a  = ir*n_th + it
            b  = a + n_th                          # next radius ring
            a1 = ir*n_th + (it+1) % n_th           # wrap angle
            b1 = a1 + n_th
            tri_list += [[a, a1, b1], [a, b1, b]]

    return pts, np.array(tri_list, dtype=int)

disk_pts, disk_tri = disk_mesh()

fig_disk = go.Figure(
    go.Mesh3d(
        x=disk_pts[:,0], y=disk_pts[:,1], z=disk_pts[:,2],
        i=disk_tri[:,0], j=disk_tri[:,1], k=disk_tri[:,2],
        color="tomato", opacity=0.6
    )
).update_layout(title="Triangular mesh of unit disk (z=0)")
fig_disk.show()
```

[12]:
```
def cylinder_side(n_z=20, n_th=60):
    z = np.linspace(0, 1, n_z)
    th = np.linspace(0, 2*np.pi, n_th, endpoint=False)
    Z, T = np.meshgrid(z, th, indexing="ij")

    x = np.cos(T).ravel()
    y = np.sin(T).ravel()
```

```
    z = Z.ravel()

    pts = np.column_stack([x, y, z])

    tri = []
    for iz in range(n_z-1):
        for it in range(n_th):
            a  = iz*n_th + it
            b  = a + n_th
            a1 = iz*n_th + (it+1) % n_th
            b1 = a1 + n_th
            tri += [[a, a1, b1], [a, b1, b]]

    return pts, np.array(tri, int)

side_pts, side_tri = cylinder_side()
top_pts, top_tri = disk_mesh()
top_pts[:,2] = 1.0                      # lift to z = 1
bot_pts, bot_tri = disk_mesh()          # z=0 already

# Offset triangle indices before merging
bot_tri_off = bot_tri
top_tri_off = top_tri + len(bot_pts)
side_tri_off = side_tri + len(bot_pts) + len(top_pts)

all_pts = np.vstack([bot_pts, top_pts, side_pts])
all_tri = np.vstack([bot_tri_off, top_tri_off, side_tri_off])

fig_cyl = go.Figure(
    go.Mesh3d(
        x=all_pts[:,0], y=all_pts[:,1], z=all_pts[:,2],
        i=all_tri[:,0], j=all_tri[:,1], k=all_tri[:,2],
        color="seagreen", opacity=0.65
    )
).update_layout(title="Closed cylinder (side + top/bottom)")
fig_cyl.show()
```

[ ]: