# Solutions to Homework 2

## Question 1: 10 points.

**Problem Statement:** A laboratory experiment is conducted on 1030 specimens to determine the compressive strength (MPa) of concrete as a function of age (days) and various ingredients (e.g., cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate).

The experimental data (see python-code.d/data/materials/concrete-strength-data.csv) comprises eight (quantitative) input parameters:

```
Parameter             Description
-----------------------------------------------
Cement                kg in a m3 mixture.
Blast Furnace Slag    kg in a m3 mixture.
Fly Ash               kg in a m3 mixture.
Water                 kg in a m3 mixture.
Superplasticizer      kg in a m3 mixture.
Coarse Aggregate      kg in a m3 mixture.
Fine Aggregate        kg in a m3 mixture.
Age                   day (1 -- 365).
-----------------------------------------------
```

and one output:

```
Parameter             Description
-----------------------------------------------------------
Concrete strength     Concrete compressive strength (MPa).
-----------------------------------------------------------
```

**What to do?** Write a Python program that will:

1. Read the experimental test results from a file `concrete-strength-data.csv` into a Pandas dataframe.
2. Extract numpy arrays from the dataframe for age (days) and concrete compressive strength (MPa).
3. Compute and print the range of parameter values for each input (e.g., Cement content, Blast Furnace Slag, Fly Ash, Water, Superplasticizer, Coarse Aggregate, Fine Aggregate and Age.

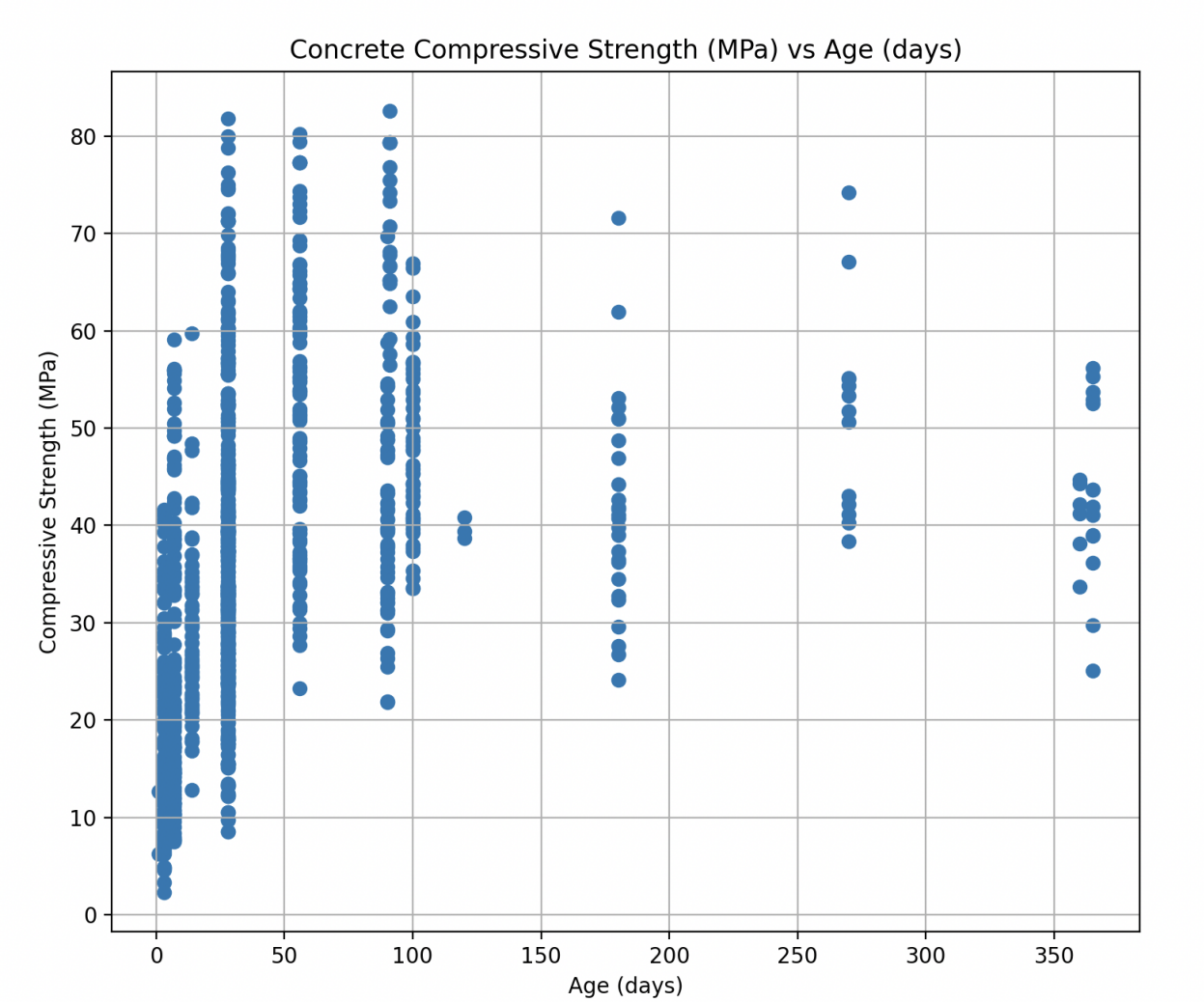



Figure 1: Scatter chart of concrete compressive strength (MPa) vs age (days).

4. Compute and print the maximum, minimum, and average concrete compressive strengths.

5. Create a scatter chart of concrete strength (MPa) vs age (days). See Figure 1.

6. Use the Pandas cut function (i.e., google `pd.cut()`) to organize the strength data into intervals: 0–25, 25–50, 50–75, and 75–100.

7. Generate a histogram of concrete compressive strength (MPa).

8. Generate the cumulative probability distribution for concrete compressive strength, and then create a stair-step graph of "cumulative frequency" versus "compressive strength."

**Note 1.** The average value of the experimental results can be computed using Python's buildin functions. The "cumulative frequency" versus "compressive strength" is given by

$$\text{Cumulative frequency(y)} = \int_0^y p(x)dx \tag{1}$$

where $p(x)$ is the probability distribution of concrete compressive strengths. The matplotlib functions `plt.hist(.)` and `plt.step(.)` create histogram and stair-step graphs.

**Note 2.** See python-code.d/pandas/TestMaterialsA36Steel.py for a very similar problem setup and solution.

---

**Python Source Code:**

```
# ==================================================================================
# TestMaterialsConcreteStrength02.py: Read, process and visualize experimental
# data on compressive strength of concrete.
#
# For details, see: data/material/concrete-strength-readme.txt
#
# Written by: Mark Austin                                              March 2025
# ==================================================================================

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

from pandas import DataFrame
from pandas import read_csv

# ================================================================
# Main function ...
# ================================================================
```

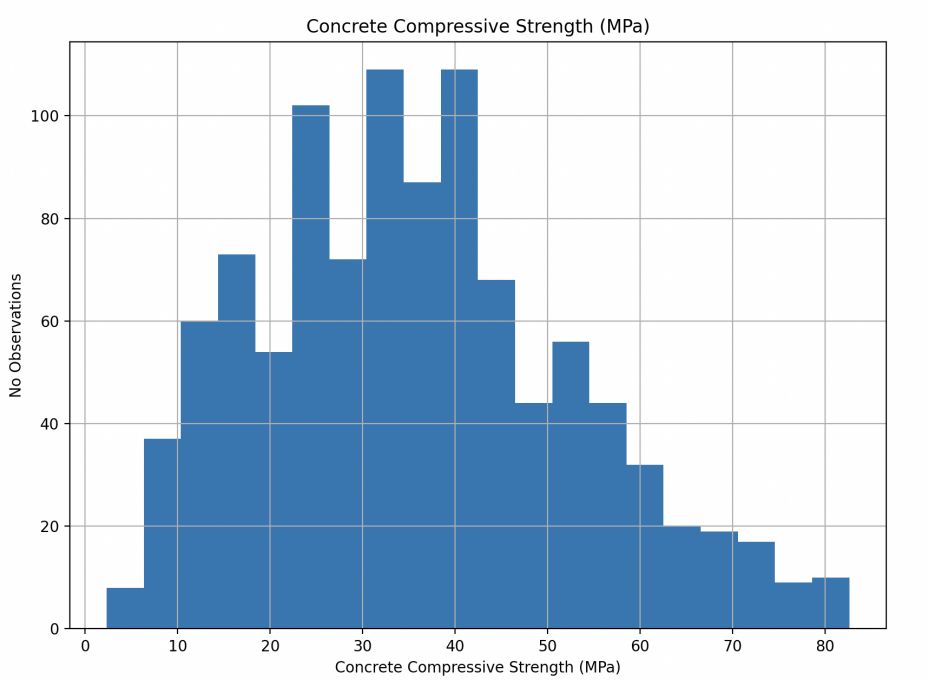



Figure 2: Histogram of concrete compressive strength (MPa).

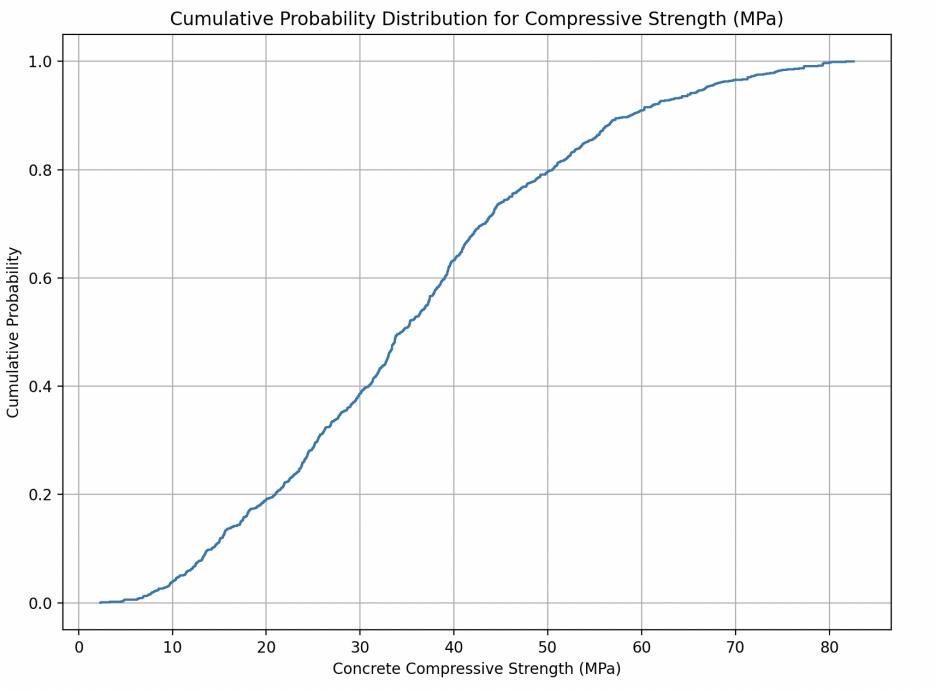



Figure 3: Cumulative distribution of concrete compressive strength (MPa).

```
def main():
    print("--- Enter TestMaterialsConcreteStrength02.main()     ... ");
    print("--- ================================================ ... ");
    print("");

    # Load and print dataset

    print("--- ");
    print("--- Part 01: Load data/materials/concrete-strength-data.csv into Pandas ... ");
    print("--- ");

    df = pd.read_csv('data/materials/concrete-strength-data.csv')
    print(df)

    # Dataframe info and shape ...

    print( df.info() )
    print( df.shape )

    print("--- ");
    print("--- Part 02: Extract numpy arrays from dataframe columns ... ");
    print("--- ");

    # Input parameters ...

    age      = np.array ( df['Age'].values )

    cement   = np.array ( df['Cement'].values )
    blastf   = np.array ( df['Blast Furnace Slag'].values )
    flyash   = np.array ( df['Fly Ash'].values )
    water    = np.array ( df['Water'].values )
    superp   = np.array ( df['Superplasticizer'].values )
    coarseag = np.array ( df['Coarse Aggregate'].values )
    fineag   = np.array ( df['Fine Aggregate'].values )

    # Output parameter ...

    strength = np.array ( df['Concrete Compressive Strength'].values )

    print("--- ");
    print("--- Part 03: Range of Input/Output Parameter Values ...");
    print("--- ");

    print("--- Input: Age: ");
    print("---       Min = {:5.2f} days ...".format(min(age)) );
    print("---       Max = {:5.2f} days ...".format(max(age)) );

    print("--- Input: Cement: ");
    print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(cement)) );
    print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(cement)) );

    print("--- Input: Blast Furnace Slag: ");
    print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(blastf)) );
    print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(blastf)) );
```

```
print("--- Input: Fly Ash:");
print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(flyash)) );
print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(flyash)) );

print("--- Input: Water:");
print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(water)) );
print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(water)) );

print("--- Input: Superplasticizer:");
print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(superp)) );
print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(superp)) );

print("--- Input: Coarse Aggregate:");
print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(coarseag)) );
print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(coarseag)) );

print("--- Input: Fine Aggregate:");
print("---       Min = {:5.2f} kg in m3 mixture ...".format(min(fineag)) );
print("---       Max = {:5.2f} kg in m3 mixture ...".format(max(fineag)) );

print("--- Output: Concrete Compressive Strength: ");
print("---       Min = {:5.2f} MPa ...".format(min(strength)) );
print("---       Max = {:5.2f} MPa ...".format(max(strength)) );

print("--- ");
print("--- Part 04: Basic statistics on concrete age/compressive strength ...");
print("--- ");

print("--- Min age     = {:5.2f} days ...".format(min(age)) );
print("--- Max age     = {:5.2f} days ...".format(max(age)) );
print("--- Average age = {:5.2f} days ...".format(sum(age)/len(age)) );
print("--- ");
print("--- Min strength     = {:5.2f} MPa ...".format(min(strength)) );
print("--- Max strength     = {:5.2f} MPa ...".format(max(strength)) );
print("--- Average strength = {:5.2f} MPa ...".format(sum(strength)/len(strength)) );

print("--- ");
print("--- Part 05: Transform data arrays into lists ...");
print("--- ");

age.tolist()
strength.tolist()

print("--- ");
print("--- Part 06: Scatter chart of concrete strength (MPa) vs age (days) ...");
print("--- ");

plt.scatter(age, strength )
plt.xlabel('Age (days)');
plt.ylabel('Compressive Strength (MPa)');
plt.title('Concrete Compressive Strength (MPa) vs Age (days)')
plt.grid()
plt.show()

print("--- ");
```

```
    print("--- Part 07: Organize strength data into intervals: 0-25, 25-50, 50-75, 75-100 ... ");
    print("--- ");

    age_sorted = np.sort( age )
    strength_sorted = np.sort( strength )

    sinterval = [ "0-20", "20-40", "40-60", "60-80", "80-100" ]
    steel_strength_intervals = pd.cut( strength_sorted, [ 0, 20, 40, 60, 80, 100 ], labels = sinterva

    # Retrieve interval categories and codes ...

    labels     = steel_strength_intervals.codes
    categories = steel_strength_intervals.categories

    # Systematically print the interval for each category ...

    for index in range(len(strength_sorted)):
        label_index = labels[index]
        print( strength_sorted[index], label_index, categories[label_index] )

    print("--- ");
    print("--- Part 08: Create histogram of Concrete Compressive Strengths (MPa) ... ");
    print("--- ");

    nbins = 20;
    plt.hist( strength_sorted, nbins );
    plt.title('Concrete Compressive Strength (MPa)');
    plt.xlabel('Concrete Compressive Strength (MPa)');
    plt.ylabel('No Observations');
    plt.grid()
    plt.show()

    print("--- ");
    print("--- Part 09: Generate cumulative frequency data and graph ... ");
    print("--- ");

    # Generate cumulative probability distribution ....

    npoints = len( strength_sorted );
    print("--- No data points = {:d} ...".format( npoints ));
    cumulative_probability = np.linspace( 0.0, 1.0, npoints, endpoint=True );

    # Step plot of cumulative probability vs yield strength ...

    plt.step( strength_sorted, cumulative_probability );
    plt.title('Cumulative Probability Distribution for Compressive Strength (MPa)');
    plt.xlabel('Concrete Compressive Strength (MPa)');
    plt.ylabel('Cumulative Probability');
    plt.grid()
    plt.show()

    print("--- ============================================== ... ");
    print("--- Leave TestMaterialsConcreteStrength02.main()   ... ");

# call the main method ...
```

```
if __name__ == "__main__":
    main()
```

**Program Output:** The abbreviated textual output is:

```
--- Enter TestMaterialsConcreteStrength02.main()     ...
--- ============================================== ...
---
--- Part 01: Load data/materials/concrete-strength-data.csv into Pandas ...
---
      Cement  Blast Furnace Slag  ...  Age  Concrete Compressive Strength
0      540.0                0.0  ...   28                          79.99
1      540.0                0.0  ...   28                          61.89
2      332.5              142.5  ...  270                          40.27
3      332.5              142.5  ...  365                          41.05
4      198.6              132.4  ...  360                          44.30
...      ...                ...  ...  ...                            ...
1025   276.4              116.0  ...   28                          44.28
1026   322.2                0.0  ...   28                          31.18
1027   148.5              139.4  ...   28                          23.70
1028   159.1              186.7  ...   28                          32.77
1029   260.9              100.5  ...   28                          32.40

[1030 rows x 9 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Cement                         1030 non-null   float64
 1   Blast Furnace Slag             1030 non-null   float64
 2   Fly Ash                        1030 non-null   float64
 3   Water                          1030 non-null   float64
 4   Superplasticizer               1030 non-null   float64
 5   Coarse Aggregate               1030 non-null   float64
 6   Fine Aggregate                 1030 non-null   float64
 7   Age                            1030 non-null   int64
 8   Concrete Compressive Strength  1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
None
(1030, 9)
---
--- Part 02: Extract numpy arrays from dataframe columns ...
---
--- Part 03: Range of Input/Output Parameter Values ...
---
--- Input: Age:
---       Min =  1.00 days ...
---       Max = 365.00 days ...
--- Input: Cement:
---       Min = 102.00 kg in m3 mixture ...
```

```
---       Max = 540.00 kg in m3 mixture ...
--- Input: Blast Furnace Slag:
---       Min =  0.00 kg in m3 mixture ...
---       Max = 359.40 kg in m3 mixture ...
--- Input: Fly Ash:
---       Min =  0.00 kg in m3 mixture ...
---       Max = 200.10 kg in m3 mixture ...
--- Input: Water:
---       Min = 121.80 kg in m3 mixture ...
---       Max = 247.00 kg in m3 mixture ...
--- Input: Superplasticizer:
---       Min =  0.00 kg in m3 mixture ...
---       Max = 32.20 kg in m3 mixture ...
--- Input: Coarse Aggregate:
---       Min = 801.00 kg in m3 mixture ...
---       Max = 1145.00 kg in m3 mixture ...
--- Input: Fine Aggregate:
---       Min = 594.00 kg in m3 mixture ...
---       Max = 992.60 kg in m3 mixture ...
--- Output: Concrete Compressive Strength:
---       Min =  2.33 MPa ...
---       Max = 82.60 MPa ...
---
--- Part 04: Basic statistics on concrete age/compressive strength ...
---
--- Min age     =  1.00 days ...
--- Max age     = 365.00 days ...
--- Average age = 45.66 days ...
---
--- Min strength     =  2.33 MPa ...
--- Max strength     = 82.60 MPa ...
--- Average strength = 35.82 MPa ...
---
--- Part 05: Transform data arrays into lists ...
---
--- Part 06: Scatter chart of concrete strength (MPa) vs age (days) ...
---
--- Part 07: Organize strength data into intervals: 0-25, 25-50, 50-75, 75-100 ...
---
2.33 0 0-20
3.32 0 0-20
4.57 0 0-20
4.78 0 0-20

.... lines of output removed ...

79.3 3 60-80
79.4 3 60-80
79.99 3 60-80
80.2 4 80-100
81.75 4 80-100
82.6 4 80-100
---
--- Part 08: Create histogram of Concrete Compressive Strengths (MPa) ...
---
```

```
--- Part 09: Generate cumulative frequency data and graph ...
---
--- No data points = 1030 ...
--- ============================================== ...
--- Leave TestMaterialsConcreteStrength02.main()   ...
```

Clearly, by itself age (days) is **not a good predictor** of concrete compressive strength (MPa).

## Question 2: 10 points.

**Problem Statement:** With only 24 counties, Maryland is a relatively small state. And yet, according to the American Road and Transportation Builders Association, there are 5,484 highway bridges in Maryland, and 250 of them (or 4.6 percent) are classified as structurally deficient.

Highway bridges in Maryland are given one of three ratings: GOOD, FAIR and POOR (or worse). The term structurally deficient means that one or more of the bridges key elements – superstructure, substructure, foundation – are in poor condition (or worse). This question takes a first step toward understanding: (1) how many bridges there are in each county?, (2) which counties contain bridges that have a POOR condition rating?, and (3) who is responsible for the maintenance of these bridges? To get a handle on these concerns we will need a map of Maryland counties and a test program to read and process highway bridge inventory, including condition ratings, location, ownership and maintenance.

**Test Program 01: Maryland Counties.** The test program:

python-code.d/io/TestReadMarylandCountiesGEOJSON01.py reads data from:

python-code.d/data/geography/maryland/Maryland-County-Boundaries-Generalized.geojson

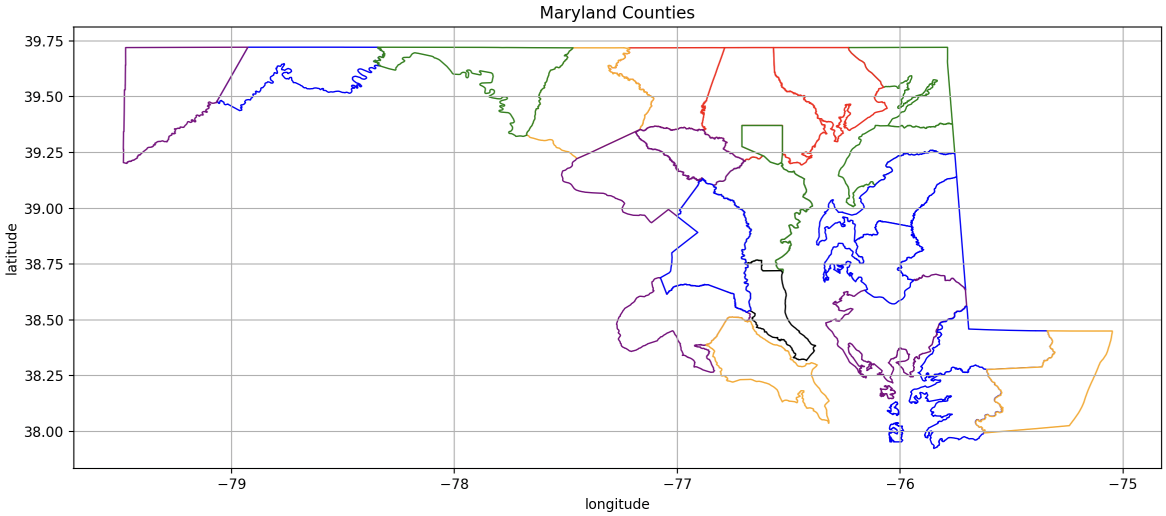



Figure 4: 24 counties in Maryland, including Baltimore City.

and then creates a dictionary of key, value pairs, connecting names of the counties (keys) to WKT (well known text) representations for geometry of the county boundary (values). Individual counties are loaded into GeoPandas, assigned colors, and visualized as shown Figure 4.

US counties are referenced by FIPS (Federal Information Processing Standard), a 5-digit code to rep-

resent a a state (first two digits) followed by a county (next three digits). For Maryland the details are as follows:

```
Maryland State Code Prefix:  24

COUNTY FIPS Codes
     001 Allegany County            025 Harford County
     003 Anne Arundel County        027 Howard County
     510 Baltimore City             029 Kent County
     005 Baltimore County           031 Montgomery County
     009 Calvert County             033 Prince George's County
     011 Caroline County            035 Queen Anne's County
     013 Carroll County             037 St.Mary's County
     015 Cecil County               039 Somerset County
     017 Charles County             041 Talbot County
     019 Dorchester County          043 Washington County
     021 Frederick County           045 Wicomico County
     023 Garrett County             047 Worcester County
```

**Test Program 02: Highway Bridge Inventory.** The test program:

python-code.d/io/TestReadHighwayBridgeSHP01.py reads data from:

python-code.d/data/bridges/maryland/Bridge_Condition_NHS_2017.shp ...

a collection of 1,932 highway bridges in Maryland. Data for each bridge is organized into 28 columns:

```
Column      Header   Description:
-------------------------------------------------------------------------
     0  Item_8_Str   Structure number.
     1  Item_5_D_R   Route number of the inventory route.
     2  Item_3_Cou   County code.
     3  Item_6a_Fe   Features Intersected.
     4  Item_7_Fac   Facility Carried by Structure.
     5  Item_9_Loc   Location
     6  Item_11_Mi   Kilometerpoint (location of bridge on base highway network).
     7  Item_12_Ba   Base highway network.
     8  Item_16_La   Latitude (xx degrees xx minutes xx.xx seconds).
     9  Item_17_Lo   Longitude (xx degrees xx minutes xx.xx seconds).
    10  Item_21_Ma   Maintenance Responsibility
    11  Item_22_Ow   Owner
    12  Item_26_Fu   Functional Classification of Inventory Route (2 digits).
    13  Item_48_Sp   Length of Maximum Span (xxx meters).
    14  Item_49_St   Structure Length (xxx meters).
    15  Item_51_Wi   Bridge Roadway Width, Curb-to-Curb (m)
    16  Item_52_Wi   Bridge Deck Width (m)
    17  Item_58_Co   Bridge Deck Condition.
    18  Item_59_Co   Superstructure Condition.
    19  Item_60_Co   Substructure Condition.
    20  Item_61_Co   Channel and Channel Protection Condition.
```

```
    21  Item_62_Co   Culverts Conditioon.
    22  Item_104_H   Highway System of Inventory Route.
    23  Item_105_F   Federal Lands Highways
    24  MinimumCon   Minumum Condition.
    25  ConditionR   Condition Rating
    26  DeckArea_S   Deck Area (m^2).
    27  geometry     Location of bridge: POINT (lat, long) ...
```

A sample of output is:

```
--- Bridge 199 ...
---   Structure Number: 100000160162015 ...
---   Route Number of Inventory Route: 00095 ...
---   County Code: 033 ...
---   Features Intersected: SUITLAND ROAD ...
---   Facility Carried by Structure: IS 95 IL ...
---   Location: 1.71 MILES SOUTH OF MD 4 ...
---   Kilometerpoint: 0014576 ...
---   Base highway network: 1 ...
---   Latitude (degree mins secs): 38491861 ...
---   Longitude (degree mins secs): 076531591 ...
---   Maintenance Responsibility: 01 ...
---   Owner: 01 ...
---   Functional Classification of Inventory Route: 11 ...
---   Length of Maximum Span (m): 00125 ...
---   Structure Length (m): 000448 ...
---   Bridge Roadway Width, Curb-to-Curb (m): 0207 ...
---   Bridge Deck Width (m): 0213 ...
---   Bridge Deck Condition: 4 ...
---   Superstructure Condition: 5 ...
---   Substructure Condition: 5 ...
---   Channel and Channel Protection Condition: N ...
---   Culverts Conditioon: N ...
---   Highway System of Inventory Route: 1 ...
---   Federal Lands Highways: 0 ...
---   Minimum Condition: 4 ...
---   Bridge Rating: POOR ...
---   Deck Area (m^2): 954.24 ...
---   Geometry POINT (-76.88776469841079 38.82183509305251) ...
```

Additional information on the codes (e.g., county code: 033; maintenance responsibility: 66) are given by FIPS, and tables within Items 22 and 23 of FHWA-PD-96-001, Recording and Coding Guide for the Structure Inventory and Appraisal of the Nation's Bridges.

**What to do?** Write a Python program that will:

1. Create dictionaries linking: (1) county codes to county names, and (2) maintenance agency codes to agency names responsible for maintenance, e.g.,

```
County code:      033   --->  Prince George's County.
Maintenance code:  01   --->  State Highway Agency.
```

2. Load the contents of Bridge_Condition_NHS_2017.shp. Create a plot of the bridges within PG County – each bridge can be drawn as a single point (e.g., POINT (-76.88 38.82)) in GeoPandas.

3. Identify the number of bridges within each county having GOOD, FAIR and POOR ratings. Output these numbers in a tidy table.

4. Generate a list of bridges having a POOR rating, along with details of ownership and and maintenance responsibility.

**Note 1:** For item 2, I originally had in mind a plot for all of the bridges in Maryland, but I think that the result will just be a sea of blue dots. Focusing on a single county might give a picture that is more reasonable.

**Note 2:** For item 3, there are a number of ways of approaching the problem. Perhaps the simplest approach is to create three dictionaries – poor condition bridges, fair condition bridges, and good condition bridges, with the key being the county and the value being the number of bridges in the county.

---

**Python Source Code:**

```
# ========================================================================
# TestReadHighwayBridge02.py: Preliminary analysis of 1,932 highway bridges
# in Maryland. Step-by-step details:
#
# --- Read GEOJSON datafile for county boundaries of Maryland.
# --- Read and process shp file containing data on
#     highway bridges in Maryland.
# --- Create dictionary of FIPS codes for MD counties ...
# --- Create dictionary of maintenance codes for highway bridge repair.
# --- Create dictionary of maintenance responsibility ...
# --- Create plot of bridges in PG county.
#
# Written by: Mark Austin                                    March 2025
# ========================================================================

import numpy as np
import pandas as pd
import geopandas

import math
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

from pandas import DataFrame
from pandas import Series

import geopandas as gpd

from prettytable import PrettyTable

# ==============================
# main method ...
# ==============================

def main():
    print("--- Enter TestReadHighwayBridge02.main()     ... ");
    print("--- ======================================== ... ");
    print("--- ");

    # Part 01: Create dictionary of FIPS codes for MD counties ...
    # ------------------------------------------------------------

    print("--- ");
    print("--- Part 01: Create dictionary of FIPS codes for MD counties ... ");
    print("--- ------------------------------------------------------------ ");
    print("--- ");

    mdcounty = {};
    mdcounty['001'] = "Allegany County";
    mdcounty['003'] = "Anne Arundel County";
    mdcounty['510'] = "Baltimore City";
    mdcounty['005'] = "Baltimore County";
    mdcounty['009'] = "Calvert County";
    mdcounty['011'] = "Caroline County";
    mdcounty['013'] = "Carroll County";
    mdcounty['015'] = "Cecil County";
    mdcounty['017'] = "Charles County";
    mdcounty['019'] = "Dorchester County";
    mdcounty['021'] = "Frederick County";
    mdcounty['023'] = "Garrett County";
    mdcounty['025'] = "Hartford County";
    mdcounty['027'] = "Howard County";
    mdcounty['029'] = "Kent County";
    mdcounty['031'] = "Montgomery County";
    mdcounty['033'] = "Prince George's County";
    mdcounty['035'] = "Queen Anne's County";
    mdcounty['037'] = "St. Mary's County";
    mdcounty['039'] = "Somerset County";
    mdcounty['041'] = "Talbot County";
    mdcounty['043'] = "Washington County";
    mdcounty['045'] = "Wicomico County";
    mdcounty['047'] = "Worcester County";

    # Print dictionary of counties ....
```

```
print("--- Dictionary of Counties: ");
countyKeyList = list( mdcounty.keys() )
for key in countyKeyList:
    print("--- key = {:s} --> value = {:s} ...".format( key, mdcounty.get(key) ) );

# Part 02: Dictionary of condition ratings.
# -----------------------------------------

print("--- ");
print("--- Part 02: Dictionary of Condition Ratings ... ");
print("--- ------------------------------------------- ");
print("--- ");

conditionrating = {};
conditionrating['N'] = "Not Applicable";
conditionrating['9'] = "Excellent Condition";
conditionrating['8'] = "Very Good Condition (no problems noted)";
conditionrating['7'] = "Good Condition (some minor problems)";
conditionrating['6'] = "Satisfactory Condition (structural elements show minor deterioration)";
conditionrating['5'] = "Fair Condition";
conditionrating['4'] = "Poor Condition";
conditionrating['3'] = "Serious Condition";
conditionrating['2'] = "Critical Condition (advanced deterioration of primary structural elements
conditionrating['1'] = "Imminent Failure Condition (bridge is closed to traffic)";
conditionrating['0'] = "Failed Condition (out of service -- beyond corrective action)";

# Print dictionary of condition ratings ...

print("--- Dictionary of Condition Ratings:");
conditionratingKeyList = list( conditionrating.keys() )
for key in conditionratingKeyList:
    print("--- key = {:s} --> value = {:s} ...".format( key, conditionrating.get(key) ) );

# Part 03: Dictionary of maintenance responsibility ...
# -----------------------------------------------------

print("--- ");
print("--- Part 03: Create dictionary of maintenance responsibility ... ");
print("--- ------------------------------------------------------------ ");
print("--- ");

maintenanceresponsibility = {};
maintenanceresponsibility['01'] = "State Highway Agency";
maintenanceresponsibility['02'] = "County Highway Agency";
maintenanceresponsibility['03'] = "Town or Township Highway Agency";
maintenanceresponsibility['04'] = "City or Municipal Highway Agency";
maintenanceresponsibility['11'] = "State Park, Forest, or Reservation Agency";
maintenanceresponsibility['12'] = "Local Park, Forest, or Reservation Agency";
maintenanceresponsibility['21'] = "Other State Agencies";
maintenanceresponsibility['25'] = "Other Local Agencies";
maintenanceresponsibility['26'] = "Private (other than railroad)";
maintenanceresponsibility['27'] = "Railroad";
maintenanceresponsibility['31'] = "State Toll Authority";
maintenanceresponsibility['32'] = "Local Toll Authority";
```

```
maintenanceresponsibility['60'] = "Other Federal Authority";
maintenanceresponsibility['61'] = "Indian Tribal Government";
maintenanceresponsibility['62'] = "Bureau of Indian Affairs";
maintenanceresponsibility['63'] = "Bureau of Fish and Wildlife";
maintenanceresponsibility['64'] = "US Forest Service";
maintenanceresponsibility['66'] = "National Park Service";
maintenanceresponsibility['67'] = "Tennessee Valley Authority";
maintenanceresponsibility['68'] = "Bureau of Land Management";
maintenanceresponsibility['69'] = "Bureau of Reclamation";
maintenanceresponsibility['70'] = "Corps of Engineers (Civil)";
maintenanceresponsibility['71'] = "Corps of Engineers (Military)";
maintenanceresponsibility['72'] = "Air Force";
maintenanceresponsibility['73'] = "Navy/Marines";
maintenanceresponsibility['74'] = "Army";
maintenanceresponsibility['75'] = "NASA";
maintenanceresponsibility['76'] = "Metropolitan Washington Airports Service";
maintenanceresponsibility['80'] = "Unknown";

# Print dictionary of maintenance responsibility`...

print("--- Dictionary of Maintenance Responsibilities");
maintenanceresponsibilityKeyList = list( maintenanceresponsibility.keys() )
for key in maintenanceresponsibilityKeyList:
    print("--- key = {:s} --> value = {:s} ...".format( key, maintenanceresponsibility.get(key) )

# Part 04: Dictionary of bridge owners ...
# ----------------------------------------

print("--- ");
print("--- Part 04: Dictionary of bridge owners ... ");
print("--- ---------------------------------------- ");
print("--- ");

bridgeowner = {};
bridgeowner['01'] = "State Highway Agency";
bridgeowner['02'] = "County Highway Agency";
bridgeowner['03'] = "Town or Township Highway Agency";
bridgeowner['04'] = "City or Municipal Highway Agency";
bridgeowner['11'] = "State Park, Forest, or Reservation Agency";
bridgeowner['12'] = "Local Park, Forest, or Reservation Agency";
bridgeowner['21'] = "Other State Agencies";
bridgeowner['25'] = "Other Local Agencies";
bridgeowner['26'] = "Private (other than railroad)";
bridgeowner['27'] = "Railroad";
bridgeowner['31'] = "State Toll Authority";
bridgeowner['32'] = "Local Toll Authority";
bridgeowner['60'] = "Other Federal Authority";
bridgeowner['61'] = "Indian Tribal Government";
bridgeowner['62'] = "Bureau of Indian Affairs";
bridgeowner['63'] = "Bureau of Fish and Wildlife";
bridgeowner['64'] = "US Forest Service";
bridgeowner['66'] = "National Park Service";
bridgeowner['67'] = "Tennessee Valley Authority";
bridgeowner['68'] = "Bureau of Land Management";
bridgeowner['69'] = "Bureau of Reclamation";
```

```
bridgeowner['70'] = "Corps of Engineers (Civil)";
bridgeowner['71'] = "Corps of Engineers (Military)";
bridgeowner['72'] = "Air Force";
bridgeowner['73'] = "Navy/Marines";
bridgeowner['74'] = "Army";
bridgeowner['75'] = "NASA";
bridgeowner['76'] = "Metropolitan Washington Airports Service";
bridgeowner['80'] = "Unknown";

# Print dictionary of maintenance responsibility`...

print("--- Dictionary of Bridge Owners");
bridgeownerKeyList = list( bridgeowner.keys() )
for key in bridgeownerKeyList:
    print("--- key = {:s} --> value = {:s} ...".format( key, bridgeowner.get(key) ) );

# Part 05: Load dataset of Maryland counties ...
# ----------------------------------------------

print("--- ");
print("--- Part 05: Load dataset of Maryland counties into GeoPandas ... ");
print("--- ------------------------------------------------------------ ");
print("--- ");

# Read dataset into geopandas

datafile01 = "data/geography/maryland/Maryland-County-Boundaries-Generalized.geojson";
counties = gpd.read_file( datafile01 )
gdf01    = gpd.GeoDataFrame ( counties )

# Create dictionary of US State Boundaries ...

countyboundaries = {}

# Assemble dictionary of county boundaries ...

i = 1
for index, row in gdf01.iterrows():
    county00 = str( row[0] );  # <-- county ID ...
    county01 = str( row[1] );  # <-- county ...
    county02 = str( row[2] );  # <-- district ...
    county03   = row[3];       # <-- county FIP ...
    county04   = row[4];       # <-- county number ...
    geometry09 = row[9];       # <-- geometry ...

    # Add county boundary to dictionary of boundaries ...

    countyboundaries[ county01 ] = str( geometry09 );

    # Summary of data ...

    print("--- ");
    print("--- County {:s}: {:s} ...".format(county00, county01 ) );
    print("---   District: {:s} ...".format( county02 ) );
    print("---   WKT Geometry: {:s} ...".format( str(geometry09)[0:70] ) );     # <-- This works
```

```
print("--- ");
print("--- Part 06: Load highway bridge data file ... ");
print("--- ---------------------------------------- ");
print("--- ");

datapath01 = "data/bridges/maryland/Bridge_Condition_NHS_2017.shp";
mdbridges = geopandas.read_file( datapath01 )

# Dataframe description ...

print( mdbridges.describe() )
print( mdbridges.info() )
print( mdbridges.shape )

# Dataset column headings ...

for col in mdbridges.columns:
   print(col)

print("--- ");
print("--- Part 07: Assemble table of highway bridge conditions ... ");
print("--- ---------------------------------------------------- ");
print("--- ");

table01 = PrettyTable(["County", "GOOD", "FAIR", "POOR"])
table01.title = "Bridge Conditions in Maryland";

# Loop over counties; count bridges in good, fair and poor condition ...

for county in mdcounty.keys():
    for cond in ["GOOD"]:
        ng = len( mdbridges[ mdbridges["Item_3_Cou"].isin([county]) &
                  mdbridges["ConditionR"].isin([cond])] )

    for cond in ["FAIR"]:
        nf = len( mdbridges[ mdbridges["Item_3_Cou"].isin([county]) &
                  mdbridges["ConditionR"].isin([cond])] )

    for cond in ["POOR"]:
        np = len( mdbridges[ mdbridges["Item_3_Cou"].isin([county]) &
                  mdbridges["ConditionR"].isin([cond])] )

    table01.add_row( [ mdcounty[county], ng, nf, np ], divider=True);

print(table01);

print("--- ");
print("--- Part 08: Print details of bridges in POOR condition ... ");
print("--- ---------------------------------------------------- ");
print("--- ");

poorconditionbridges = mdbridges[ mdbridges["ConditionR"].isin([ "POOR" ])]

i = 1
```

```
for index, row in poorconditionbridges.iterrows():
    bridge00   = str( row.iloc[0] );
    bridge01   = str( row.iloc[1] );
    bridge02   = str( row.iloc[2] );
    bridge03   = str( row.iloc[3] );
    bridge04   = str( row.iloc[4] );
    bridge05   = str( row.iloc[5] );
    bridge06   = str( row.iloc[6] );
    bridge07   = str( row.iloc[7] );
    bridge08   = str( row.iloc[8] );
    bridge09   = str( row.iloc[9] );
    bridge10   = str( row.iloc[10] );
    bridge11   = str( row.iloc[11] );
    bridge12   = str( row.iloc[12] );
    bridge13   = str( row.iloc[13] );
    bridge14   = str( row.iloc[14] );
    bridge15   = str( row.iloc[15] );
    bridge16   = str( row.iloc[16] );
    bridge17   = str( row.iloc[17] );
    bridge18   = str( row.iloc[18] );
    bridge19   = str( row.iloc[19] );
    bridge20   = str( row.iloc[20] );
    bridge21   = str( row.iloc[21] );
    bridge22   = str( row.iloc[22] );
    bridge23   = str( row.iloc[23] );
    bridge24   = str( row.iloc[24] );
    bridge25   = str( row.iloc[25] );
    bridge26   = str( row.iloc[26] );
    geometry27 = row.iloc[27];

    # Print details of bridge in poor condition ....

    print("--- ");
    print("--- Bridge {:d} ...".format(i) );
    print("---   Structure Number: {:s} ...".format( bridge00 ) );
    print("---   County Code: {:s} ...".format( bridge02 ) );
    print("---   County: {:s} ...".format( mdcounty.get(bridge02) ) );
    print("---   Maintenance: {:s} ...".format( maintenanceresponsibility.get(bridge10) ) );
    print("---   Geometry {:s} ...".format( str(geometry27) ) );
    print("---   Owner: {:s} ...".format( bridgeowner.get(bridge11) ) );
    print("---   Bridge Rating: {:s} ...".format( bridge25 ) );
    print("---   Minimum Condition: {:s} ...".format( bridge24 ) );
    print("---   Bridge Deck Condition: {:s} ...".format( bridge17 ) );
    print("---   Superstructure Condition: {:s} ...".format( bridge18 ) );
    print("---   Substructure Condition: {:s} ...".format( bridge19 ) );
    print("---   Channel and Channel Protection Condition: {:s} ...".format( bridge20 ) );
    print("---   Culverts Condition: {:s} ...".format( bridge21 ) );

    i = i + 1;

print("--- ");
print("--- Part 09: Plot of Bridges in PG County ... ");
print("--- ---------------------------------------- ");
print("--- ");
```

```
    # Part 1: Retrieve boundary of Prince George's County ...

    print("--- Retrieve boundary of Prince George's County ... ");

    key = 'Prince George\'s'
    pgCountyBoundary = countyboundaries.get(key)
    print("--- PG County = {:s} --> {:s} ...".format( key, str( pgCountyBoundary)[0:70] ) )

    g01   = gpd.GeoSeries.from_wkt( [ str(pgCountyBoundary) ]);
    df17  = g01.to_frame(name='geometry');
    geo01 = gpd.GeoDataFrame(df17, geometry='geometry')
    geo01.geometry;

    # Part 2: Setup plot ...

    ax = geo01.plot( color='red', edgecolor='black')
    ax.set_aspect('equal')

    # Create plot of PG county boundary ...

    geo01.plot( ax=ax, color='white', edgecolor='blue',   linewidth = 1 )

    # Part 3: Filter dataframe to only keep bridges in PG county ...

    options = ['033']
    pgbridges = mdbridges [ mdbridges['Item_3_Cou'].isin(options) ].copy()

    print('--- PG bridges dataframe :\n', pgbridges )

    # Create plot of PG county bridges ...

    gdf02 = gpd.GeoDataFrame( pgbridges, crs='epsg:4326')
    gdf02.geometry

    gdf02.plot( ax=ax, color='green', edgecolor='black')

    plt.xlabel('longitude')
    plt.ylabel('latitude')
    plt.title("Bridges in PG County")
    plt.grid(True)
    plt.show()

    print("--- ======================================== ... ");
    print("--- Leave TestReadHighwayBridge02.main()     ... ");

# ================================================
# call the main method ...
# ================================================

main()
```

**Program Output:** The abbreviated textual output is:

```
--- Enter TestReadHighwayBridge02.main()     ...
```

```
--- ======================================== ...
---
--- Part 01: Create dictionary of FIPS codes for MD counties ...
--- ------------------------------------------------------------
---
--- Dictionary of Counties:
--- key = 001 --> value = Allegany County ...
--- key = 003 --> value = Anne Arundel County ...
--- key = 510 --> value = Baltimore City ...

.... output removed ...

--- key = 033 --> value = Prince George's County ...

.... output removed ...

--- key = 047 --> value = Worcester County ...
---
--- Part 02: Dictionary of Condition Ratings ...
--- ----------------------------------------
---
--- Dictionary of Condition Ratings:
--- key = N --> value = Not Applicable ...
--- key = 9 --> value = Excellent Condition ...
--- key = 8 --> value = Very Good Condition (no problems noted) ...
--- key = 7 --> value = Good Condition (some minor problems) ...
--- key = 6 --> value = Satisfactory Condition (structural elements show minor deterioration) ...
--- key = 5 --> value = Fair Condition ...
--- key = 4 --> value = Poor Condition ...
--- key = 3 --> value = Serious Condition ...
--- key = 2 --> value = Critical Condition (advanced deterioration of primary structural elements) ...
--- key = 1 --> value = Imminent Failure Condition (bridge is closed to traffic) ...
--- key = 0 --> value = Failed Condition (out of service -- beyond corrective action) ...
---
--- Part 03: Create dictionary of maintenance responsibility ...
--- ------------------------------------------------------------
---
--- Dictionary of Maintenance Responsibilities
--- key = 01 --> value = State Highway Agency ...
--- key = 02 --> value = County Highway Agency ...
--- key = 03 --> value = Town or Township Highway Agency ...
--- key = 04 --> value = City or Municipal Highway Agency ...
--- key = 11 --> value = State Park, Forest, or Reservation Agency ...
--- key = 12 --> value = Local Park, Forest, or Reservation Agency ...
--- key = 21 --> value = Other State Agencies ...
--- key = 25 --> value = Other Local Agencies ...
--- key = 26 --> value = Private (other than railroad) ...
--- key = 27 --> value = Railroad ...
--- key = 31 --> value = State Toll Authority ...
--- key = 32 --> value = Local Toll Authority ...
--- key = 60 --> value = Other Federal Authority ...
--- key = 61 --> value = Indian Tribal Government ...
--- key = 62 --> value = Bureau of Indian Affairs ...
--- key = 63 --> value = Bureau of Fish and Wildlife ...
--- key = 64 --> value = US Forest Service ...
```

```
--- key = 66 --> value = National Park Service ...
--- key = 67 --> value = Tennessee Valley Authority ...
--- key = 68 --> value = Bureau of Land Management ...
--- key = 69 --> value = Bureau of Reclamation ...
--- key = 70 --> value = Corps of Engineers (Civil) ...
--- key = 71 --> value = Corps of Engineers (Military) ...
--- key = 72 --> value = Air Force ...
--- key = 73 --> value = Navy/Marines ...
--- key = 74 --> value = Army ...
--- key = 75 --> value = NASA ...
--- key = 76 --> value = Metropolitan Washington Airports Service ...
--- key = 80 --> value = Unknown ...
---
--- Part 04: Dictionary of bridge owners ...
--- ---------------------------------------
---
--- Dictionary of Bridge Owners
--- key = 01 --> value = State Highway Agency ...
--- key = 02 --> value = County Highway Agency ...
--- key = 03 --> value = Town or Township Highway Agency ...
--- key = 04 --> value = City or Municipal Highway Agency ...
--- key = 11 --> value = State Park, Forest, or Reservation Agency ...
--- key = 12 --> value = Local Park, Forest, or Reservation Agency ...
--- key = 21 --> value = Other State Agencies ...
--- key = 25 --> value = Other Local Agencies ...
--- key = 26 --> value = Private (other than railroad) ...
--- key = 27 --> value = Railroad ...
--- key = 31 --> value = State Toll Authority ...
--- key = 32 --> value = Local Toll Authority ...
--- key = 60 --> value = Other Federal Authority ...
--- key = 61 --> value = Indian Tribal Government ...
--- key = 62 --> value = Bureau of Indian Affairs ...
--- key = 63 --> value = Bureau of Fish and Wildlife ...
--- key = 64 --> value = US Forest Service ...
--- key = 66 --> value = National Park Service ...
--- key = 67 --> value = Tennessee Valley Authority ...
--- key = 68 --> value = Bureau of Land Management ...
--- key = 69 --> value = Bureau of Reclamation ...
--- key = 70 --> value = Corps of Engineers (Civil) ...
--- key = 71 --> value = Corps of Engineers (Military) ...
--- key = 72 --> value = Air Force ...
--- key = 73 --> value = Navy/Marines ...
--- key = 74 --> value = Army ...
--- key = 75 --> value = NASA ...
--- key = 76 --> value = Metropolitan Washington Airports Service ...
--- key = 80 --> value = Unknown ...
---
--- Part 05: Load dataset of Maryland counties into GeoPandas ...
--- ------------------------------------------------------------
---
--- County 1: Allegany ...
---   District: 6.0 ...
---   WKT Geometry: MULTIPOLYGON (((-78.38474173299994 39.624211004000074, -78.38467783799 ...
---
--- County 2: Anne Arundel ...
```

```
---   District: 5.0 ...
---   WKT Geometry: MULTIPOLYGON (((-76.53694920699996 38.848001004000025, -76.52852073799 ...

... output removed ...

--- County 24: Worcester ...
---   District: 1.0 ...
---   WKT Geometry: MULTIPOLYGON (((-75.09252954399994 38.32319208900003, -75.092733840999 ...

--- Part 06: Load highway bridge data file ...
--- ----------------------------------------

         DeckArea_S
count   1932.000000
mean    2775.213354
std     6121.349448
min       19.200000
25%      632.622500
50%     1180.025000
75%     2548.180000
max    96827.520000
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 1932 entries, 0 to 1931
Data columns (total 28 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Item_8_Str  1932 non-null   object
 1   Item_5_D_R  1932 non-null   object
 2   Item_3_Cou  1932 non-null   object
 3   Item_6a_Fe  1932 non-null   object
 4   Item_7_Fac  1932 non-null   object
 5   Item_9_Loc  1932 non-null   object
 6   Item_11_Mi  1932 non-null   object
 7   Item_12_Ba  1894 non-null   object
 8   Item_16_La  1932 non-null   object
 9   Item_17_Lo  1932 non-null   object
 10  Item_21_Ma  1932 non-null   object
 11  Item_22_Ow  1932 non-null   object
 12  Item_26_Fu  1932 non-null   object
 13  Item_48_Sp  1932 non-null   object
 14  Item_49_St  1932 non-null   object
 15  Item_51_Wi  1932 non-null   object
 16  Item_52_Wi  1932 non-null   object
 17  Item_58_Co  1932 non-null   object
 18  Item_59_Co  1932 non-null   object
 19  Item_60_Co  1932 non-null   object
 20  Item_61_Co  1932 non-null   object
 21  Item_62_Co  1932 non-null   object
 22  Item_104_H  1932 non-null   object
 23  Item_105_F  1932 non-null   object
 24  MinimumCon  1932 non-null   object
 25  ConditionR  1932 non-null   object
 26  DeckArea_S  1932 non-null   float64
 27  geometry    1932 non-null   geometry
dtypes: float64(1), geometry(1), object(26)
```

```
memory usage: 422.8+ KB
None
(1932, 28)

--- Part 07: Assemble table of highway bridge conditions ...
--- -----------------------------------------------------
```

| Bridge Conditions in Maryland | | | | |
|---|---|---|---|---|
| County | GOOD | FAIR | POOR | |
| Allegany County | 32 | 47 | 1 | |
| Anne Arundel County | 69 | 109 | 0 | |
| Baltimore City | 36 | 184 | 16 | <--- Just the worst !!!! |
| Baltimore County | 89 | 227 | 5 | |
| Calvert County | 2 | 10 | 0 | |
| Caroline County | 1 | 8 | 0 | |
| Carroll County | 10 | 14 | 0 | |
| Cecil County | 7 | 39 | 1 | |
| Charles County | 6 | 6 | 0 | |
| Dorchester County | 2 | 5 | 0 | |
| Frederick County | 28 | 78 | 4 | |
| Garrett County | 4 | 23 | 1 | |
| Hartford County | 13 | 30 | 0 | |
| Howard County | 22 | 94 | 0 | |
| Kent County | 0 | 3 | 1 | |
| Montgomery County | 96 | 100 | 0 | |
| Prince George’s County | 110 | 162 | 6 | |
| Queen Anne’s County | 2 | 18 | 3 | |
| St. Mary’s County | 0 | 2 | 0 | |
| Somerset County | 1 | 9 | 0 | |
| Talbot County | 0 | 12 | 0 | |
| Washington County | 23 | 85 | 2 | |

```
+------------------------+------+------+------+
|    Wicomico County     |  9   |  35  |  0   |
+------------------------+------+------+------+
|    Worcester County    |  8   |  22  |  0   |
+------------------------+------+------+------+

--- Part 08: Print details of bridges in POOR condition ...
--- ------------------------------------------------------

--- Bridge 1 ...
---   Structure Number: 200000BC5103010 ...
---   County Code: 510 ...
---   County: Baltimore City ...
---   Maintenance: City or Municipal Highway Agency ...
---   Geometry POINT (-76.63380581671525 39.26919245034452) ...
---   Owner: City or Municipal Highway Agency ...
---   Bridge Rating: POOR ...
---   Minimum Condition: 4 ...
---   Bridge Deck Condition: 5 ...
---   Superstructure Condition: 4 ...
---   Substructure Condition: 4 ...
---   Channel and Channel Protection Condition: N ...
---   Culverts Condition: N ...

... details of individual bridges removed ...

--- Bridge 17 ...
---   County Code: 021 ...
---   County: Frederick County ...
---   Maintenance: State Highway Agency ...
---
--- Bridge 18 ...
---   County Code: 033 ...
---   County: Prince George's County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 19 ...
---   County Code: 510 ...
---   County: Baltimore City ...
---   Maintenance: City or Municipal Highway Agency ...
---
--- Bridge 20 ...
---   County Code: 005 ...
---   County: Baltimore County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 21 ...
---   County Code: 023 ...
---   County: Garrett County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 22 ...
```

```
---   County Code: 510 ...
---   County: Baltimore City ...
---   Maintenance: City or Municipal Highway Agency ...
---   Owner: City or Municipal Highway Agency ...
---
--- Bridge 23 ...
---   County Code: 035 ...
---   County: Queen Anne's County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 24 ...
---   County Code: 510 ...
---   County: Baltimore City ...
---   Maintenance: City or Municipal Highway Agency ...
---   Owner: City or Municipal Highway Agency ...
---
--- Bridge 25 ...
---   County Code: 021 ...
---   County: Frederick County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 26 ...
---   County Code: 033 ...
---   County: Prince George's County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 27 ...
---   County Code: 015 ...
---   County: Cecil County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 28 ...
---   County Code: 021 ...
---   County: Frederick County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 29 ...
---   County Code: 029 ...
---   County: Kent County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 30 ...
---   County Code: 510 ...
---   County: Baltimore City ...
---   Maintenance: City or Municipal Highway Agency ...
---   Owner: City or Municipal Highway Agency ...
---
--- Bridge 31 ...
---   County Code: 043 ...
```

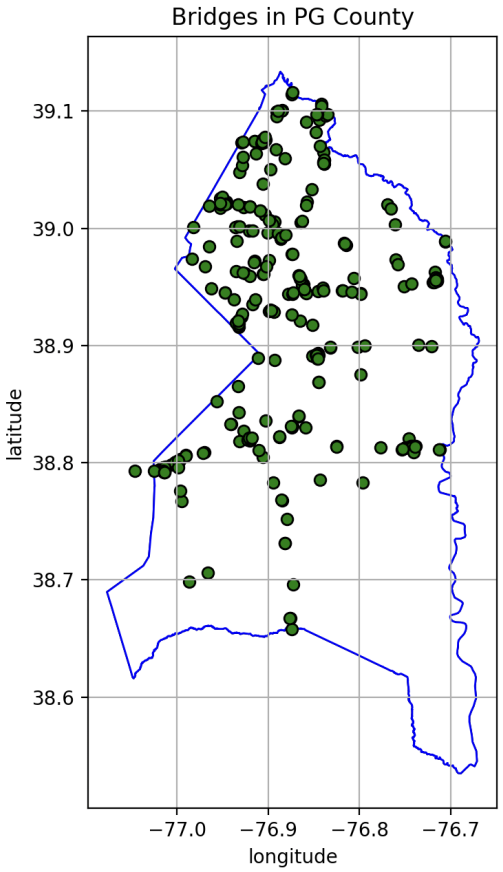
```
---   County: Washington County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 32 ...
---   County Code: 005 ...
---   County: Baltimore County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 33 ...
---   County Code: 035 ...
---   County: Queen Anne's County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 34 ...
---   County Code: 510 ...
---   County: Baltimore City ...
---   Maintenance: City or Municipal Highway Agency ...
---   Owner: City or Municipal Highway Agency ...
---
--- Bridge 35 ...
---   County Code: 510 ...
---   County: Baltimore City ...
---   Maintenance: City or Municipal Highway Agency ...
---   Owner: City or Municipal Highway Agency ...
---
--- Bridge 36 ...
---   County Code: 001 ...
---   County: Allegany County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 37 ...
---   County Code: 035 ...
---   County: Queen Anne's County ...
---   Maintenance: State Highway Agency ...
---   Owner: State Highway Agency ...
---
--- Bridge 38 ...
---   County Code: 510 ...
---   County: Baltimore City ...
---   Maintenance: City or Municipal Highway Agency ...
---   Owner: City or Municipal Highway Agency ...
---
--- Bridge 39 ...
---   County Code: 510 ...
---   County: Baltimore City ...
---   Maintenance: City or Municipal Highway Agency ...
---   Owner: City or Municipal Highway Agency ...
---
--- Bridge 40 ...
---   County Code: 510 ...
---   County: Baltimore City ...
```

```
---   Maintenance: City or Municipal Highway Agency ...
---   Owner: City or Municipal Highway Agency ...

--- Part 09: Plot of Bridges in PG County ...
--- ----------------------------------------

--- Retrieve boundary of Prince George's County ...
--- PG County = Prince George's --> MULTIPOLYGON (((-77.03704928799993 38.712400983000066, -77.03105
--- PG bridges dataframe :
           Item_8_Str Item_5_D_R  ... DeckArea_S                    geometry
5     100000160269030      00197  ...     934.40  POINT (-76.76875 39.02009)
8     100000160046010      00214  ...    2597.85  POINT (-76.80153 38.89876)
...               ...        ...  ...        ...                         ...
1930  3530035P0000000      00001  ...     496.07   POINT (-76.83929 39.0559)
1931  3530033P0000000      00001  ...    1533.60  POINT (-76.90208 38.96722)

[278 rows x 28 columns]
--- ======================================== ...
--- Leave TestReadHighwayBridge02.main()     ...
```



## Question 3: 10 points.

**Problem Statement:** As shown in Figure 5 below, rectangles may be defined by the (x,y) coordinates of corner points that are diagonally opposite.

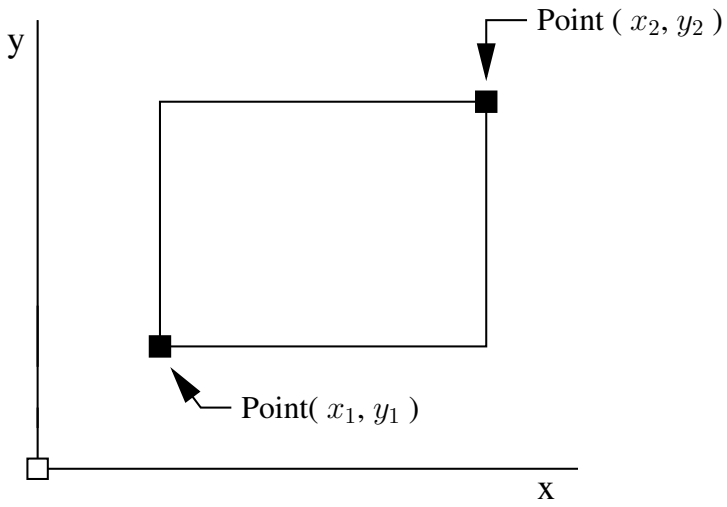



Figure 5: Definition of a rectangle via diagonally opposite corner points.

With this definition in place, the following script of code is a basic implementation of a class for creating and working with rectangle objects.

```
# ========================================================================
# Rectangle01.py: Very basic implementation of rectangle objects, where
# corner points are defined by varibles (x1, y1) and (x2, y2).
#
# Written by: Mark Austin                                    September, 2024
# ========================================================================

class Rectangle:

  def __init__(self, x1, y1, x2, y2 ):
    self.x1 = x1;
    self.y1 = y1;
    self.x2 = x2;
    self.y2 = y2;
    self.name = "";

  # Set rectangle name ...

  def setName(self, name ):
    self.name = name;

  # Compute perimeter of rectangle ..
```

```
  def getPerimeter (self):
     perimeter = 2*(abs(self.x2-self.x1) + abs(self.y2-self.y1))
     return perimeter

  # Compute area of rectangle ..

  def getArea (self):
     area = abs(self.x2-self.x1)*abs(self.y2-self.y1);
     return area

  # String represention of rectangle ...

  def __str__(self):
     rectangleinfo = [];

     ... details removed ...

     return "".join(rectangleinfo);
```

The Rectangle class uses variables x1, y1, x2, and y2 to define the corner points, and has a method to create rectangle objects (i.e, __init__), convert the details of a rectangle object into a string format (i.e., __str__), and compute the rectangle area and perimeter (i.e., getArea() and getPerimeter()).

A script of test program usage is as follows:

```
prompt >>
prompt >> python3 TestRectangle01.py
--- Enter TestRectangle01.main()    ...
--- ============================== ...
--- Part 1: Create and print rectangle A ...

--- Rectangle: A ...
--- -----------------------------------------------
---   Corner Point (x1,y1) = (  1.00,   2.00) ...
---   Corner Point (x2,y2) = (  3.00,   4.00) ...
---   Perimeter =   8.00 ...
---   Area      =   4.00 ...
--- -----------------------------------------------

--- Part 2: Create and print rectangle B ...

--- Rectangle: B ...
--- -----------------------------------------------
---   Corner Point (x1,y1) = (  0.00,   0.00) ...
---   Corner Point (x2,y2) = (  6.00,   5.00) ...
---   Perimeter =  22.00 ...
---   Area      =  30.00 ...
--- -----------------------------------------------

--- ============================== ...
--- Finished TestRectangle01.main()    ...
prompt >> exit
```

**Source Code:** Full details of the rectangle code and test program can be found in: python-code.d/classes/ ...

**Question:** Now suppose that instead of using the variables $x_1$, $y_1$, $x_2$ and $y_2$ to define the corner points, we use the class Point:

```
import math

class Point:

    def __init__(self, xCoord=0, yCoord=0):
        self.__xCoord = xCoord
        self.__yCoord = yCoord

    # get x coordinate

    def get_xCoord(self):
        return self.__xCoord

    ..... details of other functions removed ...
```

The appropriate modification for Rectangle is:

```
from Point import Point

class Rectangle:

   def __init__(self, x1, y1, x2, y2 ):
       self.pt1 = Point(x1,y1)     # <-- create lower corner point ...
       self.pt2 = Point(x2,y2)     # <-- create upper corner point ...

   ..... details rectangle removed ....
}
```

The arrangement of Rectangle and Point classes can be visualized as follows:


Figure 6: Test program script and classes in a rectangle system.

**What to do?** Fill in the missing details (i.e., constructors and __str__ method) of class Point. Modify the code in Rectangle to use the Point class. The resulting program should have essentially the same functionality as the original implementation (v1) of Rectangle.

---

**Python Source Code:** The source code is comprised of three Python files: Point.py, Rectangle.py and TestRectangle.py.

**Abbreviated Object Source Code:** Python.py

```
# =======================================================
# Point class that demonstrates overloading of operators
# for arithmetic and relational expressions.
#
# Modified by: Mark Austin                  October, 2024
# =======================================================

import math

class Point:

    def __init__(self, xCoord=0, yCoord=0):
        self.xCoord = xCoord
        self.yCoord = yCoord

    # Get/set X, Y coordinates

    def getX(self):
        return self.xCoord

    def setX(self, xCoord):
        self.xCoord = xCoord

    # Get/set Y coordinate

    def getY(self):
        return self.yCoord

    def setY(self, yCoord):
        self.yCoord = yCoord

    # Get current position

    def get_position(self):
        return self.__xCoord, self.__yCoord

    # Move x & y coordinates by p & q

    def move(self, p, q):
        self.xCoord += p
        self.yCoord += q

    # Compute distance between two points ...

    def distance(self, second):
        x_d = self.xCoord - second.xCoord
        y_d = self.yCoord - second.yCoord
        return (x_d**2 + y_d**2)**0.5

    # Return string represention of object ...

    def __str__(self):
        return "( %6.2f, %6.2f )" % ( self.xCoord, self.yCoord )
```

**Object Source Code:** Rectangle02.py

```
# ========================================================================
# Rectangle02.py: Very basic implementation of rectangle objects, where
# corner points are defined by varibles (x1, y1) and (x2, y2).
#
# Written by: Mark Austin                                    September, 2024
# ========================================================================

import math

from Point import Point

class Rectangle:

  def __init__(self, x1, y1, x2, y2 ):
    self.pt1 = Point(x1,y1)
    self.pt2 = Point(x2,y2)
    self.name = "";

  # Set rectangle name ...

  def setName(self, name ):
    self.name = name;

  # Compute perimeter of rectangle ..

  def getPerimeter (self):
     x1 = self.pt1.getX(); y1 = self.pt1.getY();
     x2 = self.pt2.getX(); y2 = self.pt2.getY();
     perimeter = 2*((x2 - x1) + abs(y2 - y1))
     return perimeter

  # Compute area of rectangle ..

  def getArea (self):
     x1 = self.pt1.getX(); y1 = self.pt1.getY();
     x2 = self.pt2.getX(); y2 = self.pt2.getY();
     area = abs( x2 - x1 )*abs( y2 - y1)
     return area

  # String represention of rectangle ...

  def __str__(self):
     rectangleinfo = [];
     rectangleinfo.append("--- Rectangle: {:s} ...  \n".format( self.name ));
     rectangleinfo.append("--- ------------------------------------------------ \n");
     rectangleinfo.append("---   Corner Point (x1,y1) = {:s} ... \n".format( self.pt1.__str__() ));
     rectangleinfo.append("---   Corner Point (x2,y2) = {:s} ... \n".format( self.pt2.__str__() ));
     rectangleinfo.append("---   Perimeter = {:6.2f} ... \n".format( self.getPerimeter()));
     rectangleinfo.append("---   Area      = {:6.2f} ... \n".format( self.getArea()));
     rectangleinfo.append("--- ------------------------------------------------ \n");
     return "".join(rectangleinfo);
```

**Test Program Source Code:** TestRectangle02.py

```
# ================================================================
# TestRectangle02.py: Exercise point() version of Rectangle.
#
# Written by: Mark Austin                              September, 2024
# ================================================================

from Rectangle02 import Rectangle;

# main method ...

def main():
    print("--- Enter TestRectangle02.main()       ... ");
    print("--- ================================== ... \n");

    print("--- Part 1: Create and print rectangle A ... \n");

    rectangleA = Rectangle( 1.0, 2.0, 3.0, 4.0 )
    rectangleA.setName("A")
    print(rectangleA)

    print("--- Part 2: Create and print rectangle B ... \n");

    rectangleB = Rectangle( 0.0, 0.0, 6.0, 5.0 )
    rectangleB.setName("B")
    print(rectangleB)

    print("--- ================================== ... ");
    print("--- Finished TestRectangle02.main()       ... ");

# call the main method ...

if __name__ == "__main__":
    main()
```

**Abbreviated Program Output:**

```
--- Enter TestRectangle02.main()       ...
--- ================================== ...

--- Part 1: Create and print rectangle A ...

--- Rectangle: A ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   1.00,   2.00 ) ...
---   Corner Point (x2,y2) = (   3.00,   4.00 ) ...
---   Perimeter =   8.00 ...
---   Area      =   4.00 ...
--- ------------------------------------------------

--- Part 2: Create and print rectangle B ...
```

```
--- Rectangle: B ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   0.00,   0.00 ) ...
---   Corner Point (x2,y2) = (   6.00,   5.00 ) ...
---   Perimeter =  22.00 ...
---   Area      =  30.00 ...
--- ------------------------------------------------

--- ================================== ...
--- Finished TestRectangle02.main()       ...
```

## Question 4: 20 points.

**Problem Statement:** The left-hand side of Figure 7 shows the essential details of a domain familiar to many children. One by one, rectangular blocks are stacked as high as possible until they come tumbling down – the goal, afterall, is to create a spectacular crash!!

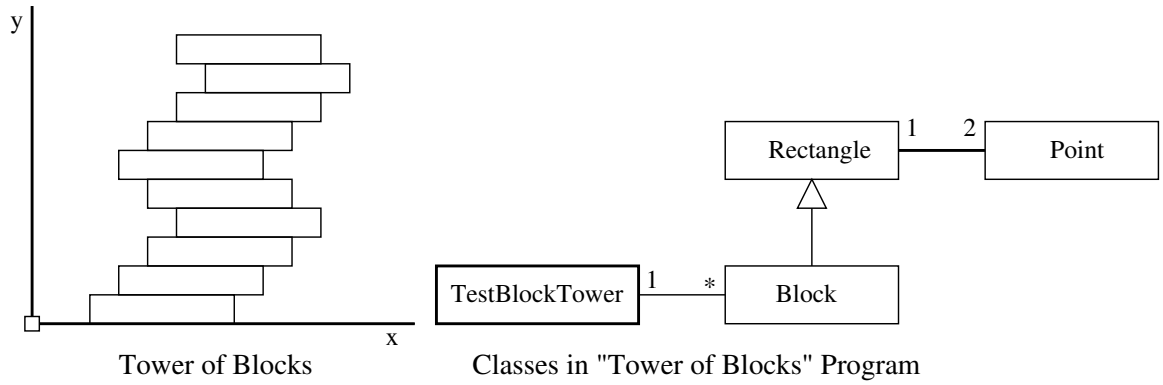



Figure 7: Schematic and classes for tower of blocks problem.

Suppose that we wanted to model this process and use engineering principles to predict incipient instability of the block tower. Consider the following observations:

**1.** Rather than start from scratch, it would make sense to create a Block class that inherits the properties of Rectangle (previous question), and adds details relevant to engineering analysis (e.g., the density of the block).

**2.** Then we could develop a BlockTower class that systematically assembles the tower, starting at the base and working upwards. At each step of the tower assembly, analysis procedures should make sure that the tower is still stable.

The right-hand side of Figure 7 shows the relatioship among the classes. One TestBlockTower program (1) will employ many blocks, as indicated by the asterik (*).

Develop a Python program that builds upon the Rectangle class written in the previous questions. The class Block should store the depth and density of the block – this will be important in determining the mass and centroid of each block. The TestBlockTower class will use block objects to build the tower. A straight forward way of modeling the block tower is with a List. After each block is added, the program should conduct a stability check. If the system is still stable, then add another block should be added. The simulation should cease when the tower of blocks eventually becomes unstable.

**Note.** To simplify the analysis, assume that adjacent blocks are firmly connected.

**Stability Considerations.** If the blocks are stacked perfectly on top of each other, then from a mathematical

standpoint the tower will never become unstable. In practice, this never happens. There is always a small offset and, eventually, it's the accumulation of offsets that leads to spectacular disaster.

For the purposes of this question, assume that blocks are five units wide, one unit high, and depth of one unit. When a new block is added, the block offset should be one unit. To make the question interesting, assume that four blocks are stacked with an offset to the right, then three blocks are added with an offset to the left, then four to the right, three to the left, and so forth. This sequence can be accomplished with the looping construct:

```
    # Compute incremental offset for i-th block .....

    offset = math.floor((BlockNo - 1)/5.0) + (BlockNo-1)%5;
    if ((BlockNo-1)%5 == 4 ):
       offset = offset - 2;
```

The tower will become unstable when the center of gravity of blocks above a particular level falls outside the edge of the supporting block.

**What to do?** Write a Python program that will:

1. Determine how many blocks can be added to the stack before it crashes.
2. Create a figure of the block configuration and centroid position immediately before collapse.

---

**Python Source Code:** Four files: Point.py, Vertex01.py, Block01.py, TestBlockTower01.py.

**Abbreviated Point Object Code:** Point02.py

```
# ======================================================
# Point class that demonstrates overloading of operators
# for arithmetic and relational expressions.
#
# Modified by: Mark Austin                 October, 2024
# ======================================================

import math

class Point:

    def __init__(self, xCoord=0, yCoord=0):
        self.xCoord = xCoord
        self.yCoord = yCoord

    # Get/set X coordinate
```

```
    def getX(self):
        return self.xCoord

    def setX(self, xCoord):
        self.xCoord = xCoord

    # Get/set Y coordinate

    def getY(self):
        return self.yCoord

    def setY(self, yCoord):
        self.yCoord = yCoord

    # Get current position

    def get_position(self):
        return self.__xCoord, self.__yCoord

    # change x & y coordinates by p & q

    def move(self, p, q):
        self.xCoord += p
        self.yCoord += q

    # compute distance between two points ...

    def distance(self, second):
        x_d = self.xCoord - second.xCoord
        y_d = self.yCoord - second.yCoord
        return (x_d**2 + y_d**2)**0.5

    # Overload Arithmetic operators ...
    # ---------------------------------------

    ... details removed ...

    # Overload Relational Operators ...
    # ---------------------------------------

    ... details removed ...

    # return string represention of object ...
    # ---------------------------------------

    def __str__(self):
        return "( %6.2f, %6.2f )" % ( self.xCoord, self.yCoord )
```

**Vertex Object Code:** Vertex01.py

```
# ============================================================
# Vertex.py: A vertex is a point with a label ...
```

```
# ============================================================

from Point import Point

class Vertex(Point):
   label = ""

   # Constructor method ...

   def __init__(self, x, y) :
     Point.__init__(self, x, y)
     self.label = ""

   # -----------------------------------------
   # Set/get label ...
   # -----------------------------------------

   def setLabel(self, label ):
     self.label = label

   def getLabel(self):
     return self.label

   # -----------------------------------------
   # Assemble string represention of Vertex ...
   # -----------------------------------------

   def __str__(self):
      vertexinfo = [];
      vertexinfo.append("\n");
      vertexinfo.append("--- Vertex: {:s} ... \n".format( self.getLabel()));
      vertexinfo.append("--- ------------------------------------------------ \n");
      vertexinfo.append("---   Coordinate: (x,y) = {:s} ... \n".format( Point.__self__()));
      vertexinfo.append("--- ------------------------------------------------ ");
      return "".join(vertexinfo);
```

**Block Object Code:** Block01.py

```
# ============================================================================
# Block01.py: A block is rectangle that has mass (density + thickness) ...
#
# Written by: Mark Austin                                    February, 2025
# ============================================================================

import math

from Rectangle02 import Rectangle
from Point import Point

from matplotlib.patches import Circle
from matplotlib.lines import Line2D

class Block (Rectangle):
```

```
density   = 0.0;
thickness = 0.0;

def __init__(self, x1, y1, x2, y2 ):
   Rectangle.__init__(self, x1, y1, x2, y2 );

# Set block density ...

def setDensity (self, density ):
   self.density = density;

# Set block thickness density ...

def setThickness (self, thickness ):
   self.thickness = thickness;

# Compute block mass and centroid ...

def getMass(self):
   volume = self.thickness * self.getArea();
   return self.density*volume;

def getCentroid(self):
    centroid = Point();

    centroid.setX( 1.0/2.0 * ( self.pt1.getX() + self.pt2.getX() ));
    centroid.setY( 1.0/2.0 * ( self.pt1.getY() + self.pt2.getY() ));
    return centroid;

# Draw block ...

def draw(self, ax):
    width = 0.1;

    x1 = self.pt1.getX();
    y1 = self.pt1.getY();
    x2 = self.pt2.getX();
    y2 = self.pt2.getY();

    # Draw block edges ...

    ax.add_line( Line2D( [x1, x1], [y1, y2] ) )
    ax.add_line( Line2D( [x1, x2], [y2, y2] ) )
    ax.add_line( Line2D( [x2, x2], [y2, y1] ) )
    ax.add_line( Line2D( [x2, x1], [y1, y1] ) )

    # Draw block vertices as small circles ...

    ax.add_patch( Circle( (x1, y1), width, facecolor='red') )
    ax.add_patch( Circle( (x2, y1), width, facecolor='red') )
    ax.add_patch( Circle( (x1, y2), width, facecolor='red') )
    ax.add_patch( Circle( (x2, y2), width, facecolor='red') )

# String represention of block ...
```

```
   def __str__(self):
      blockinfo = [];
      blockinfo.append("--- Block: {:s} ...  \n".format( self.name ));
      blockinfo.append("--- Density: {:f} ...  \n".format( self.density ));
      # blockinfo.append("--- Centroid: {:s} ...  \n".format( self.getCentroid().__str()__ ));
      blockinfo.append("--- ------------------------------------------------ \n");
      blockinfo.append("---   Corner Point (x1,y1) = {:s} ... \n".format( self.pt1.__str__() ));
      blockinfo.append("---   Corner Point (x2,y2) = {:s} ... \n".format( self.pt2.__str__() ));
      blockinfo.append("--- ------------------------------------------------ \n");
      return "".join(blockinfo);
```

**Test Program:** TestBlockTower01.py

```
# ======================================================================
# TestObjectBlockTower01.py: Assemble tower of blocks until they become
# unstable.
#
# Written by: Mark Austin                                 February, 2025
# ======================================================================

import math;
import matplotlib.pyplot as plt

from matplotlib.patches import Circle
from matplotlib.lines import Line2D

from Block01 import Block;

# main method ...

def main():
    print("--- Enter TestObjectBlockTower01.main() ... ");
    print("--- ================================== ... ");

    print("--- \n");
    print("--- Part 1: Create and print test block A ...");
    print("--- ===================================== ...");

    blockA = Block( 0.0, 0.0, 5.0, 1.0 )
    blockA.setName("A")
    blockA.setThickness( 1.0 )
    blockA.setDensity( 1.0 )

    print(blockA)
    print("--- Block mass  = {:6.2f} ...".format( blockA.getMass() ));
    print("--- CentroidX() = {:6.2f} ...".format( blockA.getCentroid().getX() ));
    print("--- CentroidY() = {:6.2f} ...".format( blockA.getCentroid().getY() ));

    print("--- ");
    print("--- Part 2: Simulate Block Tower ... ");
    print("--- ===================================== ... ");
    print("--- ");
```

```
tower = [];

blockTowerStable = True;
maxIterations = 20;
BlockNo       = 0;
towerCentroidX = 0.0;
towerCentroidY = 0.0;
while( blockTowerStable == True and BlockNo < maxIterations ):
   BlockNo = BlockNo + 1;

   print("--- ");
   print("--- Add Block No: {:d} ...".format( BlockNo ));
   print("--- ======================================== ... ");

   # Compute incremental offset for i-th block .....

   offset = math.floor((BlockNo - 1)/5.0) + (BlockNo-1)%5;
   if ((BlockNo-1)%5 == 4 ):
      offset = offset - 2;

   print("--- offset = {:d} ...".format( offset ));

   # Compute (x,y) coordinates of block vertices...

   x1 =  0.0 + offset;
   x2 =  5.0 + offset;
   y1 =  0.0 + BlockNo - 1;
   y2 =  1.0 + BlockNo - 1;

   # Create new block ...

   b = Block ( x1, y1, x2, y2 );
   b.setDensity( 1.0 );
   b.setThickness( 1.0 )
   print(b)

   # Add block to tower ....

   tower.append( b );

   # Compute (x,y) coordinates of tower centroid ...

   TotalMass = 0.0;
   FirstMomentX = 0.0;
   FirstMomentY = 0.0;

   for item in tower:
       TotalMass += item.getMass();
       FirstMomentX = FirstMomentX + item.getMass() * item.getCentroid().getX();
       FirstMomentY = FirstMomentY + item.getMass() * item.getCentroid().getY();

   print("--- Block Tower Analytics                 ...");
   print("--- ------------------------------------ ...");
   print("--- Total Mass     = {:6.2f} ...".format(TotalMass) );
   print("--- FirstMoment(X) = {:6.2f} ...".format( FirstMomentX ));
```

```
        print("--- FirstMoment(Y) = {:6.2f} ...".format( FirstMomentY ));
        print("--- Tower Centroid(X) = {:6.2f} ...".format( FirstMomentX/TotalMass ));
        print("--- Tower Centroid(Y) = {:6.2f} ...".format( FirstMomentY/TotalMass ));

        # Save tower centroid ...

        towerCentroidX = FirstMomentX/TotalMass;
        towerCentroidY = FirstMomentY/TotalMass;

        # Test for stability of tower ...

        print("--- ------------------------------------ ...");
        if ( FirstMomentX/TotalMass < 5.0 ):
           print("--- Tower of blocks is stable ...");
        else:
           blockTowerStable = False;
           print("");
           print("--- Crash!!");
           print("--- Tower of {:d} blocks is unstable".format(BlockNo) );

    print("--- ");
    print("--- Part 3: Draw Block Tower ... ");
    print("--- ==================================== ... ");

    # Define Matplotlib figure and axis

    fig, ax = plt.subplots()

    # Draw individual blocks in tower ...

    for block in tower:
        block.draw(ax);

    # Draw tower centroid ...

    width = 0.2;
    ax.add_patch( Circle( (towerCentroidX, towerCentroidY), width, facecolor='green') )

    # Create and show matplotlib graphic ...

    plt.title('Block Tower at Collapse')
    plt.ylabel('y')
    plt.xlabel('x')
    plt.ylim( -1, 15)
    plt.xlim( -1, 11)
    plt.grid(True)
    plt.show()

    print("--- ======================================= ... ");
    print("--- Finished TestObjectBlockTower01.main() ... ");

# call the main method ...

if __name__ == "__main__":
    main()
```

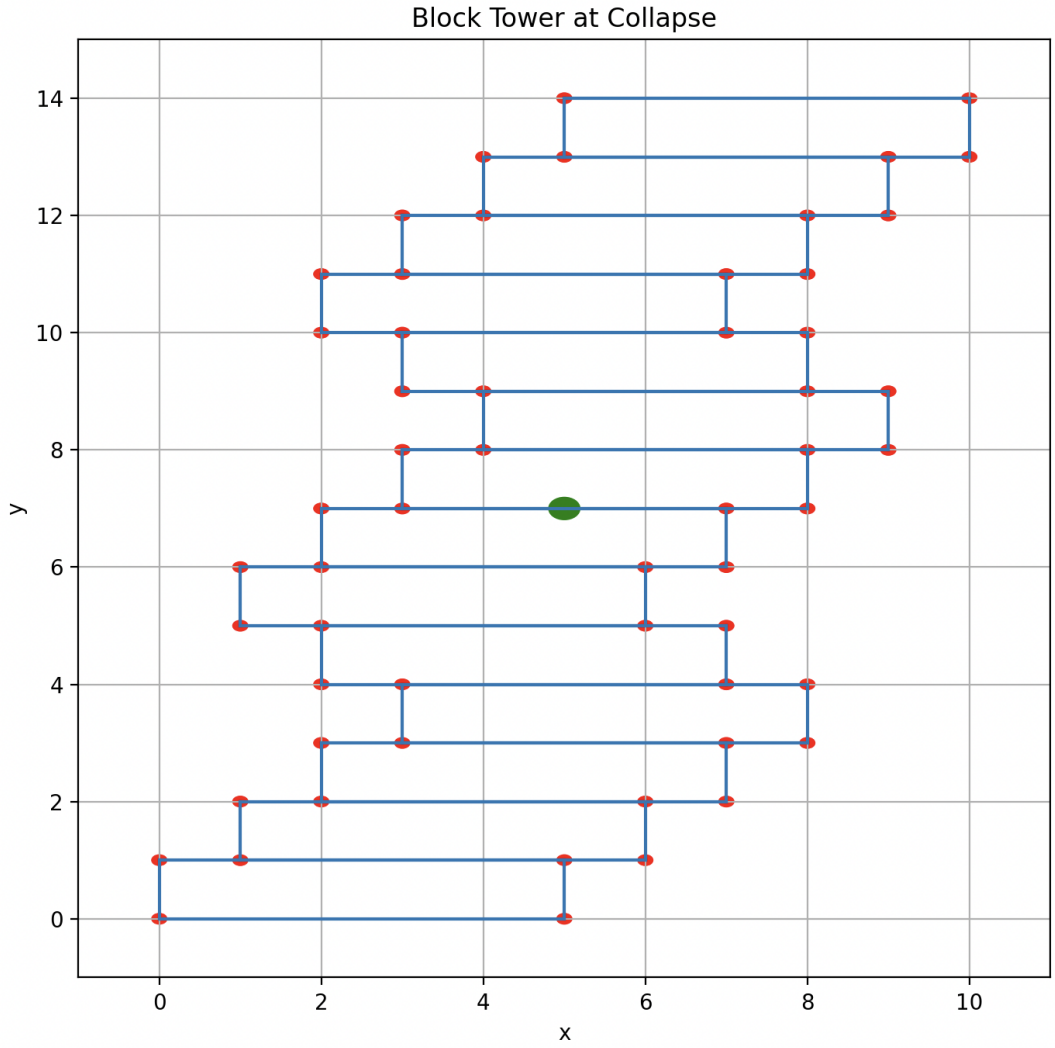



Figure 8: Block tower at collapse.

**Program Output:** The abbreviated textual output is:

```
--- Enter TestObjectBlockTower01.main() ...
--- ================================== ...

--- Part 1: Create and print test block A ...
--- ==================================== ...
--- Block: A ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   0.00,   0.00 ) ...
---   Corner Point (x2,y2) = (   5.00,   1.00 ) ...
--- ------------------------------------------------

--- Block mass  =   5.00 ...
--- CentroidX() =   2.50 ...
--- CentroidY() =   0.50 ...
---
--- Part 2: Simulate Block Tower ...
--- ==================================== ...
---
--- Add Block No: 1 ...
--- ======================================== ...
--- offset = 0 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   0.00,   0.00 ) ...
---   Corner Point (x2,y2) = (   5.00,   1.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                ...
--- ---------------------------------- ...
--- Total Mass     =   5.00 ...
--- FirstMoment(X) =  12.50 ...
--- FirstMoment(Y) =   2.50 ...
--- Tower Centroid(X) =   2.50 ...
--- Tower Centroid(Y) =   0.50 ...
--- ---------------------------------- ...
--- Tower of blocks is stable ...
---
--- Add Block No: 2 ...
--- ======================================== ...
--- offset = 1 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   1.00,   1.00 ) ...
---   Corner Point (x2,y2) = (   6.00,   2.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                ...
--- ---------------------------------- ...
--- Total Mass     =  10.00 ...
--- FirstMoment(X) =  30.00 ...
```

```
--- FirstMoment(Y) =  10.00 ...
--- Tower Centroid(X) =   3.00 ...
--- Tower Centroid(Y) =   1.00 ...
--- ------------------------------------ ...
--- Tower of blocks is stable ...
---
--- Add Block No: 3 ...
--- ======================================== ...
--- offset = 2 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   2.00,   2.00 ) ...
---   Corner Point (x2,y2) = (   7.00,   3.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                 ...
--- ------------------------------------ ...
--- Total Mass     =  15.00 ...
--- FirstMoment(X) =  52.50 ...
--- FirstMoment(Y) =  22.50 ...
--- Tower Centroid(X) =   3.50 ...
--- Tower Centroid(Y) =   1.50 ...
--- ------------------------------------ ...
--- Tower of blocks is stable ...
---
--- Add Block No: 4 ...
--- ======================================== ...
--- offset = 3 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   3.00,   3.00 ) ...
---   Corner Point (x2,y2) = (   8.00,   4.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                 ...
--- ------------------------------------ ...
--- Total Mass     =  20.00 ...
--- FirstMoment(X) =  80.00 ...
--- FirstMoment(Y) =  40.00 ...
--- Tower Centroid(X) =   4.00 ...
--- Tower Centroid(Y) =   2.00 ...
--- ------------------------------------ ...
--- Tower of blocks is stable ...
---
--- Add Block No: 5 ...
--- ======================================== ...
--- offset = 2 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   2.00,   4.00 ) ...
---   Corner Point (x2,y2) = (   7.00,   5.00 ) ...
--- ------------------------------------------------
```

```
--- Block Tower Analytics                ...
--- ----------------------------------- ...
--- Total Mass     =  25.00 ...
--- FirstMoment(X) = 102.50 ...
--- FirstMoment(Y) =  62.50 ...
--- Tower Centroid(X) =   4.10 ...
--- Tower Centroid(Y) =   2.50 ...
--- ----------------------------------- ...
--- Tower of blocks is stable ...
---
--- Add Block No: 6 ...
--- ======================================= ...
--- offset = 1 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   1.00,   5.00 ) ...
---   Corner Point (x2,y2) = (   6.00,   6.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                ...
--- ----------------------------------- ...
--- Total Mass     =  30.00 ...
--- FirstMoment(X) = 120.00 ...
--- FirstMoment(Y) =  90.00 ...
--- Tower Centroid(X) =   4.00 ...
--- Tower Centroid(Y) =   3.00 ...
--- ----------------------------------- ...
--- Tower of blocks is stable ...
---
--- Add Block No: 7 ...
--- ======================================= ...
--- offset = 2 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   2.00,   6.00 ) ...
---   Corner Point (x2,y2) = (   7.00,   7.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                ...
--- ----------------------------------- ...
--- Total Mass     =  35.00 ...
--- FirstMoment(X) = 142.50 ...
--- FirstMoment(Y) = 122.50 ...
--- Tower Centroid(X) =   4.07 ...
--- Tower Centroid(Y) =   3.50 ...
--- ----------------------------------- ...
--- Tower of blocks is stable ...
---
--- Add Block No: 8 ...
--- ======================================= ...
--- offset = 3 ...
--- Block:  ...
```

```
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   3.00,   7.00 ) ...
---   Corner Point (x2,y2) = (   8.00,   8.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                 ...
--- ------------------------------------ ...
--- Total Mass     =  40.00 ...
--- FirstMoment(X) = 170.00 ...
--- FirstMoment(Y) = 160.00 ...
--- Tower Centroid(X) =   4.25 ...
--- Tower Centroid(Y) =   4.00 ...
--- ------------------------------------ ...
--- Tower of blocks is stable ...
---
--- Add Block No: 9 ...
--- ======================================== ...
--- offset = 4 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   4.00,   8.00 ) ...
---   Corner Point (x2,y2) = (   9.00,   9.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                 ...
--- ------------------------------------ ...
--- Total Mass     =  45.00 ...
--- FirstMoment(X) = 202.50 ...
--- FirstMoment(Y) = 202.50 ...
--- Tower Centroid(X) =   4.50 ...
--- Tower Centroid(Y) =   4.50 ...
--- ------------------------------------ ...
--- Tower of blocks is stable ...
---
--- Add Block No: 10 ...
--- ======================================== ...
--- offset = 3 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   3.00,   9.00 ) ...
---   Corner Point (x2,y2) = (   8.00,  10.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                 ...
--- ------------------------------------ ...
--- Total Mass     =  50.00 ...
--- FirstMoment(X) = 230.00 ...
--- FirstMoment(Y) = 250.00 ...
--- Tower Centroid(X) =   4.60 ...
--- Tower Centroid(Y) =   5.00 ...
--- ------------------------------------ ...
--- Tower of blocks is stable ...
```

```
---
--- Add Block No: 11 ...
--- ======================================== ...
--- offset = 2 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   2.00,  10.00 ) ...
---   Corner Point (x2,y2) = (   7.00,  11.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                ...
--- ----------------------------------- ...
--- Total Mass     =  55.00 ...
--- FirstMoment(X) = 252.50 ...
--- FirstMoment(Y) = 302.50 ...
--- Tower Centroid(X) =   4.59 ...
--- Tower Centroid(Y) =   5.50 ...
--- ----------------------------------- ...
--- Tower of blocks is stable ...
---
--- Add Block No: 12 ...
--- ======================================== ...
--- offset = 3 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   3.00,  11.00 ) ...
---   Corner Point (x2,y2) = (   8.00,  12.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                ...
--- ----------------------------------- ...
--- Total Mass     =  60.00 ...
--- FirstMoment(X) = 280.00 ...
--- FirstMoment(Y) = 360.00 ...
--- Tower Centroid(X) =   4.67 ...
--- Tower Centroid(Y) =   6.00 ...
--- ----------------------------------- ...
--- Tower of blocks is stable ...
---
--- Add Block No: 13 ...
--- ======================================== ...
--- offset = 4 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   4.00,  12.00 ) ...
---   Corner Point (x2,y2) = (   9.00,  13.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                ...
--- ----------------------------------- ...
--- Total Mass     =  65.00 ...
--- FirstMoment(X) = 312.50 ...
```

```
--- FirstMoment(Y) = 422.50 ...
--- Tower Centroid(X) =   4.81 ...
--- Tower Centroid(Y) =   6.50 ...
--- ------------------------------------ ...
--- Tower of blocks is stable ...
---
--- Add Block No: 14 ...
--- ======================================== ...
--- offset = 5 ...
--- Block:  ...
--- Density: 1.000000 ...
--- ------------------------------------------------
---   Corner Point (x1,y1) = (   5.00,  13.00 ) ...
---   Corner Point (x2,y2) = (  10.00,  14.00 ) ...
--- ------------------------------------------------

--- Block Tower Analytics                ...
--- ------------------------------------ ...
--- Total Mass     =  70.00 ...
--- FirstMoment(X) = 350.00 ...
--- FirstMoment(Y) = 490.00 ...
--- Tower Centroid(X) =   5.00 ...
--- Tower Centroid(Y) =   7.00 ...
--- ------------------------------------ ...

--- Crash!!
--- Tower of 14 blocks is unstable

--- ====================================== ...
--- Finished TestObjectBlockTower01.main() ...
```