

MA1008 Introduction to Computational Thinking

Mini Project: Set Algebra and Shape Algebra

Semester 1, AY 2025/2026, Week 10 – Week 13

1. Introduction

The objective of the mini project is for you to produce a program of a moderate size and depth that will require you to utilise what you have learned in this course, and a bit more, to do something useful and interesting. Through this, you will learn to design, manage and execute a sizable program.

2. The Project

There are three operators in set algebra: Union (\cup), difference ($-$) and intersection (\cap). Their effects are shown using Venn diagrams in Figure 1:

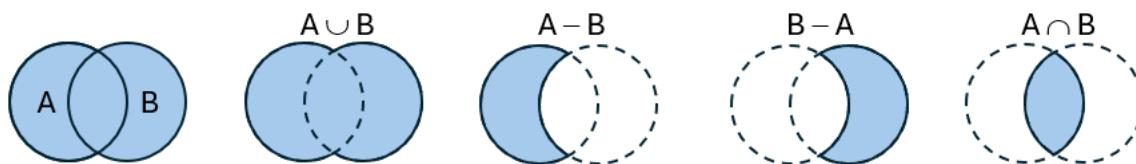


Figure 1: Venn Diagrams. The union and intersection operators are both commutative. The difference operator is not commutative, hence $A - B$ and $B - A$ are different.

The Venn diagram is apt for illustrating the purpose of this project – shape algebra – where we look to form complex shapes from simpler ones using the set operators. You can imagine the two Venn circles as some other shapes, and their combination via a set operation produces a new shape, as illustrated using two polygons in Figure 2. This new shape is simply another polygon.

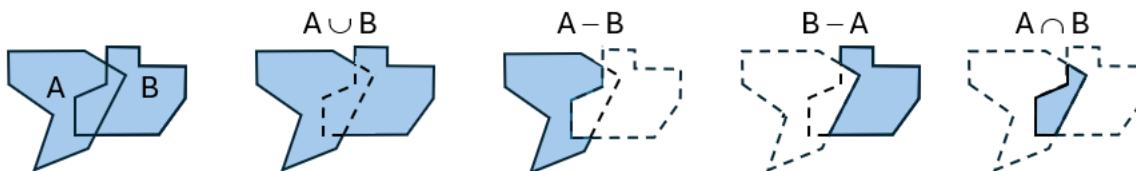


Figure 2: Venn Diagrams for two overlapping polygons. The dotted lines are there to show the previously existing edges. They do not form part of the result (shaded area).

The result of the combination could be further operated on with another polygon, as in Figure 3.

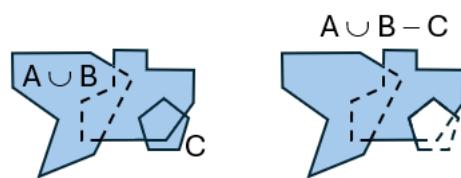


Figure 3: Operation with a third polygon

This process can carry on using as many polygons and operators as necessary to form the required shape. Furthermore, each polygon may not be a lone polygon, but a polygon formed by previous operations. Therefore, one could possibly build a set expression of the form

$A \text{ <op>} (B \text{ <op>} C) \text{ <op>} (D \text{ <op>} E \text{ <op>} F) \dots$

where <op> is one of the three set operators, to form complex shapes, with each bracketed term forming one new polygon before being combined with the others.

This project requires you to write a Python program that will (1) define individual polygons and (2) combine them using set operators to form new polygons.

2.1 Your tasks

First Task: write a Python program with the following capabilities:

1. Define and display polygons
2. Select two polygons you have defined and a set operator, and perform the set operation
3. Display the result

Appendix A provides you with the algorithm for combining two polygons. Appendix B provides the mathematics required.

Second Task: Once you can do the above, proceed to operate on the resulting polygon with another polygon, to form a more complex shape. Extend the program to allow the user to combine as many polygons as desired. The second polygon (and third, and fourth, ...) could be the result of another set operation.

2.2 Actions you or your program need to take

2.2.1. Define a polygon

You need to be able to create a polygon in Python, which is defined by a list of vertices. This can be done in two ways: the first is to create it interactively, and the other is to read it from a file. Your program needs to provide both capabilities.

Interactive input means you enter the vertices of a polygon when running your program. Therefore, you need to provide the (x, y) coordinates of every vertex in your polygon. There are two ways to do this: (a) by typing in the coordinate values one by one or (b) by using the mouse to click at a point on the screen and register the cursor coordinates. You are required to provide Method (a). Method (b) is optional and earns you bonus marks.

For file input, you need to have your polygon coordinates stored in a text file. Your program simply reads the data from the file and store them in a list in the program. There are two ways to create a file. The first is to create it using a text editor and type in the coordinates. The second is to output the data you have created interactively to a file. You need to specify the format of your data file and make sure that both methods adhere to it. File input is a requirement.

2.2.2 Identify a polygon you have specified and define the set operation

Say you have two polygons defined, each stored as a list in your system. You need to be able to identify each one when running the program, either by a name or a number. Let's say you identify them by names A and B. Then you need to specify the operation required, either $A \cup B$, $A - B$, $B - A$, or $A \cap B$. You also need to give the resulting polygon a name, so that you can use it in a later operation.

Of course, you could have more polygons defined and name them suitably.

Your program should perform the operation you have specified with polygons you have created.

2.2.3 Display your input polygons and the result

You need to display the input polygons and the result, suitably scaled and positioned in your display window. For that, you need the turtle graphics library, which is discussed further in Section 2.5.

2.2.4 Exclusion

A set operation on shapes can produce disjoint polygons, such as after $A - B$ in Figure 4a or $A \cap B$ in Figure 4b. Such results require some special handling and are excluded from this project. However, if you do handle them, you can earn bonus marks.

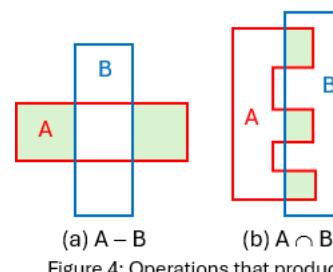


Figure 4: Operations that produce disjoint polygons (shaded areas)

2.3 Program Design

- a. A program needs to be designed before you start any coding. Work out first how you would store your data and the form of the dialogue/interaction between the program and the user.
- b. Most operations should be coded as functions. Your main program should be fairly short, consisting mainly of calls to these functions.
- c. Where a set of statements is repeated in different places in the program, consider putting it into a function, and call the function where the repetition occurs.
- d. When a repetition occurs but with different variables and outputs, then the variables probably should be parameters to your function, and the outputs should be the returned values.

2.4 Computations

You will need the algorithm for performing the set operations. This is given in Appendix A. The operations involve intersection of straight lines in 2D, which is a mathematical operation. This is given in Appendix B.

2.5 Graphics

The polygons are to be displayed in a graphics window. This is done using the graphics library called Turtle. Turtle capabilities and functions are described in the Python documentation, which you can access via the IDLE interactive shell by clicking [Help > Python Docs](#) and then search for “turtle”.

In doing the drawing, you need to consider a few basic things.

- a. Your display window is fixed, in terms of the number of pixels in the x and y direction. (Turtle gives you a default window size if you do not specify it yourself.) Your polygon size is variable. Hence, you should handle the size of the window carefully so that the polygons appear nicely in the window.
- b. Turtle provides you with many functions for doing display, but you only need to use a few of them for the purpose of this project, mainly for posting text, drawing straight lines and dots, and selecting colours. So, determine the graphics functions you need before reading the Turtle documentation. That would save you some time.
- c. By default, Turtle draws slowly to allow us to see how the drawing is done. However, for this project, you should draw quickly. So make sure that you set the drawing speed to the fastest possible, again doable by calling the appropriate Turtle function with the appropriate parameter value. You should hide the turtle too.

2.6 Program Flow and Control

- a. When running the program, it should be clear at every stage what the user needs to do.
- b. Therefore, you need to ensure that the information presented on the screen, such as prompts for user inputs, is clear and unambiguous.
- c. When asking the user to make a selection, always require the minimal input from the user to reduce burden and error.
- d. Allow the user to select between ending the program and returning to input for a new start.
- e. Your program should trap and handle errors appropriately. For example, in the case of an input error, your program should tell the user what the error is and allow the user to re-input.
- f. Your program should be easy to read and understand. So, make sure that it is well commented, well modularized and uses meaningful (but not overly long) variable and function names.

3. Prohibitions

Apart from the graphics library, everything you need for this project can be done through what you have learnt in this course. To allow you to exercise what you have learnt and to prevent you from veering wildly beyond the scope of this project, your program should:

- a. Not use the *class* construct to define objects you require in your program
- b. Only use the *Turtle* library for graphics. You must not use *tkinter*, *matplotlib*, *plotly* or similar.
- c. Not use the libraries *numpy*, *pygame* or *json*.
- d. Not use the *lambda* construct or the *eval()* function.

You may use the standard Python libraries like *math* and *datetime*. Beyond that, using any external library requires the approval of your tutor. So, please consult your tutor beforehand.

4. Conduct of the Project and the Submission

4.1 Schedule

The duration of this project is from the Monday of Week 10 to the Sunday of Week 13, spanning four weeks. The class hours in Weeks 11-13 are dedicated to the project, apart from CA4 in Week 11 for one hour. Your tutors will be in the class to offer you help. Make full use of them.

The project deadline is at **23:59 hours, Sunday 16 November 2025**.

4.2 Project Log

You are required to keep a project log in a Word or Excel file, listing the things you have done for the project day by day. You are required to submit the file weekly at the end of each of the four weeks, together with the Python program you have written up to that time. This program does not need to be complete or fully working, it merely serves as a record of your progress. This log should be cumulative, which means one week's log should be added to the end of the previous week's log, without changing anything of the previous week. A sample project log is provided separately.

See Section 4.4 on how to submit the log. **The weekly logs and programs show your interim progress in the project and are worth 15% of the project grade.**

In the four project weeks, your tutor may seek to speak to you concerning your progress, your log or your program. So, you should be present during the tutorial hours.

4.3 What you need to submit finally

- i. **A working Python program** that gets the relevant inputs from the user and displays the required information in a graphics window.
- ii. Your weekly log for Week 4.
- iii. **A report in a Word or PDF file**, providing
 - A guide on how to run your program
 - A list of the actions your program handles, including their input requirements.
 - Graphical outputs from your program showing the input polygons and the results. Provide the results, at least one each for the four different set operations with different polygons, and two results using multiple polygons and set operators.
 - Highlights of the **key strengths and limitations** of the program.
 - Highlights of features you consider to be worth bonus marks (see examples in Section 5.)

4.4 How to Submit

Please submit all your files through the course site at the same item where you fetched this project file. The title of the item is a link to the submission page, which has a link for uploading your files. This applies to the submission of the project logs and the interim programs as well as the final submission.

5. Grading Rubrics

Here are what the graders will be looking for when grading:

- i. The ease in interacting with the program, which includes specifying the input data and working with the input files.
- ii. The quality and correctness of your program, which includes:
 - a. How easy it is to read and understand your program. This means your program should be adequately commented with good choice of variable and function names.
 - b. Logical program flow.
 - c. Modularisation of the program including appropriate use of functions, the design of the functions which include the appropriateness of parameters in the functions.
- iii. The quality and correctness of the outputs.

- iv. Bonus marks, up to a maximum of 20, will be awarded for nice and useful features that your program offers. Hence potentially, you can score a maximum of 120 marks. Marks beyond 100 will be added to your CA1 or CA3 marks. Bonus features include
- smooth and easy creation of polygons interactively
 - creation of interesting shapes
 - attractive displays
 - interesting algorithms of your own design to solve some problems, for which you need to explain the algorithm in the report
 - and more ...

Make sure that you highlight the bonus features in your report.

6. Epilogue

You should start working on the program immediately. Do not procrastinate. What you produce depends on the time you spend on the project and your ability in creating good algorithms and writing good code.

One of the major problems of past projects for the tutors is that there is no way to tell how to run a program. The program must offer help to the user directly (using prompts and on-screen instructions) on what input is required at every stage. Quite often, the input is very cumbersome, such as requiring the user to type a long string exactly. You need to help your user by making your input requirements as obvious and as simple as possible.

This is an individual project. You may consult the tutors and discuss with your classmates, but everyone must write their own program. **Any programs deemed to have been done dishonestly will be penalised heavily. Submissions that are copies of each other will be penalised regardless of who did the copying.**

If in doubt or in difficulty, always ask. And ask early!

Appendix A: Algorithm for Combining two Polygons

We first need to define the polygon representation. A polygon is defined by an ordered list of its vertices. Two adjacent vertices are joined by an edge. The edges form a closed loop, which is the polygon. The edges do not intersect each other except at the vertices. Here we only deal with polygons with straight edges. We exclude curved edges as they are more complex mathematically.

There are three set operators for combining two polygons, A and B: Union (\cup), difference ($-$) and intersection (\cap). The combining requires determining which parts of A and B form the outcome.

Two polygons can interact in different ways. Figure A-1 shows five of them.

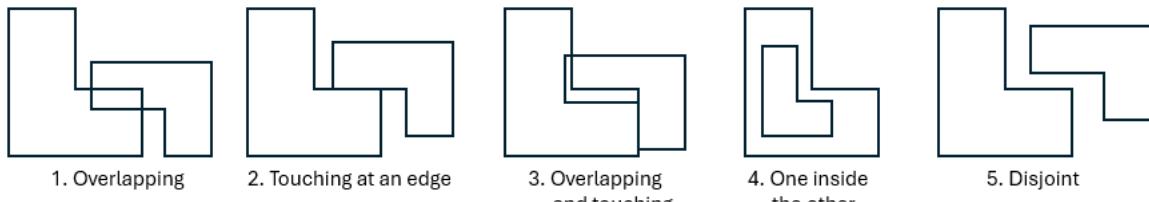


Figure A-1: Five different cases of two polygons interacting

In this project, we will only deal with Case 1 where the two polygons overlap and produce one polygon upon combining. This simplifies matters because the others require some special treatments which add complications.

When two polygons overlap, some of their edges intersect. An intersected edge is divided into two or more segments, each of which may or may not form a part of the boundary of the resulting polygon. A segment that is in the new boundary is retained while one that isn't is discarded. The problem here is therefore to determine which part is to be retained and which not. We also need to determine the status of every edge that has not been split as well. For example, in the Venn diagrams for the different operations in Figure 2, the dashed edges are discarded, while the solid edges are retained.

Algorithm to determine which part of an edge is to be retained

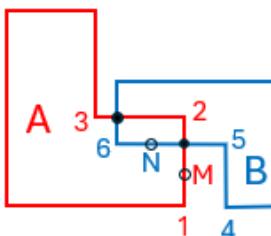


Figure A-2 Two overlapping polygons

In Figure A-2, the red polygon A and the blue polygon B overlap, and their edges intersect at two points, marked by two dots, as shown. Take Edge 1-2 of Polygon A, it is split into two segments. To determine which segment to retain or discard, we take the midpoint of each segment and check if it is inside or outside the other polygon, Polygon B. A little circle marks the mid-point, M, of the lower segment of Edge 1-2, and we can see that it is outside Polygon B. Whether to retain this segment depends on which of the four set operations we are doing: $A \cup B$, $A - B$, $B - A$ or $A \cap B$.

Similarly, we need to consider N, the mid-point of a segment of Edge 5-6 of Polygon B after it has been split (see Figure A-2). Whether it is inside or outside A determines if we are to retain or discard the segment. The table below lists the actions for the four set operations, for both M inside B and M outside B, and also N inside A and N outside A:

	M inside B	M outside B	N inside A	N outside A
$A \cup B$	Discard	Retain	Discard	Retain
$A - B$	Discard	Retain	Retain	Discard
$B - A$	Retain	Discard	Discard	Retain
$A \cap B$	Retain	Discard	Retain	Discard

Table A-1: Action on a segment of a split edge

This action is taken on every segment of every edge in A that has been split. Similarly, the same action needs to be taken on every segment of every edge in B that has been split.

Also, the same action needs to be taken on every edge of both A and B that has not been split. We find the mid-point of every unsplit edge in A and B and use the same table to determine if it is to be retained or discarded.

After the decisions on retaining or discarding the different edges and segments have been made, the retained edges and segments need to be reassembled into a polygon, by placing their vertices in the correct order in a list. This is the resulting polygon.

The major steps in the algorithm can be summarised in pseudo code thus:

```

For each edge E1 in A
  For each edge E2 in B
    Intersect E1 and E2
    If there is intersection
      Store the segments of E1 and E2 split at the intersection point

```

```

For each edge of A and B that are not split and each segment of the edges that are split
  Use Table A-1 to determine if it is to be retained
  If to be retained
    Add it to a new list for the resulting polygon

```

Reassemble the retained edges to form a closed loop for the resulting polygon

Table A-1 requires determining whether a point is inside or outside a polygon. This can simply be done by casting a ray (i.e. a straight line) from the point in any direction to infinity. If this ray intersects the polygon boundary at an odd number of points, the point is inside the polygon. If the number of points is even, the point is outside the polygon.

Need to watch out when the ray crosses a vertex of a polygon, as the number of intersection points there could be 0, 1, or 2. Also, there is complication when the ray coincides with an edge. Both these can be resolved by simply changing slightly the direction of the ray, which can be done since the ray can be in any direction.

Line/line intersection is required at two places in this algorithm: when finding where two polygon boundaries intersect and finding the number of times a ray crosses a polygon. This is a mathematical operation which is given in Appendix B.

Appendix B: Line/line Intersection

The vector equation of a straight line may be written in the parametric form:

$$\mathbf{P} = \mathbf{A} + \mathbf{V}t \quad (1)$$

where $\mathbf{P} = (x, y)$ is a point on the line at parameter t , $-\infty < t < \infty$, $\mathbf{A} = (A_x, A_y)$ is a given point on the line and $\mathbf{V} = (V_x, V_y)$ is the direction of the line. (Note: The straight-line equation $y = mx + c$, the form commonly taught in schools, cannot be used here because it is not able to deal with vertical lines for which m is infinity.)

Let us write the equations of two such straight lines as

$$\mathbf{P}_1 = \mathbf{A}_1 + \mathbf{V}_1 t_1 \quad (2)$$

$$\mathbf{P}_2 = \mathbf{A}_2 + \mathbf{V}_2 t_2 \quad (3)$$

The two lines intersect when $\mathbf{P}_1 = \mathbf{P}_2$. Therefore, at the point of intersection,

$$\mathbf{A}_1 + \mathbf{V}_1 t_1 = \mathbf{A}_2 + \mathbf{V}_2 t_2 \quad (4)$$

This looks like one equation with two unknowns (t_1 and t_2), but it is actually two equations with two unknowns, because it is a vector equation. In component form, the two equations are

$$A_{1x} + V_{1x} t_1 = A_{2x} + V_{2x} t_2$$

$$A_{1y} + V_{1y} t_1 = A_{2y} + V_{2y} t_2$$

These are two equations with two unknowns t_1 and t_2 , therefore the solutions are

$$t_1 = ((A_{2x}V_{2y} - A_{2y}V_{2x}) - (A_{1x}V_{2y} - A_{1y}V_{2x})) / (V_{1x}V_{2y} - V_{1y}V_{2x}) \quad (5)$$

$$t_2 = ((A_{2x}V_{1y} - A_{2y}V_{1x}) - (A_{1x}V_{1y} - A_{1y}V_{1x})) / (V_{1x}V_{2y} - V_{1y}V_{2x}) \quad (6)$$

Note that the two denominators are the same. If its value is zero, t_1 and t_2 are infinity, which means the intersection point is at infinity, which further means that the two lines are parallel. For our purpose, this means there is no intersection point.

Equation of a polygon edge: A polygon is represented by its vertices, and therefore an edge of the polygon is represented by its two end points, \mathbf{P}_0 and \mathbf{P}_1 . For the equation of the edge, we may choose either point to be \mathbf{A} of Eqn (1). Let's choose $\mathbf{A} = \mathbf{P}_0$. Then the direction vector \mathbf{V} is given by $\mathbf{V} = \mathbf{P}_1 - \mathbf{P}_0$.

With this setup, you may note that at \mathbf{P}_0 , $t = 0$, and at \mathbf{P}_1 , $t = 1$. A point of intersection must lie within the edge, which means $0 \leq t \leq 1$. There is no intersection when the t value lies outside this range.

For two edges bounded by two pairs of end points, we have the parameters t_1 and t_2 (Eqn 2 and 3). An intersection point exists between the two edges if and only if $0 \leq t_1, t_2 \leq 1$.

Equation of a ray from a midpoint: When casting a ray from a midpoint \mathbf{M} , the start point is \mathbf{M} . Because we can cast it in any direction, we can choose a random end point, \mathbf{Q} , provided it is beyond the extent of the polygon we are dealing with. Then, \mathbf{M} and \mathbf{Q} are the equivalent of \mathbf{P}_0 and \mathbf{P}_1 in a polygon edge, and the same computation applies in finding intersection points.

Footnote: Those of you good in maths should be able to see that we can find t_1 more elegantly using vector algebra from Eqn (4) by taking the cross product of both sides of Eqn (4) with \mathbf{V}_2 :

$$(\mathbf{A}_1 + \mathbf{V}_1 t_1) \times \mathbf{V}_2 = (\mathbf{A}_2 + \mathbf{V}_2 t_2) \times \mathbf{V}_2$$

Rearranging, we get

$$(\mathbf{A}_1 \times \mathbf{V}_2) + (\mathbf{V}_1 \times \mathbf{V}_2) t_1 = (\mathbf{A}_2 \times \mathbf{V}_2) + (\mathbf{V}_2 \times \mathbf{V}_2) t_2.$$

But a vector crossed with itself is the zero vector, so $(\mathbf{V}_2 \times \mathbf{V}_2) t_2$ disappears. So, we are left with

$$t_1 = ((\mathbf{A}_2 \times \mathbf{V}_2) - (\mathbf{A}_1 \times \mathbf{V}_2)) / (\mathbf{V}_1 \times \mathbf{V}_2)$$

Expanding the cross products on the right-hand side, we get exactly Eqn (5). Similarly, we can take the cross product of both sides of Eqn (4) with \mathbf{V}_1 , which then leads to the value for t_2 as in Eqn (6).