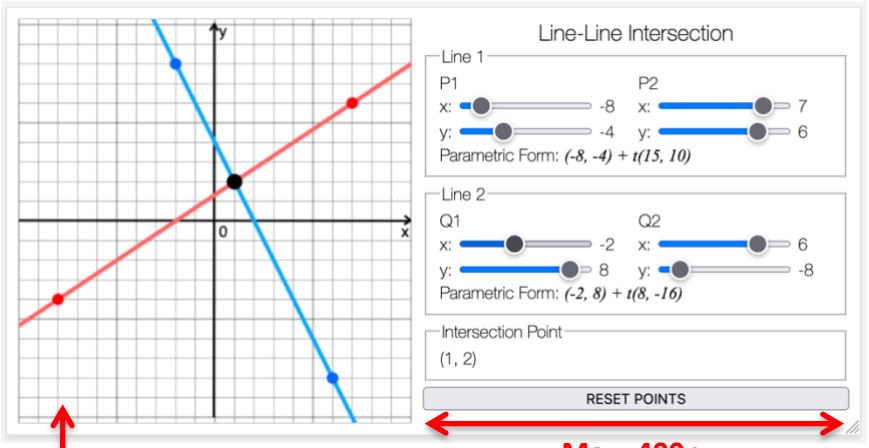
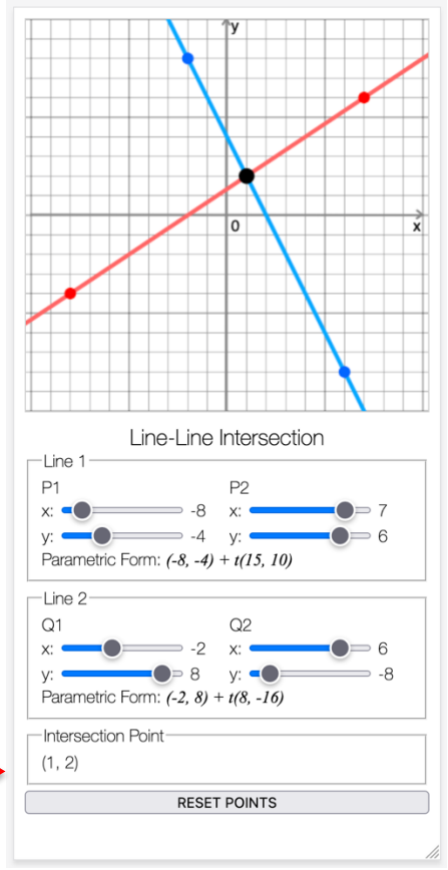
# Assignment 2: Line Intersection Calculator

## Description

Write a responsive web application to find the intersection point of 2 lines, and draw the lines and the intersection point on the <canvas>.. Your application provides user interface (sliders) to change the 2D coordinates of 2 points of each line, and displays the parametric forms of the lines and the intersection point up to 3 decimal places. If 2 lines are not met, display (NaN, NaN) instead. When the application is launched or the reset button is pressed, it uses the following default points;

- **Line 1**: P1 (-8, -4), P2 (7, 6)
- **Line 2**: Q1 (-2, 8), Q2 (6, -8)




## General Requirements

- Must use your own HTML/CSS/JavaScript (Not allow to use third-party libraries)
- Must be unobtrusive (no inline CSS and JavaScript)
- Must be validated (no errors or warnings in HTML, CSS & JavaScript)
- Must be responsive using full device width and height (All content should fit in 800x400)
- Must use flex layout; 2 columns for landscape and 2 rows for portrait mode
- Must use <canvas> for drawing 2D graphics
- Must define separate files for Vector2.js, Line.js and main.js
- Must handle “input” events for “range” input elements using addEventListener()
- Must use relative paths to reference files (No absolute path)
- Must be accurate

## Deliverables

An archive file, **Assignment2-*<yourname>*.zip**, which contains all the files (HTML, CSS, JavaScript).
***NOTE**: You must include the file header at the beginning of each file. The header must contain a short description, your name, email, date, etc.*

## Submission and Due Date

Submit your deliverables to SLATE/Assignments/Assignment2 by **Saturday, Jul. 8, 11:59 PM**.
You may submit multiple versions, but only the latest version will be evaluated.
***NOTE**: Late submission will be deducted 10% per day. (max. 3 days)*
***NOTE**: Partial implementation will be accepted.*
***NOTE**: This assignment is individual work and subject to Sheridan Academic Integrity Policy.*

## References

- Note01 ~ Note04, Lab01 ~ Lab04
- http://www.songho.ca/math/line/line.html

## Tasks and Evaluations (Total 100 points)

**Task 1: HTML (20 points)**

1. Construct DOM elements same as the screenshot above by referencing testCanvas.html
   **REFERENCE**: testCanvas.html at SLATE/Assignment2
2. The range inputs varies from -10 to + 10 incremented by 0.1
3. Load necessary CSS and JavaScript files
4. Not allowed inline CSS styles and JavaScript codes
5. Validate HTML (No warning and error)

**Task 2: CSS (20 points)**

1. Use full device width and height
2. Use 2-column flex layout for the main container block, and switch the column direction depending on device orientation
3. The width of the right-side control block should be 400px or less
4. The dimension of the left-side <canvas> block should be dynamically resized
5. Make sure all content are fit well in 800x400 screen
6. Use bigger font sizes for the title

**Task 3: JavaScript (60 points)**

1. Extend your **Vector2 class (Vector2.js in Lab03)** by including the following member functions
   a. **clone()**: Make deep copy and return a new Vector2 object
   b. **add(v)**: Add the given vector to this object; $\overrightarrow{v1} \leftarrow \overrightarrow{v1} + \overrightarrow{v2}$
   c. **subtract(v)**: Subtract the given vector to this object; $\overrightarrow{v1} \leftarrow \overrightarrow{v1} - \overrightarrow{v2}$
   d. **scale(s**): Multiply a scalar to this object; $\overrightarrow{v1} \leftarrow s \cdot \overrightarrow{v1}$
   e. **normalize()**: Make this vector object as a unit length; $(x, y) \leftarrow \left(\frac{x}{\sqrt{x^2+y^2}}, \frac{y}{\sqrt{x^2+y^2}}\right)$

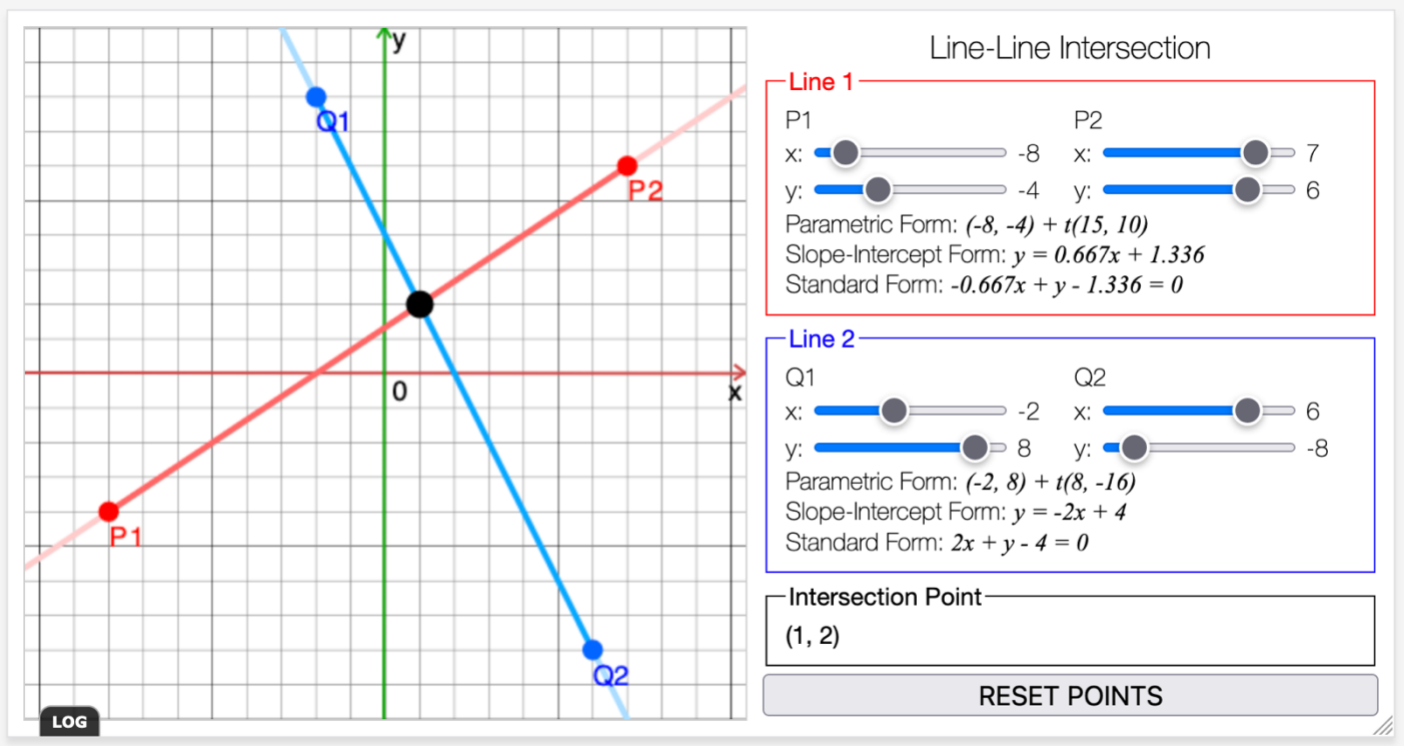
   **HINT**: http://www.songho.ca/glsl/files/js/Vectors.js
2. Extend your **Line class (Line.js in Lab03)** by including the following member functions
   a. **intersect(line)**: Return the intersection point of this object and the given line as Vector2
   b. **isIntersect(line)**: Return a Boolean if the given line is intersected with this object
   c. **getPointX(y)**: Return the x value on the line for the given y value
   d. **getPointY(x)**: Return the y value on the line for the given x value

   **HINT**: See **Q1** & **Q2** below
3. Define **main.js** for event handlers and drawing line graphs
   a. Register the event handlers for all UI components
   b. Initialize the lines with (-8,-4), (7,6) and (-2,8), (6,-8) when the page is loaded or the reset button is clicked
   c. Compute and print the intersection point up to 3 decimal places whenever the inputs are changed
   d. Redraw the line graphs and intersection point whenever the inputs are changed
   e. Update the canvas dimension whenever the screen is resized

   **REFERENCE**: http://www.songho.ca/math/line/line.html#example_lineline

## **Bonus** (20 points max)

- Enhance visual with colour coding, e.g. red for the line 1, blue for the line 2
- Use different colours for X/Y axes and every multiple of 5 grid lines
- Add standard or/and slope-intercept equations
- Add additional functions to JavaScript classes; dot(), length(), distance() for Vector2.js
- Add additional features to enhance the application




## Q & A (Extra Notes)

**Q1. How to find the intersection point of 2 lines?**

A way to find the intersection point is to solve a linear system of 2 line equations in standard form;

| Solve for x: | Solve for y: |
|---|---|
| $\begin{cases} ax + by = c & (eq1) \\ dx + ey = f & (eq2) \end{cases}$ | $\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$ |
| 1. Multiply $e$ and $b$ on each eq. $\begin{cases} aex + bey = ce \\ bdx + bey = bf \end{cases}$ | 1. Multiply $d$ and $a$ on each eq. $\begin{cases} adx + bdy = cd \\ adx + aey = af \end{cases}$ |
| 2. Subtract *eq1-eq2* $(ae - bd)x = ce - bf$ | 2. Subtract *eq2-eq1* $(ae - bd)y = af - cd$ |
| 3. Solve for ***x*** $x = \frac{ce - bf}{ae - bd}$ | 3. Solve for ***y*** $y = \frac{af - cd}{ae - bd}$ |

**NOTE**: If the denominator of x and y is 0, then there is no solution (no intersect) because of division by zero. The denominator $(ae - bd)$ is called determinant.

Use the determinant in **isIntersect**() method;

```
// return the intersection point as Vector2
isIntersect(line)
{
    // return true if (ae - bd) != 0, otherwise return false

}
```

**Conversion from Parametric to Standard form**

Since our line class uses parametric form, we have to convert it to standard form $(ax + by = c)$ in order to compute the intersection point.

| Parametric Form | Standard Form |
|---|---|
| $\frac{x - x_1}{v_x} = \frac{y - y_1}{v_y} = t$<br>$v_y(x - x_1) = v_x(y - y_1)$<br>$v_y x - v_y x_1 = v_x y - v_x y_1$<br>$v_y x - v_x y = v_y x_1 - v_x y_1$ | $v_y x - v_x y = v_y x_1 - v_x y_1$<br>$a$ $b$ $c$<br>Coefficients of standard form<br>$ax + by = c$ or $dx + ey = f$ |

```
// return the intersection point as Vector2
intersect(line)
{
    // find coefficients of line1: aX + bY = c
    let a =
    let b =
    let c =

    // find coefficients of line2: dX + eY = f
    let d =
    let e =
    let f =

    // find determinant: ae - bd
    let det =

    // find the intersect point if det != 0:
    // x = (ce - bf) / det
    // y = (af - cd) / det
    // if det=0, return a Vector2 with NaN
    if(det == 0)
    {

    }
    else
    {

    }
}
```

**Q2. How to find the point (x, y) on the line if you know only x or y coordinate?**

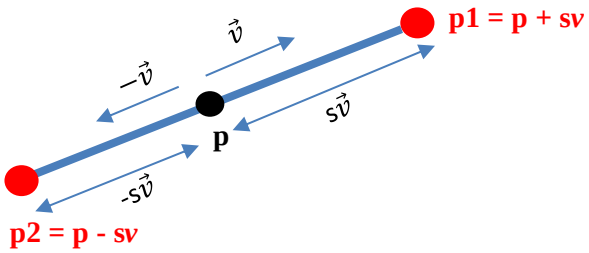
$$\begin{cases} x = x_1 + t(x_2 - x_1) \\ y = y_1 + t(y_2 - y_1) \end{cases}$$

1. Find t value using the known coordinate from the parametric form
2. Substitute t in the other coordinate equation

```
// return x-ccord for a given y value on the line
getPointX(y)
{
    // find t = (y - y1) / (y2 - y1)
    let t =
    // find x using x = x1 + t(x2 - x1)
    x =
}
```

**Q3. How to draw a line segment from a line equation?**

From a point on the line, extend the line along the direction vector. In order to extend a certain length, *s*, you need to normalize the direction vector (make it a unit length) and multiply the amount, *s*.




```
let v = line1.direction.clone().normalize();      // unit length direction vec
let p1 = point.clone().add(v.clone().scale(s));   // move s unit along v
let p2 = point.clone().add(v.clone().scale(-s)); // move -s unit along v

// draw line
context.beginPath();
context.moveTo(p1.x, p1.y);
context.lineTo(p2.x, p2.x);
context.stroke();
```