CEGE0096: Geospatial Programming

Lecture 5: Geometry with Python

October 28, 2019

Aldo Lipani

Review Exercise 7

Design an object-oriented program for the following classes:
`Geometry`, `Point`, `Line`, `Polygon`, `Triangle` and `Square`.

This program should allow the construction of:

- A point;
- A line given two points;
- A polygon given 3 or more points;
- A triangle given 3 points;
- A square given its bottom left point and its side length.

These classes should implement the methods to get or compute (where and when appropriate): the name, the x, the y, the lines, and the area.

TIP: use the Shoelace Formula to compute the are of a triangle given 3 points.

Solution

```
class Geometry:

    def __init__(self, name):
        self.__name = name

    def get_name(self):
        return self.__name


class Point(Geometry):

    def __init__(self, name, x, y):
        super().__init__(name)
        self.__x = x
        self.__y = y

    def get_x(self):
        return self.__x

    def get_y(self):
        return self.__y
```

Solution (II)

```
class Line(Geometry):

    def __init__(self, name, point_1, point_2):
        super().__init__(name)
        self.__point_1 = point_1
        self.__point_2 = point_2


class Polygon(Geometry):

    def __init__(self, name, points):
        super().__init__(name)
        self.__points = points

    def get_points(self):
        return self.__points

    def lines(self):
        res = []
        points = self.get_points()
        point_a = points[0]
        for point_b in points[1:]:
            res.append(Line(point_a.get_name() + "-" + point_b.get_name(), point_a, point_b))
            point_a = point_b
        res.append(Line(point_a.get_name() + "-" + points[0].get_name(), point_a, points[0]))
        return res
```

Solution (III)

```
class Triangle(Polygon):

    def __init__(self, name, point_1, point_2, point_3):
        super().__init__(name, [point_1, point_2, point_3])

    def area(self):
        res = 0
        ps = self.get_points()
        # Shoelace Formula
        res = res + ps[1].get_x() * ps[2].get_y() - ps[2].get_x() * ps[1].get_y()
        res = res - ps[0].get_x() * ps[2].get_y() + ps[2].get_x() * ps[0].get_y()
        res = res + ps[0].get_x() * ps[1].get_y() - ps[1].get_x() * ps[0].get_y()
        res = abs(res) / 2
        return res
```

Solution (IV)

```
class Square(Polygon):

    def __init__(self, name, bl_point, side):
        points = [bl_point,
                  Point("top left point", bl_point.get_x(), bl_point.get_y() + side),
                  Point("top right point", bl_point.get_x() + side, bl_point.get_y() + side),
                  Point("bottom right point", bl_point.get_x() + side, bl_point.get_y())]
        super().__init__(name, points)
        self.__side = side

    def area(self):
        return self.__side ** 2
```

Solution (V)

```
point_a = Point("A", 0, 0)
point_b = Point("B", 0, 1)
point_c = Point("C", 1, 1)

triangle = Triangle("A'", point_a, point_b, point_c)

print("Lines of the triangle " + triangle.get_name() + ":")
for line in triangle.lines():
    print(line.get_name())

print("Area:", triangle.area())
```

Review Exercise 8

Write a program that asks the user to `input` a temperature in degree Celsius and `print`s out a suitable message according to the temperature state below:

- "Freezing" when the temperature is below t_1;
- "Very cold" when it's between t_1 and t_2 (not included);
- "Cold" when it's between t_2 and t_3 (not included);
- "Normal" when it's between t_3 and t_4 (not included);
- "Hot" when it's between t_4 and t_5 (not included);
- "Very hot" when it's equal or above t_5.

The temperatures t_1, t_2, t_3, t_4 and t_5 are **stored in a file named `temps.csv` like this:**

```
0
10
20
30
40
```

Solution

```
print("Temperature to Perceived Temperature Converter (0.4)")

perceived_temps = ["freezing", "very cold", "cold", "normal", "hot", "very hot"]
temps = []
with open("temps.csv", "r") as f:
    for line in f.readlines():
        temp = float(line)
        temps.append(temp)

temp = float(input("What temperature is outside? "))
perceived_temp = perceived_temps[-1]
for i in range(len(temps)):
    if temp < temps[i]:
        perceived_temp = perceived_temps[i]
        break

print("Your temperature today is", perceived_temp)
```

WRITING A PROGRAM

Modeling and Representation

To write a program, first write down a short description of what you want the program to do.

For your coursework, this is more or less the text of the assignment.

The next step is to ask yourself the following questions:

- Who is going to use this program?
- How many times will it be used?
- Will it be extended in the future?

Input Data

What data do I have access to?

What file format is it in?

What is the best way for my program to access the data?

- Hard coding;
- From a file;
- From a database;
- From the web;
- Using a Graphical Unit Interface (GUI);
- Keyboard input.
- Etc.

Storing the Data in the Program

When do I want to read the data?

- As required?
- At the start of the program?

Having read the data, what is the best way to represent it within my program?

- String;
- List;
- Set;
- Dictionary;
- Custom Data Structure.

Output Data

What is the best way to report my output?

- To screen
  - In the command prompt
  - In a GUI
- To a file
- To a database
- To a webpage
- To an email
- Etc.

Developing the Program in Pseudocode

Rewrite the problem statement as a logical sequence:

```
# Read price data from file into a list

# Read the first item of the list

# If price of item is greater than 100 pounds then add item to
list of expensive items

# Repeat with the next item on the list

# Convert the list of items into a string with a newline after
each item

# Save the string to a file
```

Review the Pseudocode

Review the logical sequence:

- Identify flow control:
  - While or for best?
  - If/elif/else?
  - Define a function?
  - Define a class?
- Identify objects, literals and variables
- Identify expressions, operators, functions and methods

Review the Pseudocode (II)

Re-write the logical sequence:

```
# READ price data from file into a list_of_items

# FOR each_item IN the list_of_items

#     split each_item string and assign parts to item_name and item_price

#     IF price of each_item > 100

#         THEN add each_item to list_of_expensive_items

# string_of_expensive_items = list_of_expensive_items converted to string
with new line

# SAVE string_of_expensive_items to a file
```

Rewrite Pseudocode into Python

Re-write the logical sequence:

```
# READ price data from file into a list_of_items

with open("data.csv") as f:
    list_of_items = f.readlines()

list_of_expensive_items = []

# If price of item is greater than 100 pounds then add item to list of expensive
items
for each_item in list_of_items:
    item_name, item_price = each_item.split(",")
    if float(item_price) > 100:
        list_of_expensive_items .append(item_name.strip())

# Convert the list of items into a string with a newline after each item
string_of_expensive_items = "\n".join(list_of_expensive_items)

# SAVE string_of_expensive_items to a file
with open("expensive_items.csv", "w") as f:
    f.write(string_of_expensive_items)
```

Commenting

A well-written program makes clear to the reader what is happening at each step.

Helpful for:

- Other programmers;
- You when returning to the code later.

Comments can either be inline on their own line.

Use your Pseudocode as the basis of your comments.

But do not comment the obvious:

```
some_test = "example"  # setting some_text

# print string some_text
print(some_test)
```

Multiple Files Program

When the program becomes complex you can divide your program into multiple files. To use functions or classes defined in other files you can use the keyword `import`:

```
import foo
foo.bar()
```

Or you can import only a part of it:

```
from foo import bar
bar()
```

You can also give aliases to these imports:

```
import foo as f
f.bar()
from foo import bar as b
b()
```

To successfully import code from another file you need to fulfil the basic program planning: have a main function.

What happens if you don't do it?

Basic Program Planning (II)

It is standard practice to turn all code into a main function:

- Ensures that there are no stray global variables;
- Futureproofs your code should you want to use it into another script;

When `import`-ing a file, this file gets parsed and execute.


matplotlib

Matplotlib

Matplotlib is one of the most commonly used data visualization packages in Python.

This is commonly imported as `plt`.

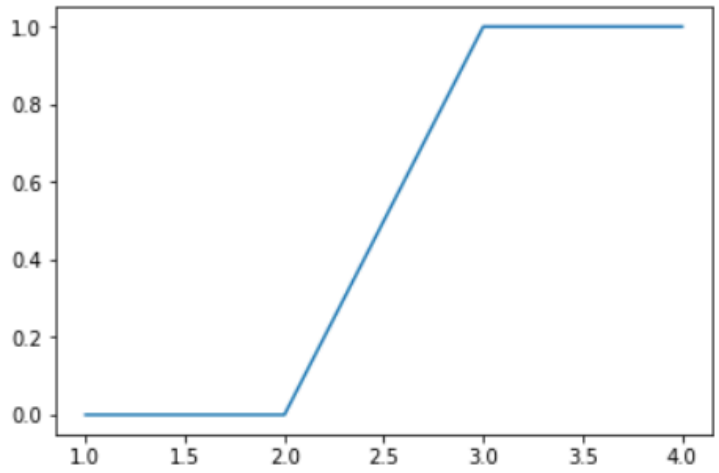
```
import matplotlib.pyplot as plt
```

Matplotlib in Jupyter Notebooks

Jupyter notebook has built in matplotlib support showing plots in the notebook.

```
1  plt.plot([1, 2, 3, 4], [0, 0, 1, 1])
```

```
[<matplotlib.lines.Line2D at 0x7f1aa7683a58>]
```



Matplotlib in Standalone Programs

When developing standalone programs you also need to call the show method.

```
import matplotlib.pyplot as plt

if __name__ == "__main__":
    plt.plot([1, 2, 3, 4], [0, 0, 1, 1])
    plt.show()
```

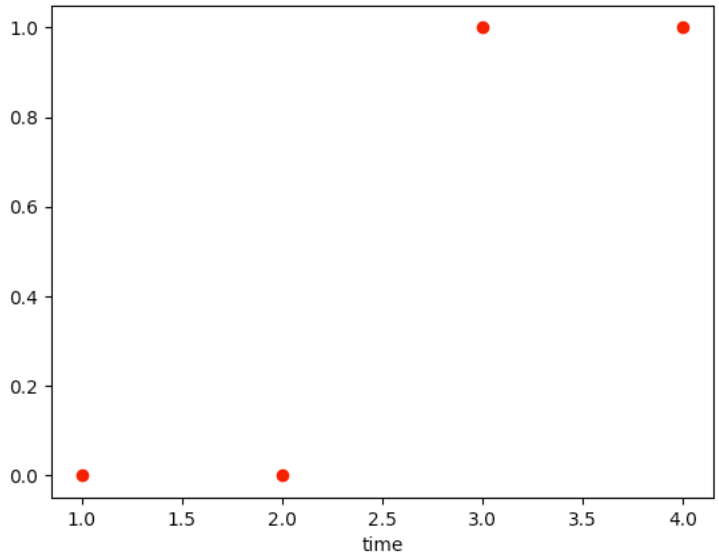
`show` opens a window and pauses the program until the window is closed.

Formatting the Style

With a third argument you can indicate the colour and the line type of the plot:

```
plt.plot([1, 2, 3, 4], [0, 0, 1, 1], 'ro’)
```

r stands for red and o for point.



To label the axis you can use:

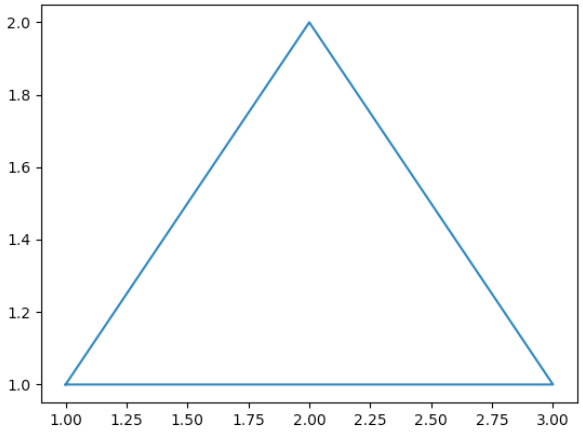
```
plt.xlabel("time")
```

Here you find a complete list of format strings:
https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot

Drawing Shapes

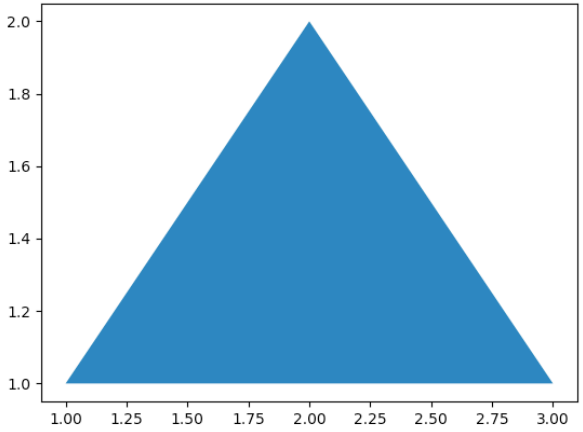
Based on the input of the function you can draw shapes:

```
plt.plot([1, 3, 2, 1], [1, 1, 2, 1])
```



To plot filled shapes you can use the method fill:

```
plt.fill([1, 3, 2, 1], [1, 1, 2, 1])
```



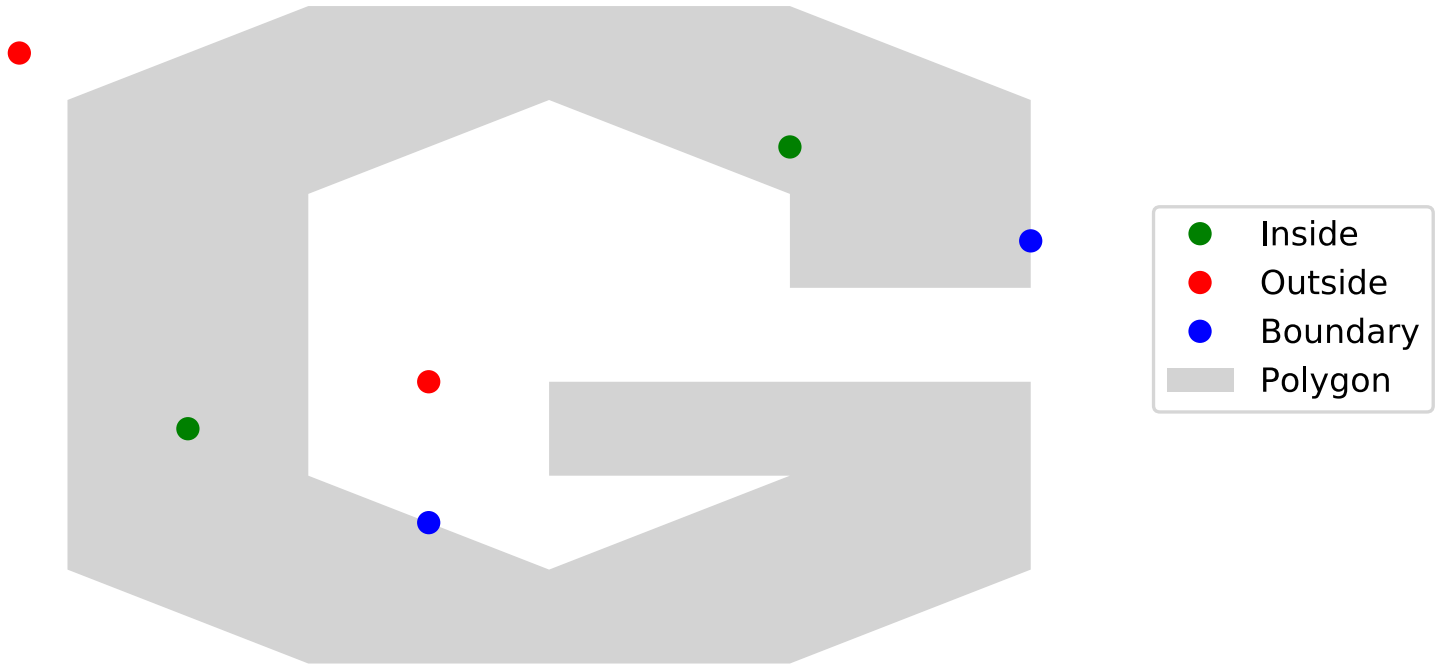
Documentation and Tutorials

For documentation and tutorials about matplotlib visit:

- https://matplotlib.org/index.html
- https://matplotlib.org/tutorials/index.html

POINT-IN-POLYGON TEST

Point-in-Polygon Test



Point-in-Polygon Test

You need to build on the Geometry, Point, Line, and Polygon classes that you have created.

Your task is to write two Python programs:

`main_from_file.py` and `main_from_user.py`

`main_from_file.py` should:

1. **Read** a list of coordinates from a CSV file and create a polygon;
2. **Read** a list of coordinates from a file and create a list of testing points;
3. **Categorize** these points into: “inside”, “outside” and “boundary”;
4. **Output** the result of each point in a CSV file;
5. **Plot** the points and polygon in a plot window.

Point-in-Polygon Test (II)

You need to build on the Geometry, Point, Line, and Polygon classes that you have created.

Your task is to write two Python programs:

`main_from_file.py` and `main_from_user.py`

`main_from_user.py` should:

1. **Read** a list of coordinates from a CSV file and create a polygon;
2. **Read** a list of coordinates from shell;
3. **Categorize** this point into “inside”, “outside” and “boundary”;
4. **Plot** the point and polygon in a plot window.

Input

Reading the CSV file. Create a function or method that takes a CSV file and creates a list of points. This should may be reused also to read the list of testing points.

You can get more marks if you can pass the file path as an argument to the main function.

Input a Point for testing. Ask the user to input the coordinates from the keyboard.

You can get more marks if you can pass a tuple pairs as an argument to a main function.

Output

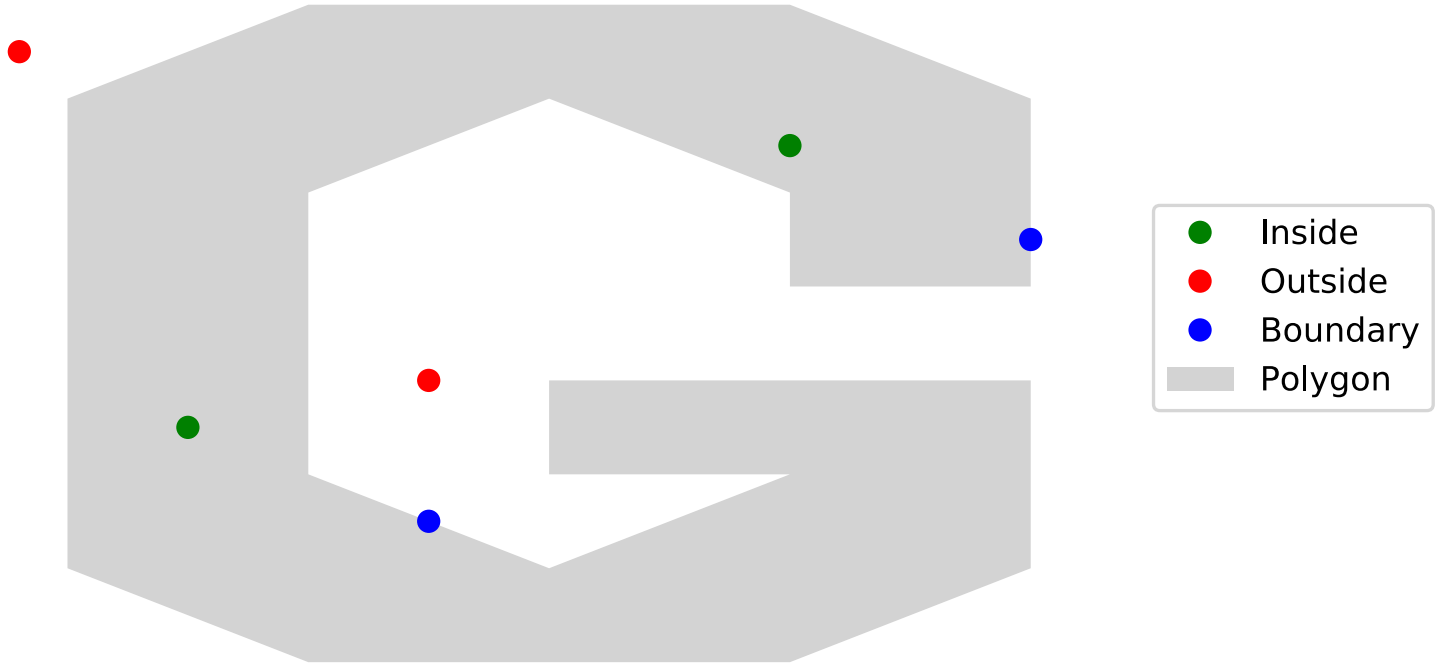
Writing the CSV file. Create a function or method that takes a file name and creates a CSV containing the id of each point and its categorization.

You can get more marks if you can pass the output file name as an argument to the main function.

A sample CSV called '`sample_output.csv`' is provided.

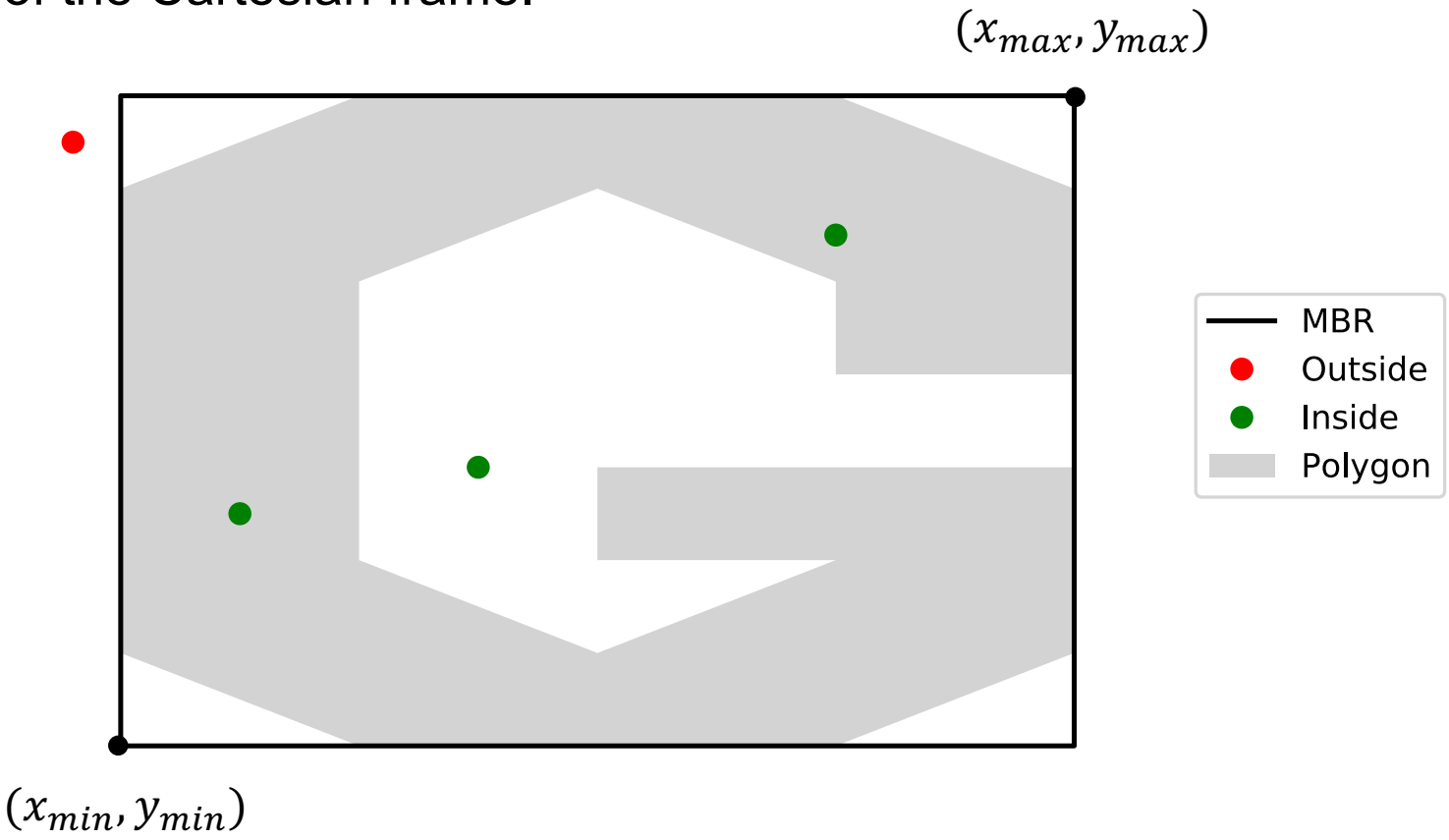
Test Whether a Point is in the Polygon

Given a point and a polygon, the point-in-polygon problem involves determining whether the point lies inside, outside or on the boundary of the polygon.



Minimum Bounding Rectangle

The MBR is defined as the smallest rectangle with sides parallel to the axes of the Cartesian frame.



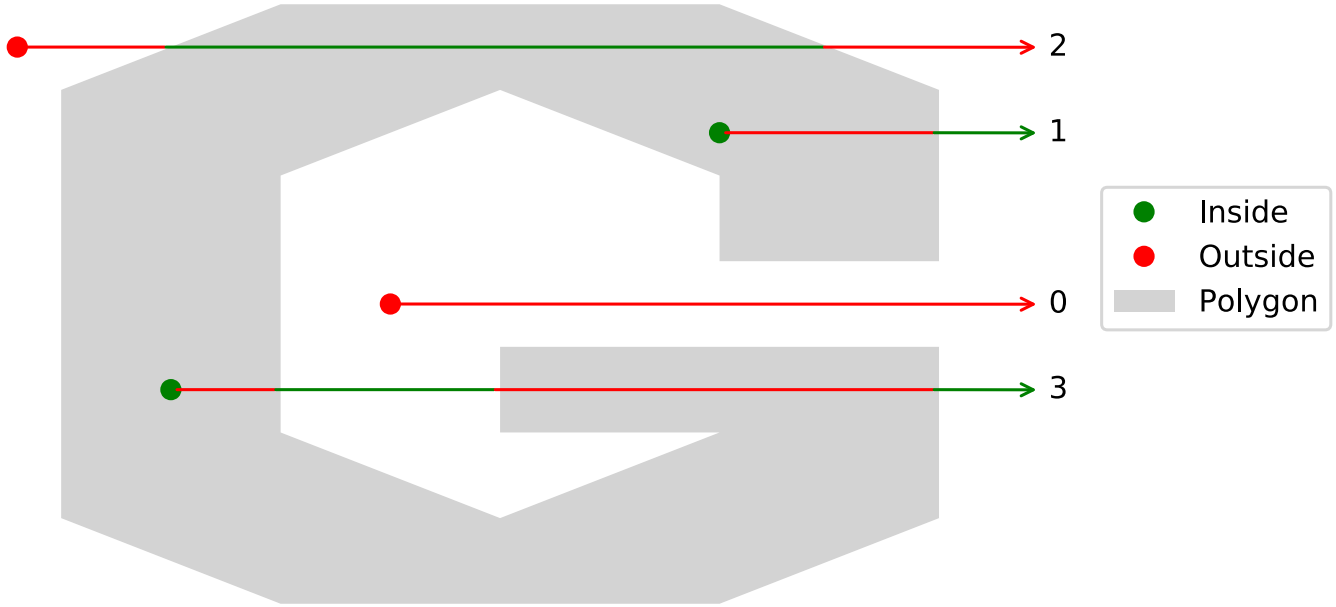
Minimum Bounding Rectangle (II)

If a point is outside the MBR than the point is certainly outside the polygon, otherwise we do not know yet: A more sophisticated algorithm is needed to determine it.

```
# IF x_min < point.x < x_max AND y_min < point.y < y_max:
#     THEN point is inside the MBR but we don't know about the polygon
# ELSE:
#     THEN point is outside the MBR
```

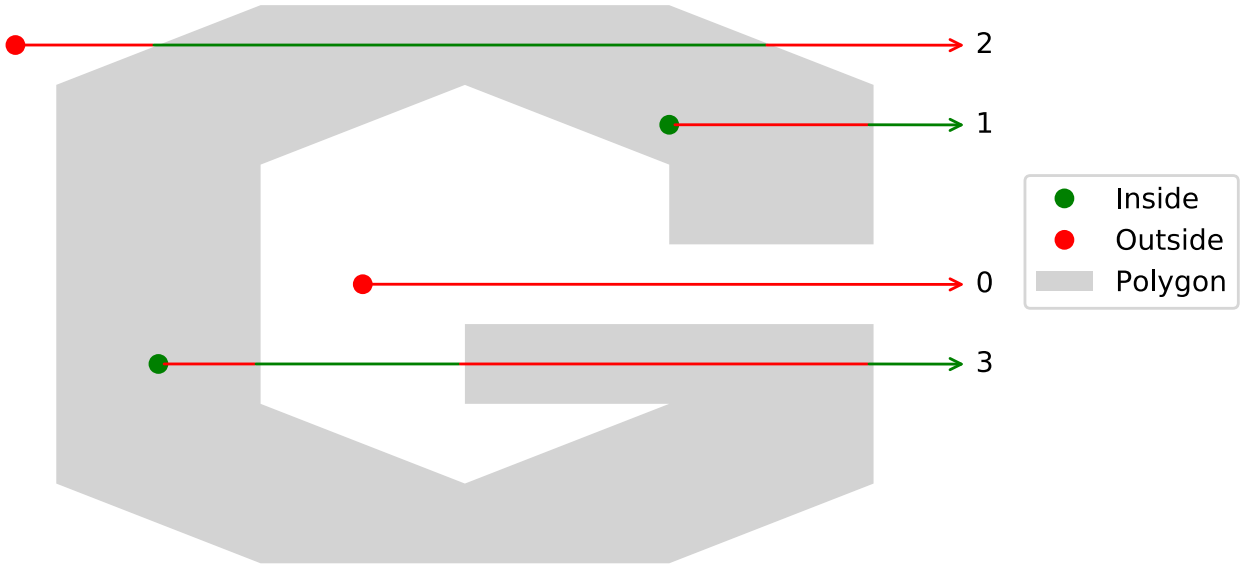
Point-in-Polygon

If the point is inside the MBR then a Point-in-Polygon (PiP) algorithm is needed. The **ray casting algorithm** (RCA) is an example of a PiP algorithm.



Ray Casting Algorithm

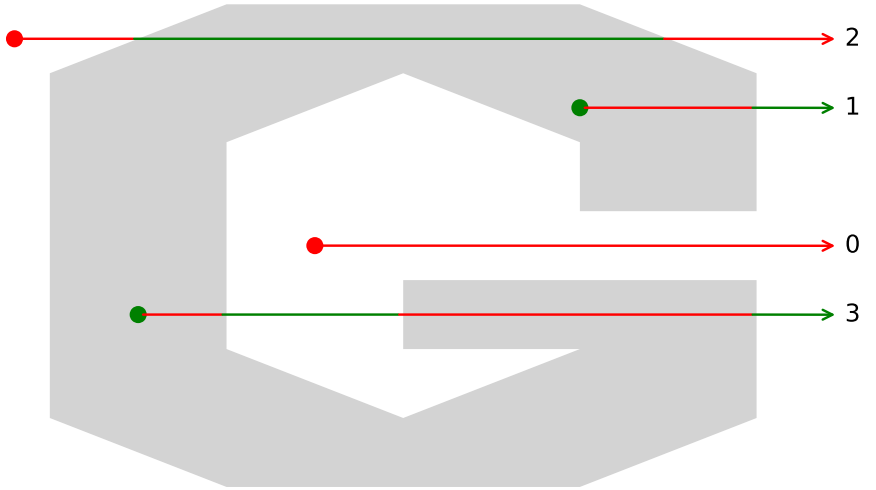
Draw a line from the point to “infinity” (in any direction) then count the number of times the line crosses polygon boundary. If the number of crossings is **even**, then the point is **outside** the polygon, else if the number of crossings is **odd** the point is **inside** the polygon.



Ray Casting Algorithm (II)

Pseudo code of the RCA algorithm:

```
# SET counting to 0
# FOR each line of polygon:
#     IF ray crosses the line:
#         THEN INC counting by 1
# IF counting is odd:
#     THEN the point is inside
# ELSE
#     THEN the point is outside
```



Ray Casting Algorithm (II)

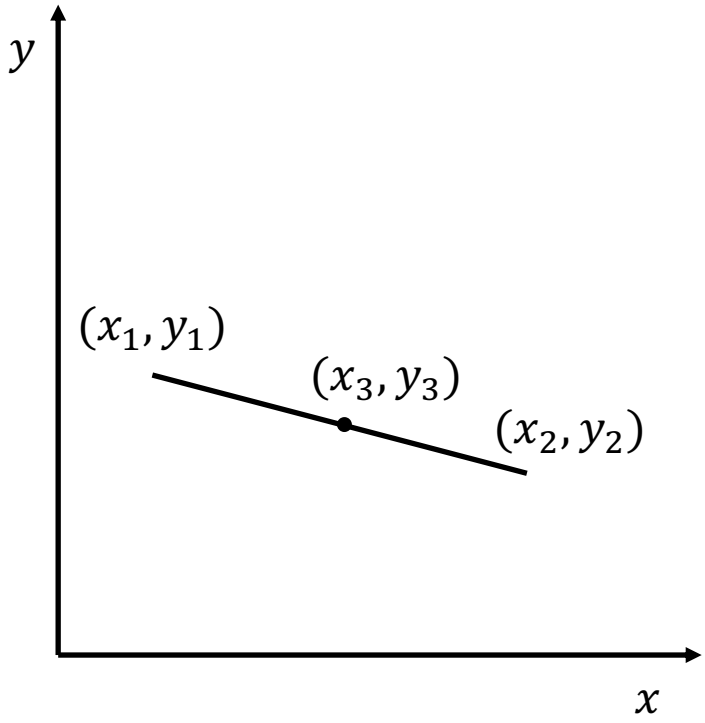
What point at “infinity” should I use?

- Any point outside the MBR would work;
- However, if you assume that the point is at infinity, the math simplifies, so actually you don’t need to define such a point.

We will now review some checks you need to solve this assignment:

- Point on a line;
- Line crossing;
- Line crossing with a parallel line to the x-axis;
  - Special Case for RCA: use this if your rays are parallels to the x-axis.

Point on a Line Algorithm



First check if y_3 is within y_1 and y_2 included.
If not the point is not on the line.

If the line is **parallel to the y-axis** and x_3 is equal to x_1 or x_2 then the point is on the line, else the point is not on the line.

If the **line is not parallel to the y-axis**, recall a line passing between two points:

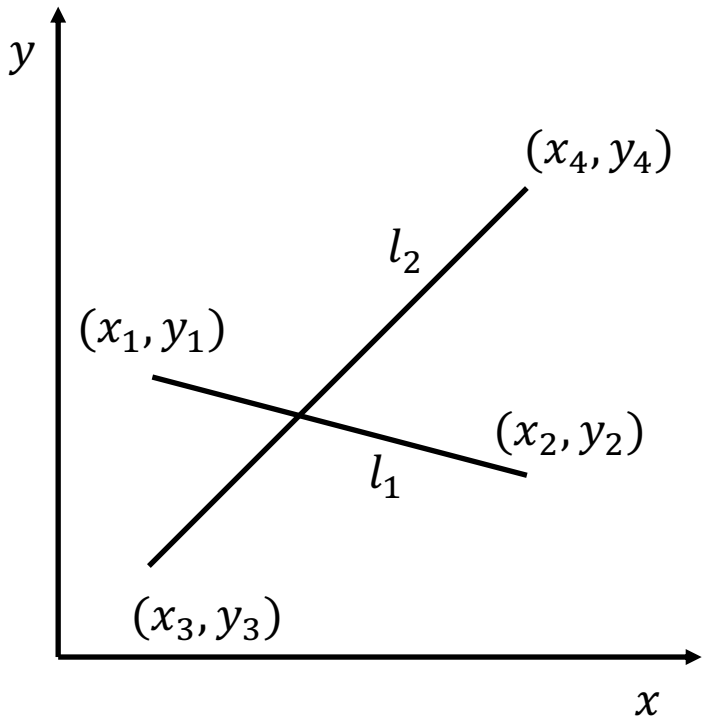
$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1}$$

Solving for y:

$$y = \frac{x - x_1}{x_2 - x_1}(y_2 - y_1) + y_1$$

If when substituting $\boldsymbol{x_3}$ **to** $\boldsymbol{x}$ **and get** $\boldsymbol{y_3}$ **as** $\boldsymbol{y}$ then the point is on the line.

Line Crossing Algorithm



Let's first extend the equation of a line passing by two points:

$$y_{l_1} = \begin{cases} y_1, & x_2 - x_1 = 0 \\ \frac{x - x_1}{x_2 - x_1}(y_2 - y_1) + y_1, & otherwise \end{cases}$$

$$y_{l_2} = \begin{cases} y_3, & x_4 - x_3 = 0 \\ \frac{x - x_3}{x_4 - x_3}(y_4 - y_3) + y_3, & otherwise \end{cases}$$

To find the intersecting point we solve the following for x:

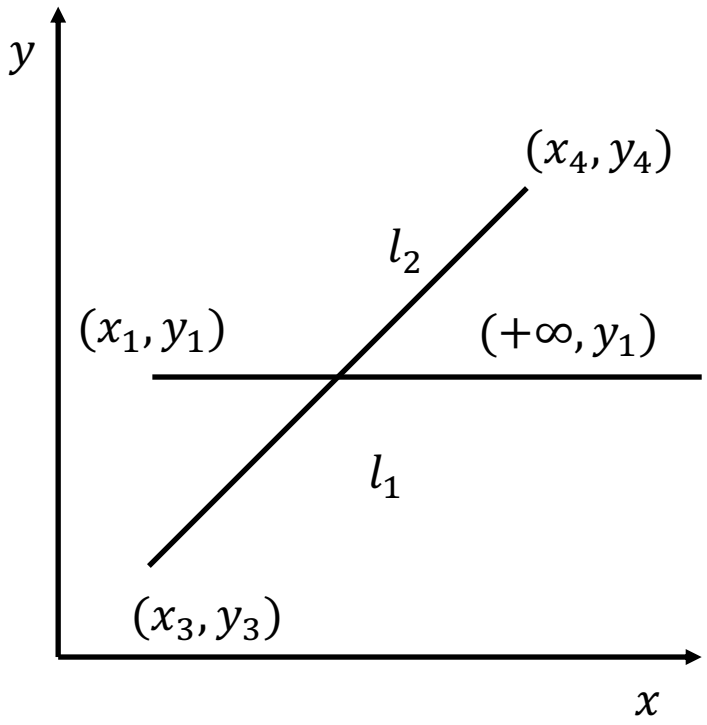
$$y_{l_1} = y_{l_2}$$

We have 4 cases:

- If they are both **parallel with the y-axis**:
  - If $\boldsymbol{x_1 = x_3}$ then the problem reduces to checking if the y coordinates intersect; If they do, then the lines are crossing.
  - If not then the lines are not crossing.
- In the other 3 cases:
  - If the $\boldsymbol{x}$ **exists and it is between the x coordinates of one line** then the lines are crossing.
  - If not the the lines are not crossing.

What happens when the lines are parallel with each other?

Line Crossing Algorithm (Special Case)



Our problem simplifies to:

$$y_{l_1} = y_1$$

$$y_{l_2} = \begin{cases} y_3, & x_4 - x_3 = 0 \\ \frac{x - x_3}{x_4 - x_3}(y_4 - y_3) + y_3, & otherwise \end{cases}$$

To find the intersecting point we solve the following for x:

$$y_{l_1} = y_{l_2}$$

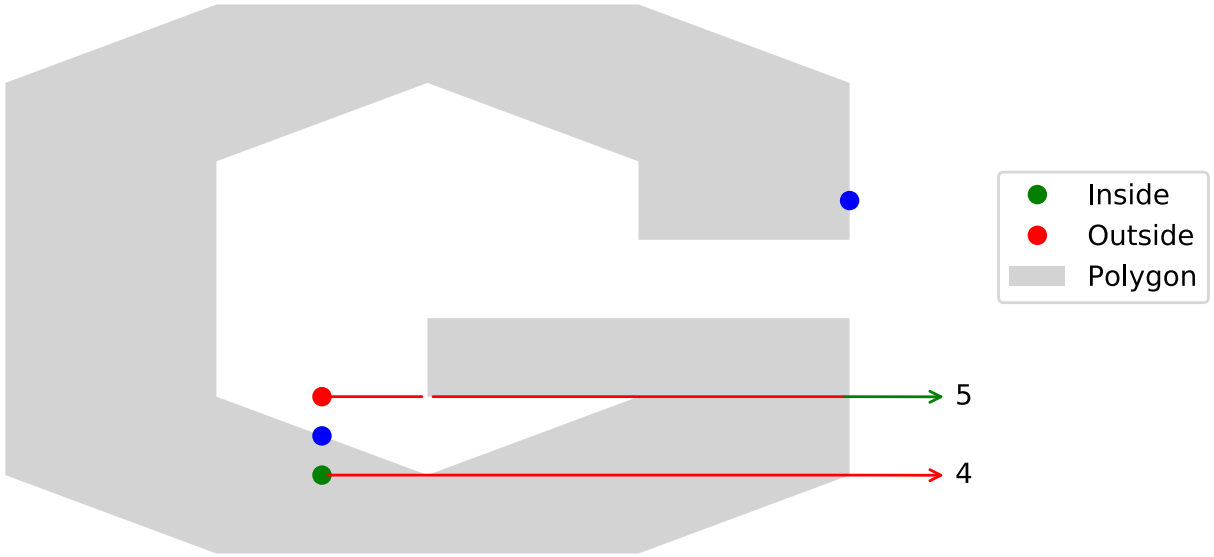
Now we have 2 cases:

- If $\boldsymbol{l_2}$ **is also parallel** to the x-axis:
  - If $\boldsymbol{y_1 = y_3}$ then the problem reduces to checking if $\boldsymbol{x_1}$ **is less or equal than** $\boldsymbol{x_3}$ **or** $\boldsymbol{x_4}$; If it is, then the lines are crossing on an infinite number of points.
  - If not then the lines are not crossing.
- In the other case:
  - If $\boldsymbol{y_1}$ **is between** $\boldsymbol{y_3}$ **and** $\boldsymbol{y_4}$ the problem reduces to checking if $\boldsymbol{x}$ **is greater or equal than** $\boldsymbol{x_1}$; If it is, then the lines are crossing on one point.
  - If not then the lines are not crossing.

RCA Special Cases

Boundary points: Easy, just check if a point lie on a line of the polygon.

Points crossing vertices of the polygon: More difficult, it requires some additional considerations.

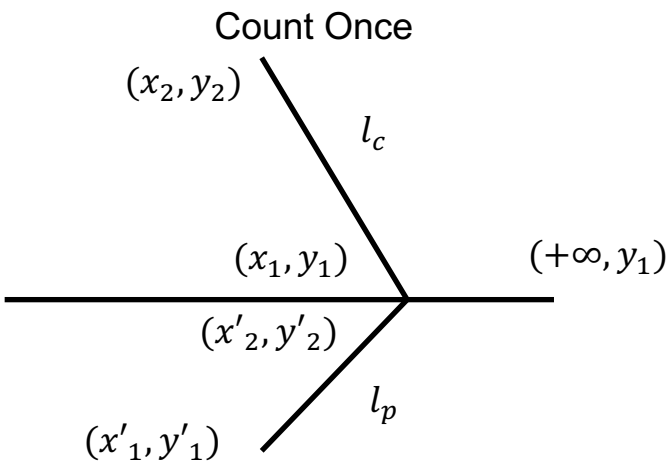


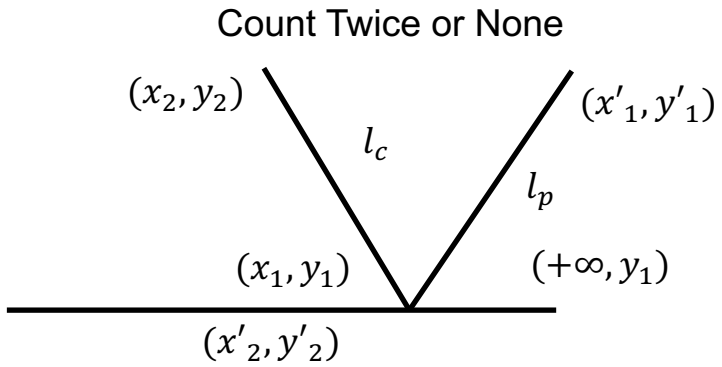
Point Crossing Vertices

First, make sure that you are visiting each line in order (clock-wise or counter clock-wise). This may be a feature of the way you define your polygon.

When iterating over the for loop we need to keep track of two lines: the current one (l_c) and the previous one (l_p).

If the non vertex points of the two lines are both above the ray, then you should count this crossing only ones, otherwise you can count them twice or none.





You will soon discover that this solution solves only one of the two cases, the bottom ray. To solve the top ray you need to think about the parallel lines with the x-axis.

Provided Material

Data (provided together with the project template):

- 1 CSV file containing the coordinates of a polygon;
- 1 CSV file of testing point;
- 1 CSV file containing a sample output.

Project template:

- Predefined main functions;
- Plotter class already defined.

Project report:

- https://docs.google.com/document/d/1hwSchzGKcCURs5tO8r3OFI5h_FokUDd6y9ryttIZfHk

Submission

This assignment should be submitted as follows:

- **a zip file** containing the project solution to the Assessment tab of the module Moodle page;
- **a pdf file** of the project report to the Assessment tab of the module Moodle page;
- **as a GitHub repository** by inviting me as a collaborator (aldolipani).

Failing to carefully follow these instructions may result in penalties.

Note that an element of collaborative work is allowed in this project but please do not submit scripts that are identical to one another. In this case your work will not be accepted.

Marking Scheme

n.	Task Description	Marks
1	Successfully Implement MBR.	15
2	Successfully Implement RCA.	15
3	Successfully categosization special cases.	10
4	Write a report about the project.	10
5	Make your code Object-oriented.	10
6	Make regular commits on the GitHub repository.	10
7	Comment your code clearly so that it can be understood by others.	5
8	Apply PEP8 style.	5
9	Plotting of points, polygons, and ray with appropriate axis labels and annotations.	5
10	Incorporate some simple error handling functionality.	5
11	Creativity marks are available under certain conditions for adding features that have not been specified.*	10

See you on Thursday for the Practical

Christopher Ingold Building G20

Do not forget to take your personal laptop with you.