

# Coordinate Geometry

## Why This Matters

Coordinate geometry connects **algebra** (numbers and equations) with **geometry** (shapes and space). It's the foundation for:

- **Computer graphics:** Pixels, rendering, transformations
- **Data visualization:** Charts, plots, scatter plots
- **Game development:** Position, movement, collision
- **Mapping:** GPS coordinates, navigation
- **UI layouts:** Positioning elements on screen

Understanding the coordinate plane is essential for visualizing mathematical relationships and working with spatial data.

---

## The Big Picture: Numbers Meet Space

**Before coordinate geometry:** Geometry was shapes (circles, triangles) without numbers.

**After coordinate geometry** (invented by Descartes): Every point has numbers (coordinates), and every shape has an equation.

Point: (3, 5)  
Line:  $y = 2x + 1$   
Circle:  $x^2 + y^2 = 25$

This merger of algebra and geometry revolutionized mathematics.

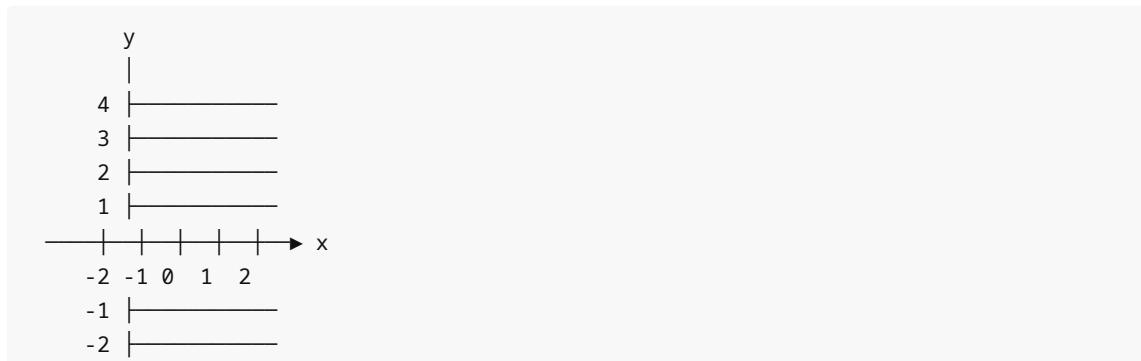
---

## 1. The Cartesian Plane

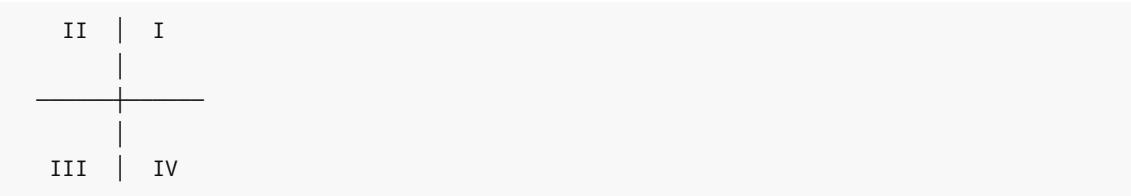
### Structure

The **Cartesian plane** (named after Descartes) has two perpendicular number lines:

- **x-axis:** Horizontal (left/right)
- **y-axis:** Vertical (up/down)
- **Origin:** Where they meet (0, 0)



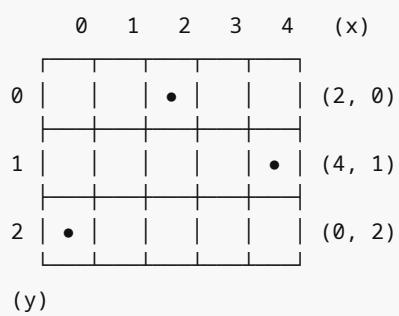
### The Four Quadrants



Quadrant	x	y	Example
I	+	+	(3, 2)
II	-	+	(-3, 2)
III	-	-	(-3, -2)
IV	+	-	(3, -2)

### Mental Model: A Grid

Think of it like a spreadsheet or image coordinates:



### Programming Analogy:

```
const point = { x: 3, y: 5 };

// Canvas coordinates (inverted y)
const canvasPoint = { x: 100, y: 200 };

// 2D array indexing
grid[y][x] = value; // [row][column]
```

## 2. Points: Ordered Pairs

### Notation

A **point** is written as **(x, y)**:

- **x**: horizontal distance from origin (left/right)
- **y**: vertical distance from origin (up/down)

(3, 2) means:  
- Go 3 units right ( $x = 3$ )

- Go 2 units up ( $y = 2$ )

## Order Matters

$(3, 5) \neq (5, 3)$

$(3, 5)$ :  $x=3, y=5$

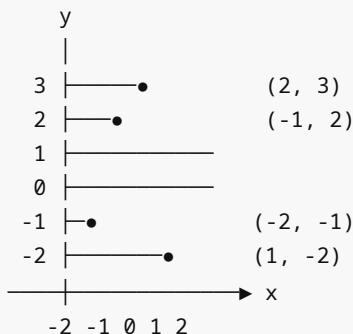
$(5, 3)$ :  $x=5, y=3$

**Analogy:** Like function parameters:

```
function plot(x, y) {  
    // (x, y) is ordered  
}  
  
plot(3, 5); // Not the same as plot(5, 3)
```

## Plotting Points

**Example:** Plot  $(2, 3), (-1, 2), (-2, -1), (1, -2)$



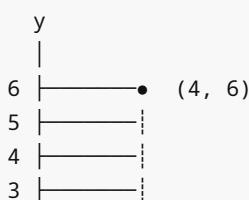
## Special Points

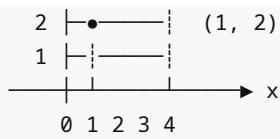
Origin:  $(0, 0)$  - Center  
On x-axis:  $(x, 0)$  -  $y$  is zero  
On y-axis:  $(0, y)$  -  $x$  is zero

## 3. Distance Between Two Points

### The Problem

What's the distance between  $(1, 2)$  and  $(4, 6)$ ?





We can't just subtract—that only works on a straight line.

## The Distance Formula

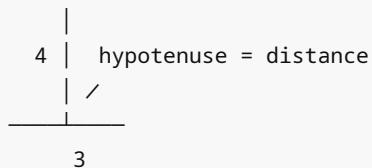
**Use the Pythagorean theorem:**

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

**Derivation:**

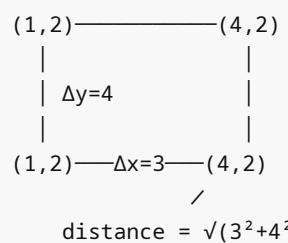
Horizontal distance:  $\Delta x = x_2 - x_1 = 4 - 1 = 3$   
 Vertical distance:  $\Delta y = y_2 - y_1 = 6 - 2 = 4$

Form a right triangle:



$$\text{Distance} = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

## Visual



## Examples

**Distance from origin to (3, 4):**

$$\begin{aligned} d &= \sqrt{(3-0)^2 + (4-0)^2} \\ &= \sqrt{9 + 16} \\ &= \sqrt{25} \\ &= 5 \end{aligned}$$

**Distance between (-1, 2) and (2, -2):**

$$\begin{aligned} \Delta x &= 2 - (-1) = 3 \\ \Delta y &= -2 - 2 = -4 \end{aligned}$$

$$d = \sqrt{(3^2 + (-4)^2)}$$

```
= √(9 + 16)
= √25
= 5
```

## Programming

```
function distance(p1, p2) {
  const dx = p2.x - p1.x;
  const dy = p2.y - p1.y;
  return Math.sqrt(dx*dx + dy*dy);
}

distance({x:1, y:2}, {x:4, y:6}); // 5

// Or destructured
const dist = Math.hypot(x2-x1, y2-y1);
```

## 4. Midpoint Between Two Points

### The Problem

What's the point exactly halfway between (1, 2) and (5, 8)?

### The Midpoint Formula

Average the coordinates:

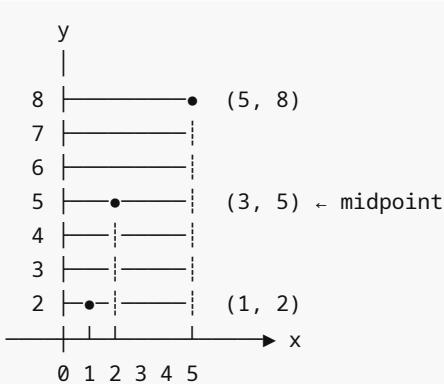
$$M = ((x_1+x_2)/2, (y_1+y_2)/2)$$

Example:

(1, 2) and (5, 8)

$$\begin{aligned} M &= ((1+5)/2, (2+8)/2) \\ &= (6/2, 10/2) \\ &= (3, 5) \end{aligned}$$

### Visual



The midpoint is equidistant from both endpoints.

## Why It Works

The midpoint is the **average position**:

- x-coordinate: average of  $x_1$  and  $x_2$
- y-coordinate: average of  $y_1$  and  $y_2$

**Programming:**

```
function midpoint(p1, p2) {
  return {
    x: (p1.x + p2.x) / 2,
    y: (p1.y + p2.y) / 2
  };
}

midpoint({x:1, y:2}, {x:5, y:8}); // {x:3, y:5}
```

## 5. Slope: Rate of Change

### What Is Slope?

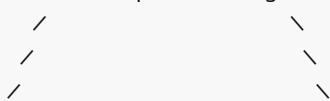
**Slope** measures how steep a line is—how much  $y$  changes per unit of  $x$ .

$$m = \frac{\text{rise}}{\text{run}}$$
$$m = (y_2 - y_1) / (x_2 - x_1)$$

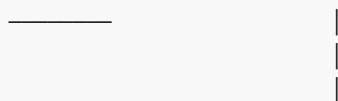
- **rise**: vertical change ( $\Delta y$ )
- **run**: horizontal change ( $\Delta x$ )
- **m**: slope (traditional symbol)

### Visual Intuition

Positive slope:      Negative slope:



Zero slope:      Undefined slope:



### Calculating Slope

**Example:** Slope between (1, 2) and (4, 8)

```

m = (y2 - y1) / (x2 - x1)
= (8 - 2) / (4 - 1)
= 6 / 3
= 2

```

**Interpretation:** For every 1 unit right, go up 2 units.

## Types of Slopes

Slope	Value	Shape	Example
Positive	$m > 0$	/ Rising	$m = 2$
Negative	$m < 0$	\ Falling	$m = -1$
Zero	$m = 0$	— Flat	Horizontal line
Undefined	$\Delta x = 0$	Vertical	Vertical line

## Special Cases

**Horizontal line:**  $y$  stays constant

```

(1, 3) to (5, 3)
m = (3 - 3) / (5 - 1) = 0 / 4 = 0

```

**Vertical line:**  $x$  stays constant

```

(2, 1) to (2, 5)
m = (5 - 1) / (2 - 2) = 4 / 0 = undefined

```

**Note:** Division by zero! Vertical lines have no slope (or "infinite" slope).

## Programming Analogy: Velocity

```

// Slope is like velocity (rate of change)
const velocity = (finalPosition - initialPosition) / time;

// Slope
const slope = (y2 - y1) / (x2 - x1);

// In animation
const speed = deltaY / deltaX; // pixels per frame

```

## 6. Graphing on the Plane

### Plotting Data Points

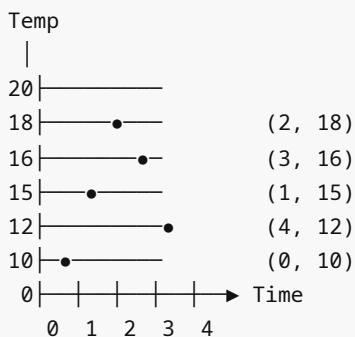
**Example:** Temperature over time

```

Time (h): [0, 1, 2, 3, 4]
Temp (°C): [10, 15, 18, 16, 12]

```

```
Points: (0,10), (1,15), (2,18), (3,16), (4,12)
```



## Scatter Plots

**Correlation between variables:**

```
const data = [
  {x: 1, y: 2},
  {x: 2, y: 4},
  {x: 3, y: 5},
  {x: 4, y: 7}
];

// Positive correlation (upward trend)
```

## Understanding Graphs

A graph shows **relationships** between variables:

- **x-axis:** Independent variable (input)
- **y-axis:** Dependent variable (output)

## 7. Real-World Applications

### Computer Graphics

```
// Screen coordinates (origin top-left)
const player = { x: 100, y: 200 };

// Update position
player.x += velocityX;
player.y += velocityY;

// Distance to enemy
const enemy = { x: 300, y: 400 };
const dist = Math.hypot(enemy.x - player.x, enemy.y - player.y);
```

## GPS Coordinates

```
Latitude: y (North/South)
Longitude: x (East/West)

New York: (40.7°N, 74.0°W) → (40.7, -74.0)
London: (51.5°N, 0.1°W) → (51.5, -0.1)
```

## Data Visualization

```
// Chart library (e.g., Chart.js)
const chartData = {
  labels: ['Jan', 'Feb', 'Mar'],
  datasets: [
    data: [10, 20, 15] // Points: (0,10), (1,20), (2,15)
  ]
};
```

## Collision Detection

```
// Circle collision (distance < sum of radii)
function checkCollision(circle1, circle2) {
  const dist = Math.hypot(
    circle2.x - circle1.x,
    circle2.y - circle1.y
  );
  return dist < (circle1.radius + circle2.radius);
}
```

## Path Finding

```
// A* algorithm uses distance heuristic
function heuristic(node, goal) {
  return Math.abs(node.x - goal.x) + Math.abs(node.y - goal.y);
}
```

## 8. Transformations (Preview)

### Translation (Moving)

Shift every point by (dx, dy):

```
(x, y) → (x + dx, y + dy)
```

```
Example: Move (3, 2) by (1, -1)
Result: (4, 1)
```

```
function translate(point, dx, dy) {  
    return { x: point.x + dx, y: point.y + dy };  
}
```

## Reflection

**Over x-axis:** Flip y

$$(x, y) \rightarrow (x, -y)$$

$$(3, 2) \rightarrow (3, -2)$$

**Over y-axis:** Flip x

$$(x, y) \rightarrow (-x, y)$$

$$(3, 2) \rightarrow (-3, 2)$$

## Rotation (Around Origin)

90° counterclockwise:

$$(x, y) \rightarrow (-y, x)$$

$$(3, 2) \rightarrow (-2, 3)$$

## Scaling

Multiply coordinates:

$$(x, y) \rightarrow (sx \times x, sy \times y)$$

$$\text{Scale by 2: } (3, 2) \rightarrow (6, 4)$$

---

## Common Mistakes & Misconceptions

**✗ "x is always the first number"**

True, but remember: **x is horizontal, y is vertical**. Don't confuse them.

**✗ "(x, y) and (y, x) are the same"**

No! Order matters:

(3, 5) is at x=3, y=5  
(5, 3) is at x=5, y=3 (different point)

**✗ "Distance can be negative"**

Distance is always positive (or zero):

$d = \sqrt{(\dots)}$  always gives positive result

### ✗ "Slope is always a simple fraction"

Slope can be any real number:

$m = 2.5$  (positive decimal)  
 $m = -\sqrt{2}$  (negative irrational)  
 $m = 0$  (horizontal)  
 $m = \text{undefined}$  (vertical)

### ✗ "The origin is always in the middle"

It's wherever the axes cross. In many graphics systems,  $(0,0)$  is the top-left corner.

## Tiny Practice

Plot these points:

1.  $(2, 3)$
2.  $(-1, 4)$
3.  $(-3, -2)$
4.  $(4, -1)$

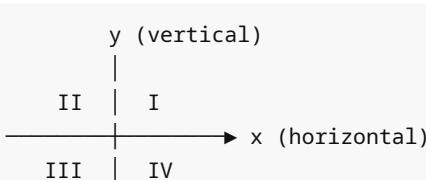
**Calculate:** 5. Distance between  $(0, 0)$  and  $(3, 4)$  6. Distance between  $(1, 2)$  and  $(4, 6)$  7. Midpoint of  $(2, 3)$  and  $(6, 7)$  8. Slope between  $(1, 2)$  and  $(3, 8)$  9. Slope of horizontal line through  $(2, 5)$  and  $(7, 5)$  10. Slope of vertical line through  $(3, 1)$  and  $(3, 9)$

**True or False:** 11.  $(3, 5)$  is in Quadrant I 12.  $(-2, -4)$  is in Quadrant III 13. The point  $(0, 5)$  is on the x-axis

► Answers

## Summary Cheat Sheet

### The Cartesian Plane



### Points

$(x, y)$  = ordered pair

$x$ : horizontal position

$y$ : vertical position

Origin:  $(0, 0)$

## Distance Formula

```
d = sqrt((x2-x1)2 + (y2-y1)2)
```

Based on Pythagorean theorem

## Midpoint Formula

```
M = ((x1+x2)/2, (y1+y2)/2)
```

Average of coordinates

## Slope

$$m = \frac{\text{rise}}{\text{run}} = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

Positive: / (rising)

Negative: \ (falling)

Zero: — (horizontal)

Undefined: | (vertical)

## Programming Patterns

```
// Point representation
const point = { x: 3, y: 5 };

// Distance
const dist = Math.sqrt((x2-x1)**2 + (y2-y1)**2);
// or
const dist = Math.hypot(x2-x1, y2-y1);

// Midpoint
const mid = { x: (x1+x2)/2, y: (y1+y2)/2 };

// Slope
const m = (y2-y1) / (x2-x1);
```

## Next Steps

You now understand the coordinate plane—how to locate points, measure distances, and calculate slopes. This is the visual foundation for understanding functions.

Next, we'll explore **Functions**—the single most important concept in mathematics and programming.

Continue to: [06-functions.md](#)