JavaScript – Module 5

Dr. L.M. Jenila Livingston
Professor
VIT Chennai

Module 5

- External JavaScript Files
- Manipulating CSS with JavaScript
- While Loop, do Loop, for Loop
- Radio Buttons, Checkboxes,
- fieldset and legend Elements
- Textarea Controls
- Pull-Down Menus- List Boxes
- Using z-index to Stack Elements
- Canvas and Drawing
- Event Handler and Listener.

Presentation Overview

- Statements
  - Conditional statements
  - Looping Statements
- Date and Time

JavaScript Statements

- JavaScript statements are separated by semicolon
  - **Conditional statements**
  - **Looping statements**

Conditional Statement

- “if … else” statement
- “switch” statement

“if … else” statement

```
if (condition) { statement; }
else if (condition) { statement; }
else { statement; }
```

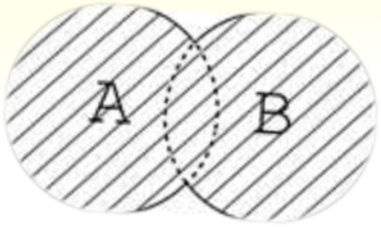
Conditional Statement (`if`)

```
unitPrice = 1.30;
if (quantity > 100) {
  unitPrice = 1.20;
}
```

Symbol	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal
!=	Not equal

Conditional Statement (`if`) (2)

- The condition may be of Boolean or integer type:



conditional-statements.html

```
var a = 0;
var b = true;
if (typeof(a)=="undefined" || typeof(b)=="undefined") {
  document.write("Variable a or b is undefined.");
}
else if (!a && b) {
  document.write("a==0; b==true;");
} else {
  document.write("a==" + a + "; b==" + b + ";");
}
```

“switch” statement

```
switch (expression) {
        case label1:
                statements;
                break;
        default:
                statements;
}
```

- Allows you to merge several evaluation tests of the same variable into a single block of statements.

Switch Statement

- The `switch` statement works like in C#:

switch-statements.html

```
switch (variable) {
  case 1:
    // do something
    break;
  case 'a':
    // do something else
    break;
  case 3.14:
    // another code
    break;
  default:
    // something completely different
}
```

Loops

- `for` loop
- `while` loop
- `do … while` loop

“for” loop

```
for (initialization; condition; increment/decrement){
        statements;
}
```

```
var counter;
for (counter=0; counter<4; counter++) {
  alert(counter);
}
```

“while” loop

```
initialization;
while (entry-condition) {
        statements;
        increment/decrement;
}
```

```
var counter;
for (counter=0; counter<4; counter++) {
  alert(counter);
}
```

“do … while” loop

```
initialization;
do {
        statements;
         increment/decrement;
 } while (exit-condition)
```

Java Script Date

- <!DOCTYPE html>
- <html>
- <body>
- <p id="demo"></p>
- <script>
- document.getElementById("demo").innerHTML = **Date();**
- </script>
- </body>
- </html>

Date and Time

```
<SCRIPT LANGUAGE = "JavaScript">
var current = new Date();
document.writeln(current);
document.writeln( "<H1>Get methods for local time zone</H1>" );
document.writeln( "getDate: " + current.getDate() +
 "<BR>getDay: " + current.getDay() +
"<BR>getMonth: " + current.getMonth() +
"<BR>getFullYear: " + current.getFullYear() +
 "<BR>getTime: " + current.getTime() +
"<BR>getHours: " + current.getHours() +
"<BR>getMinutes: " + current.getMinutes() +
 "<BR>getSeconds: " + current.getSeconds() +
"<BR>getMilliseconds: " + current.getMilliseconds() +
"<BR>getTimezoneOffset: " + current.getTimezoneOffset() );
```

Date and Time

```
document.writeln( "<H1>Specifying arguments for a new
Date</H1>" );
var D1 = new Date( 1999, 2, 18, 1, 5, 3, 9 );
document.writeln( "Date: " + D1 );
document.writeln( "<H1>Set methods for local time zone</H1>" );

D1.setDate( 31 );
D1.setMonth( 11 );
D1.setFullYear( 1999 );
D1.setHours( 23 );
D1.setMinutes( 59 );
D1.setSeconds( 59 );
document.writeln( "Modified date: " + D1 );
</SCRIPT>
```

Example - Date

new Date() puts the current time/date (from your computer) into the variable currentTime:

```
<script language="JavaScript">
currentTime=new Date();
if (currentTime.getHours() < 12)
  document.greet.greetingbox.value="Good morning!"
else if (currentTime.getHours() < 17)
  document.greet.greetingbox.value="Good afternoon!"
else document.greet.greetingbox.value="Good evening!"
</script>
```

Thank You

Event Listeners in JavaScript

Dr. L.M. Jenila Livingston
Professor
VIT Chennai

Event Listener

- An **event listener** is a function in JavaScript that **waits** for an event (like a click, keypress, or mouse movement) to occur on an HTML element and then executes a specified function in response.
- add an event listener by using the **addEventListener()** method

Basic Syntax

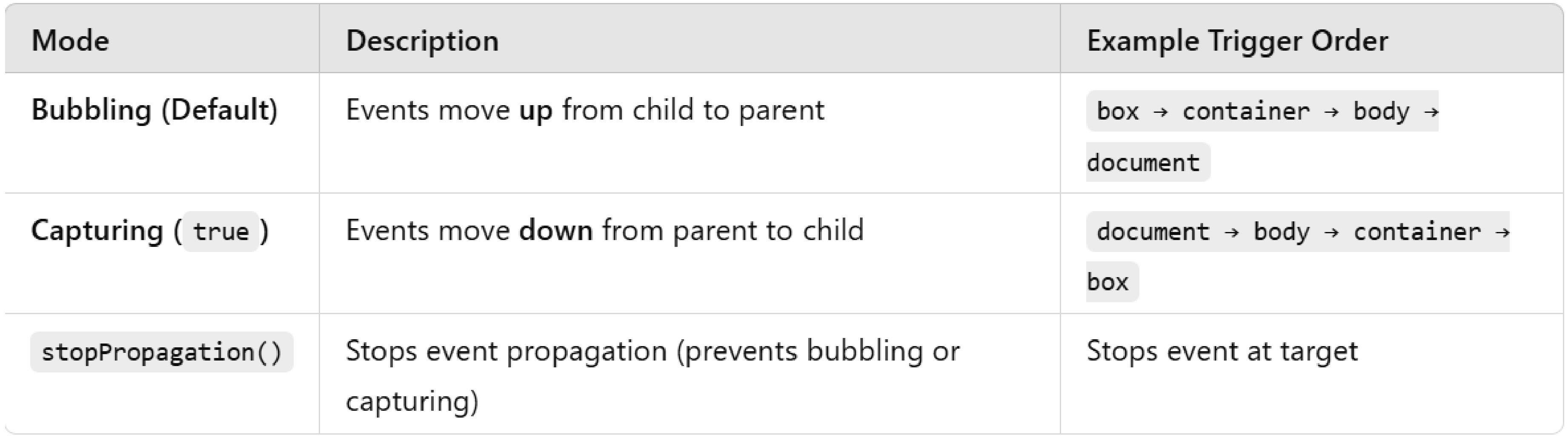
element.addEventListener(event, function, useCapture);

- **element:** The HTML element to which the event listener is attached.
- **event:** Type of event (e.g., 'click', 'mouseover', 'keydown').
- **function:** Function to execute when the event occurs.
- **useCapture (optional):** true for capturing, false for bubbling (default).

Syntax

element.addEventListener(event, function, useCapture);

- If **useCapture is true** → **Capturing Phase** (Top-Down: Parent ⟶ Child).
- If **useCapture is false** (or omitted) → **Bubbling Phase** (Bottom-Up: Child ⟶ Parent). **default** behavior



Mode	Description	Example Trigger Order
Bubbling (Default)	Events move **up** from child to parent	`box → container → body → document`
Capturing (`true`)	Events move **down** from parent to child	`document → body → container → box`
`stopPropagation()`	Stops event propagation (prevents bubbling or capturing)	Stops event at target

Click Event Example

```
document.getElementById("myButton").addEventListener("click", function() {
    alert("Button Clicked!");
});
```

Call Another function:

-
  ```
  document.getElementById("myButton").addEventListener("click", function() { showMessage();
  });
  ```

-
  ```
  function showMessage() {
  alert("Button Clicked! Function Called."); }
  ```

Removing Event Listener

Directly pass the function name (without parentheses):

- document.getElementById("myButton").**addEventListener**("click", **greet**);

- function **greet()** {
  alert("Hello!");
  }

- document.getElementById("myButton").**removeEventListener**("click", greet);

Mouseover Event Example

```
document.getElementById("myDiv").addEventListener("mouseover", function() {
   this.style.backgroundColor = "yellow";
});
```

The “this” keyword refers to the element that triggered the event.

Keyboard Event Example

```
document.addEventListener("keydown",
function(event) {
    alert("Key pressed: " + event.key);
});
```

Arrow Functions in Event Listeners

```
document.getElementById("myButton").addEventListener
("click", () => {
   alert("Arrow Function Clicked!");
});
```

//Call another function:

```
document.getElementById("myButton").addEventListener
("click", () => {
myFunction();
});
function myFunction() {
alert("Arrow Function Called Another Function!");
}
```

Handling Multiple Events

```
let btn = document.getElementById("myButton");

// First event listener
btn.addEventListener("click", function() {
    console.log("First Event Listener Triggered!");
});

// Second event listener
btn.addEventListener("click", function() {
    console.log("Second Event Listener Triggered!");
});
```

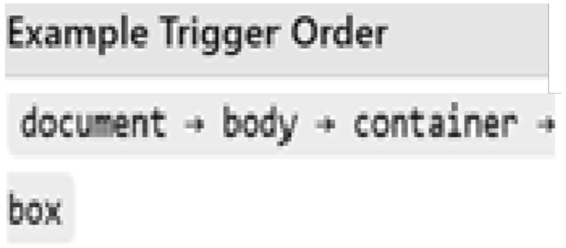
```
First Event Listener Triggered!
Second Event Listener Triggered!
```

Event Propagation (Bubbling & Capturing)

- **//Events move down from parent to child**
- document.getElementById("outer").addEventListener("click", function() {

  alert("Outer Div Clicked!");

  }, **true);** // Capturing phase



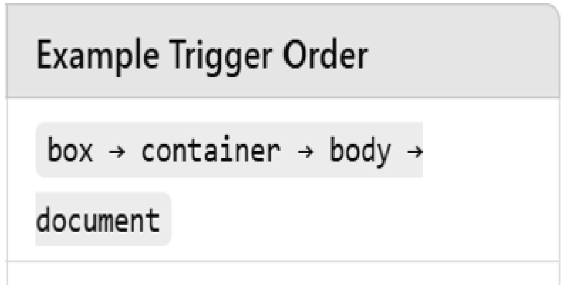
Example Trigger Order

document → body → container → box

- **//Events move up from child to parent**
- document.getElementById("inner").addEventListener("click", function() {

  alert("Inner Div Clicked!");

  }, **false);** // Bubbling phase (default)

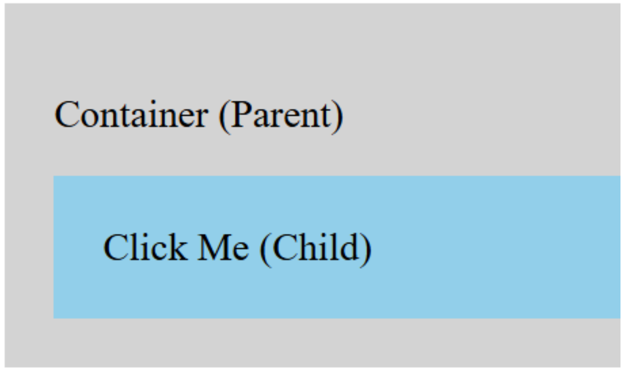


Example Trigger Order

box → container → body → document

Example – Handling Multiple Events

```
<head>
     <style>
        .container {
            padding: 20px;
            background-color: lightgray;
        }
        .box {
            padding: 20px;
            background-color: skyblue;
            cursor: pointer;
        }
    </style>
</head>
<body>

    <div class="container">
        <p>Container (Parent)</p>
        <div class="box">Click Me (Child)</div>
    </div>
```



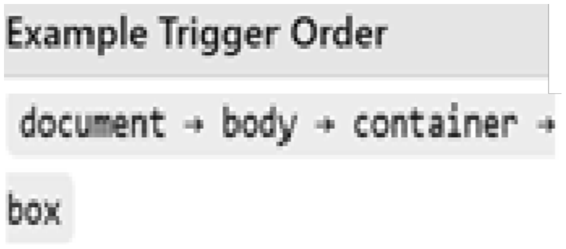
//Box inside the containter

Capturing

//Events move down from parent to child

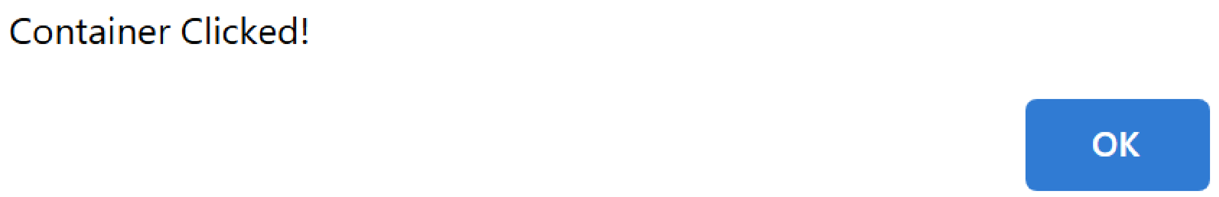
```
<script>
    document.querySelector(".container").addEventListener("click", function()
        alert("Container Clicked!");
    },true);

    document.querySelector(".box").addEventListener("click", function() {
        alert("Box Clicked!");
    });
</script>
```

Output sequence on clicking the child

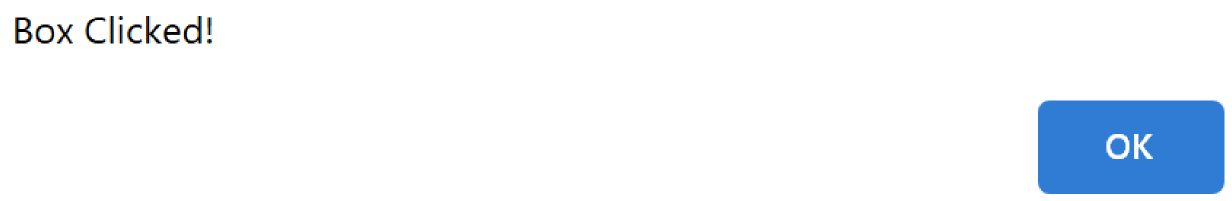


Example Trigger Order

document → body → container → box



Container Clicked!



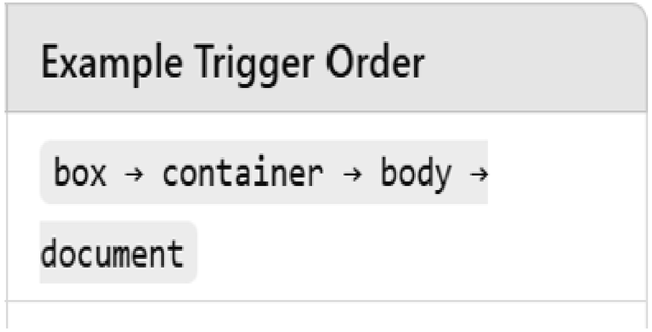
Box Clicked!

Bubbling

//Events move up from child to parent

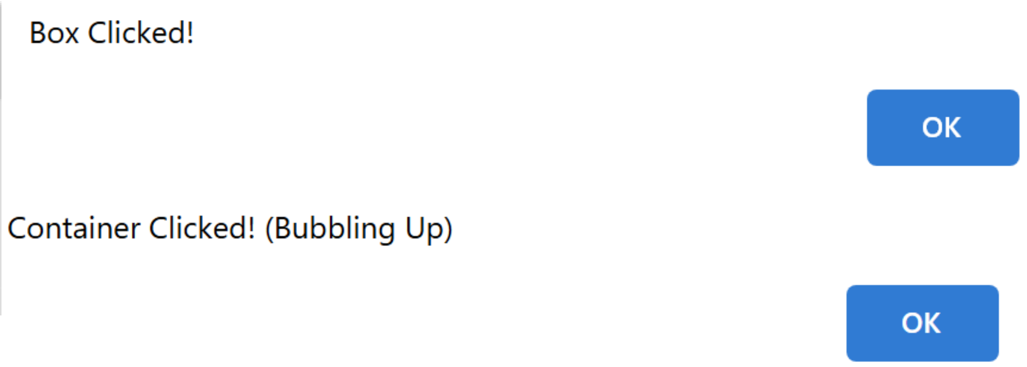
```
<script>
    document.querySelector(".container").addEventListener("click", function() {
        alert("Container Clicked! (Bubbling Up)");
    });

    document.querySelector(".box").addEventListener("click", function() {
        alert("Box Clicked!");
    });
</script>
```

Output sequence on clicking the child



Example Trigger Order

box → container → body → document



Box Clicked!

Container Clicked! (Bubbling Up)

By default, the **third parameter** of addEventListener() is **false**, which means **event bubbling** is the default behavior.

stopPropagation

```
<script>
    document.querySelector(".container").addEventListener("click", function() {
        alert("Container Clicked!");
    }); // Bubbling mode

    document.querySelector(".box").addEventListener("click", function(event) {
        alert("Box Clicked!");
        event.stopPropagation(); // Stops bubbling or capturing
    });
</script>
```

Output:

```
Box Clicked!
```

Key Differences
Event Handling vs Event Listener

Feature	Event Handling	Event Listener
Definition	Process of responding to user actions.	Function that listens for an event and executes a callback.
Method	\`element.onclick = function() {...}\`	\`element.addEventListener(' event', function) {...}\`
Multiple Handlers	❌ No (Overwrites previous handler)	✅ Yes (Allows multiple handlers for the same event)
Best Practice	❌ Not preferred	✅ Recommended

Thank You!

Position in CSS & Using z-index to Stack Elements

Dr. L.M. Jenila Livingston
Professor
VIT Chennai

What is position in CSS?

- The position property defines **how an element is positioned** in the document.
- It determines whether the element follows normal flow or is positioned independently.
- **Types:** static, relative, absolute, fixed, sticky.

Types of Position in CSS

Position Type	Description
Static	Default position, follows **normal** document flow. It does not accept properties like top, left, right, or bottom.
Relative	Positioned **relative to its normal position**.
Absolute	Positioned **relative to the nearest** positioned (non-static) ancestor i.e., with a positioning context (relative, absolute, or fixed).
Fixed	Positioned relative to the viewport, **does not move when scrolling.**
Sticky	Behaves like relative until scrolling, then acts like fixed.

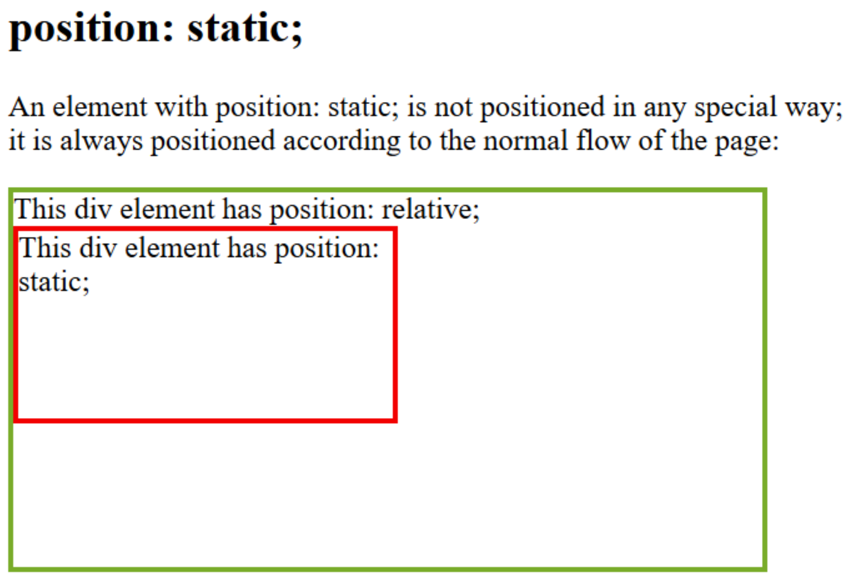
Position in CSS

- **Static:** The default behavior of elements (normal flow).
- **Relative:** Moves based on its normal position.
- **Absolute:** Moves based on its parent element.
- **Fixed:** Remains fixed in place even while scrolling.
- **Sticky:** Moves with scrolling and stops at a specific position.

static:
normal position

It does not accept properties like top, left, right, or bottom.

```
<head>
<style>
div.relative {
  width: 400px;
  height: 200px;
  border: 3px solid #73AD21;
}

div.static {
  position: static;
  top:80px;
  right:10px;
  width: 200px;
  height: 100px;
  border: 3px solid red;
}
</style>
</head>
<body>

<h2>position: static;</h2>
An element with position: static; is not positioned in any special way;<br>
it is always positioned according to the normal flow of the page:</p>

<div class="relative">This div element has position: relative;
  <div class="static">This div element has position: static;</div>
</div>

</body>
```



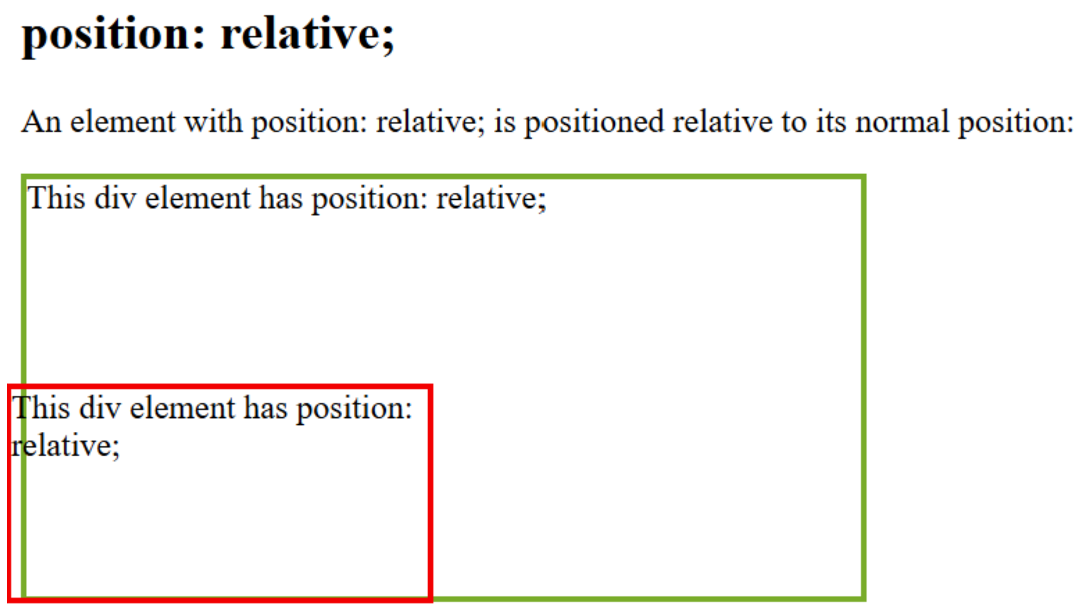
position: static;

An element with position: static; is not positioned in any special way;
it is always positioned according to the normal flow of the page:

This div element has position: relative;
This div element has position: static;

relative

```
<head>
<style>
div.relative {
  width: 400px;
  height: 200px;
  border: 3px solid #73AD21;
}

div.relative2 {
  position: relative;
  top:80px;
  right:10px;
  width: 200px;
  height: 100px;
  border: 3px solid red;
}
</style>
</head>
<body>

<h2>position: relative;</h2>

<p>An element with position: relative; is positioned relative to its normal position:</p>

<div class="relative">This div element has position: relative;
  <div class="relative2">This div element has position: relative;</div>
</div>

</body>
```



position: relative;

An element with position: relative; is positioned relative to its normal position:

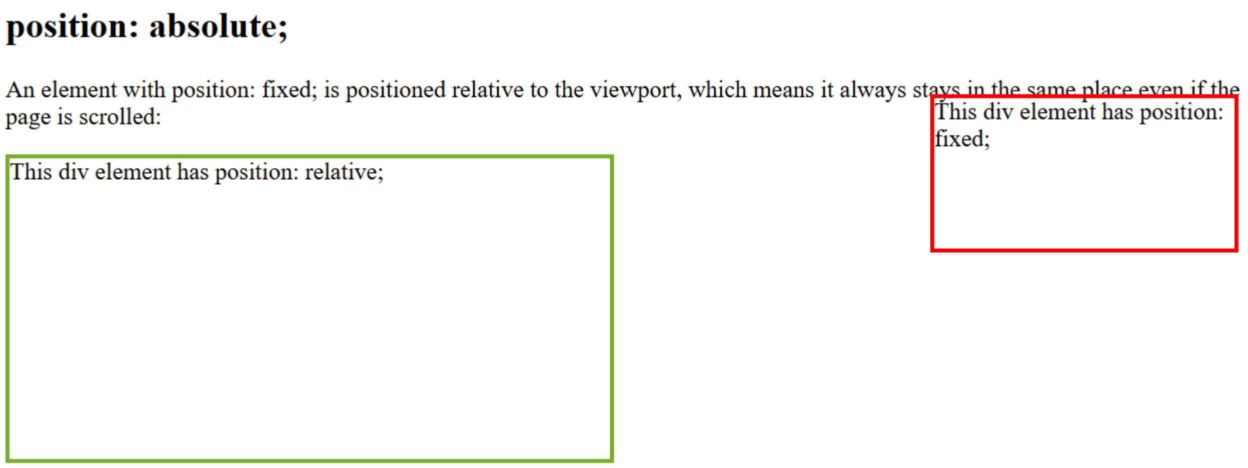
This div element has position: relative;

This div element has position: relative;

Inner box is shifted 80px down and 10px to the left from its normal position.

fixed

```
<head>
<style>
div.relative {
  position: relative;
  width: 400px;
  height: 200px;
  border: 3px solid #73AD21;
}

div.fixed {
  position: fixed;
  top: 80px;
  right: 10px;
  width: 200px;
  height: 100px;
  border: 3px solid red;
}
</style>
</head>
<body>

<h2>position: fixed;</h2>

<p>An element with position: fixed; is positioned relative to the viewport,
which means it always stays in the same place even if the page is scrolled:</p>

<div class="relative">This div element has position: relative;
  <div class="fixed">This div element has position: fixed;</div>
</div>

</body>
```



position: absolute;

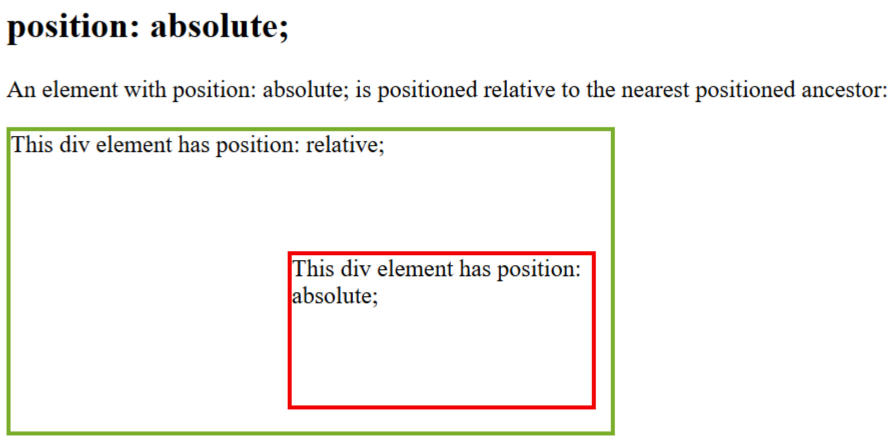
An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled:

This div element has position: relative;

This div element has position: fixed;

Absolute

```
<head>
<style>
div.relative {
  position: relative;
  width: 400px;
  height: 200px;
  border: 3px solid #73AD21;
}

div.absolute {
  position: absolute;
  top: 80px;
  right: 10px;
  width: 200px;
  height: 100px;
  border: 3px solid red;
}
</style>
</head>
<body>

<h2>position: absolute;</h2>

<p>An element with position: absolute; is positioned relative to the
nearest positioned ancestor:</p>

<div class="relative">This div element has position: relative;
  <div class="absolute">This div element has position: absolute;</div>
</div>

</body>
```



position: absolute;

An element with position: absolute; is positioned relative to the nearest positioned ancestor:

This div element has position: relative;

This div element has position: absolute;

What is z-index?

- The z-index property in CSS controls the stacking order of elements.
- **Elements with higher z-index values appear in front of those with lower values**.
- It only works on elements with position: **relative, absolute, fixed, or sticky (should not be static (default).**

z-index values

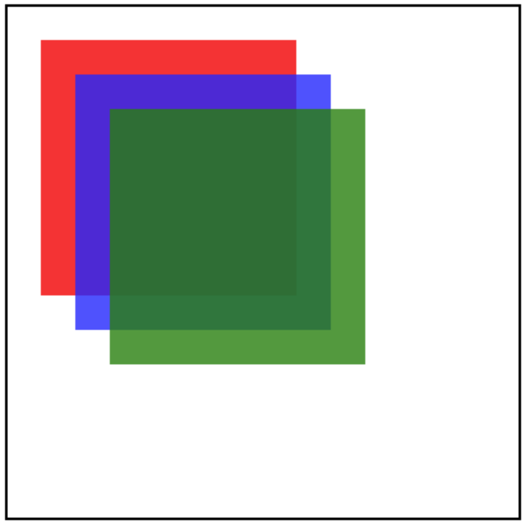
- **Negative values**: Push elements behind other elements.
- **Zero (0)**: Default stacking position (same as z-index is not specified).
- **Positive values**: Bring elements in front of others.
- **Auto**: The element inherits the stacking context of its parent.
- When z-index values are equal, the element that appears later in the document structure (HTML) will be displayed in front of the others.

Syntax of z-index

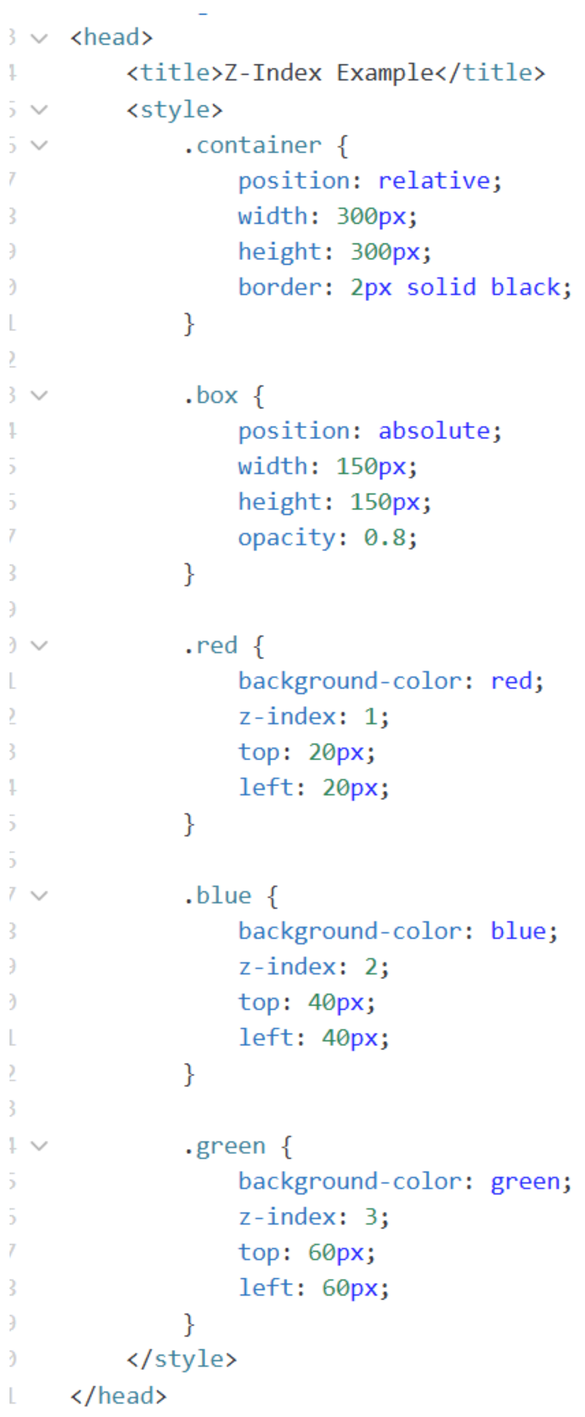
```
.element {
    position: absolute; /* Required for z-index to work */
    z-index: 10; /* Higher value means it appears on top */
}
```

Example of z-index

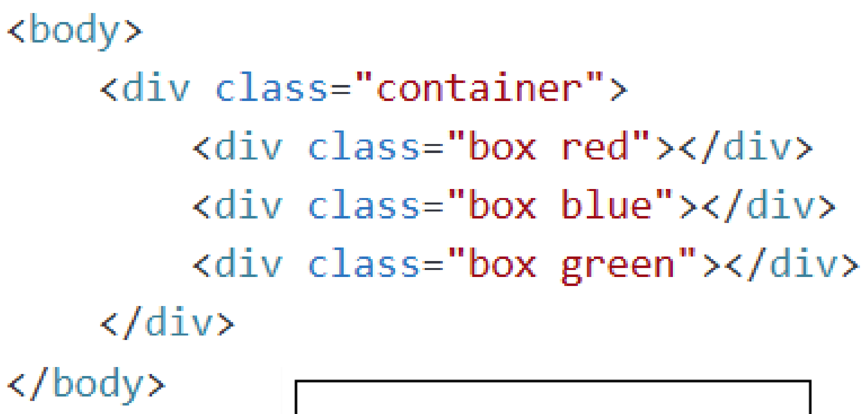
- A red box with z-index: 1 (lower priority)
- A blue box with z-index: 2 (higher priority)
- A green box with z-index: 3 (highest priority, appears on top)



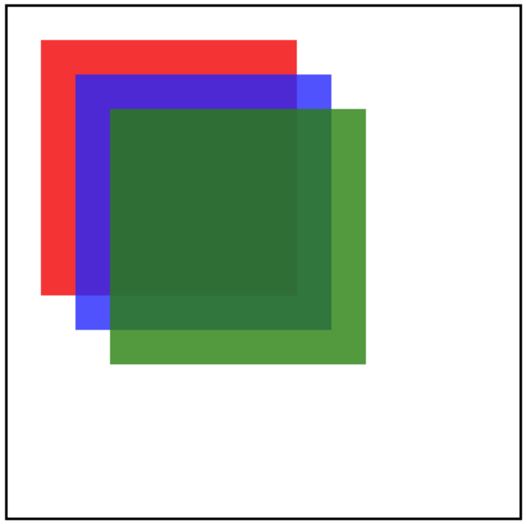
Example



```html
<head>
    <title>Z-Index Example</title>
    <style>
        .container {
            position: relative;
            width: 300px;
            height: 300px;
            border: 2px solid black;
        }

        .box {
            position: absolute;
            width: 150px;
            height: 150px;
            opacity: 0.8;
        }

        .red {
            background-color: red;
            z-index: 1;
            top: 20px;
            left: 20px;
        }

        .blue {
            background-color: blue;
            z-index: 2;
            top: 40px;
            left: 40px;
        }

        .green {
            background-color: green;
            z-index: 3;
            top: 60px;
            left: 60px;
        }
    </style>
</head>
```

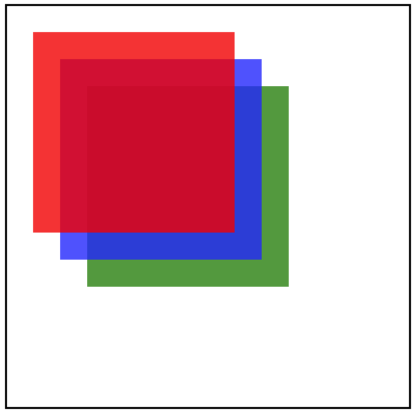


```html
<body>
    <div class="container">
        <div class="box red"></div>
        <div class="box blue"></div>
        <div class="box green"></div>
    </div>
</body>
```



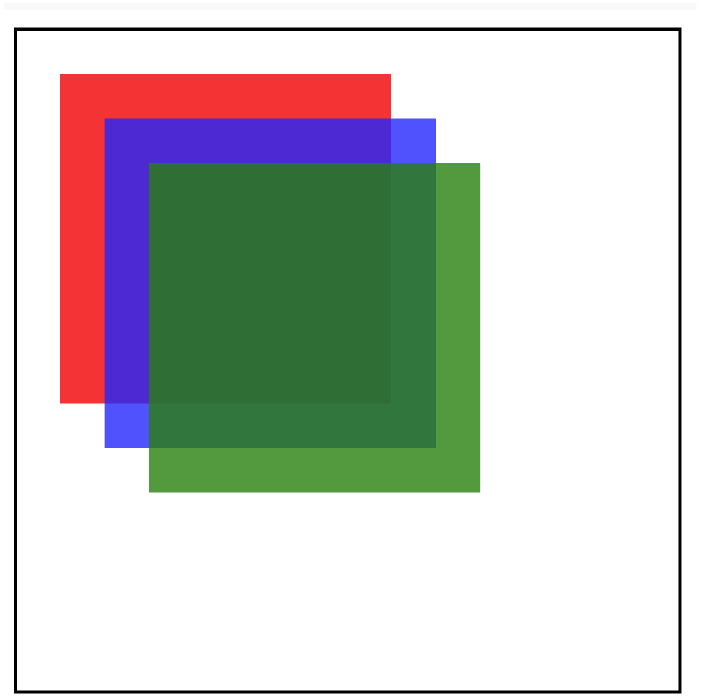
Example

```
.red {
    background-color: red;
    z-index: 3;
    top: 20px;
    left: 20px;
}

.blue {
    background-color: blue;
    z-index: 2;
    top: 40px;
    left: 40px;
}

.green {
    background-color: green;
    z-index: 1;
    top: 60px;
    left: 60px;
}
```

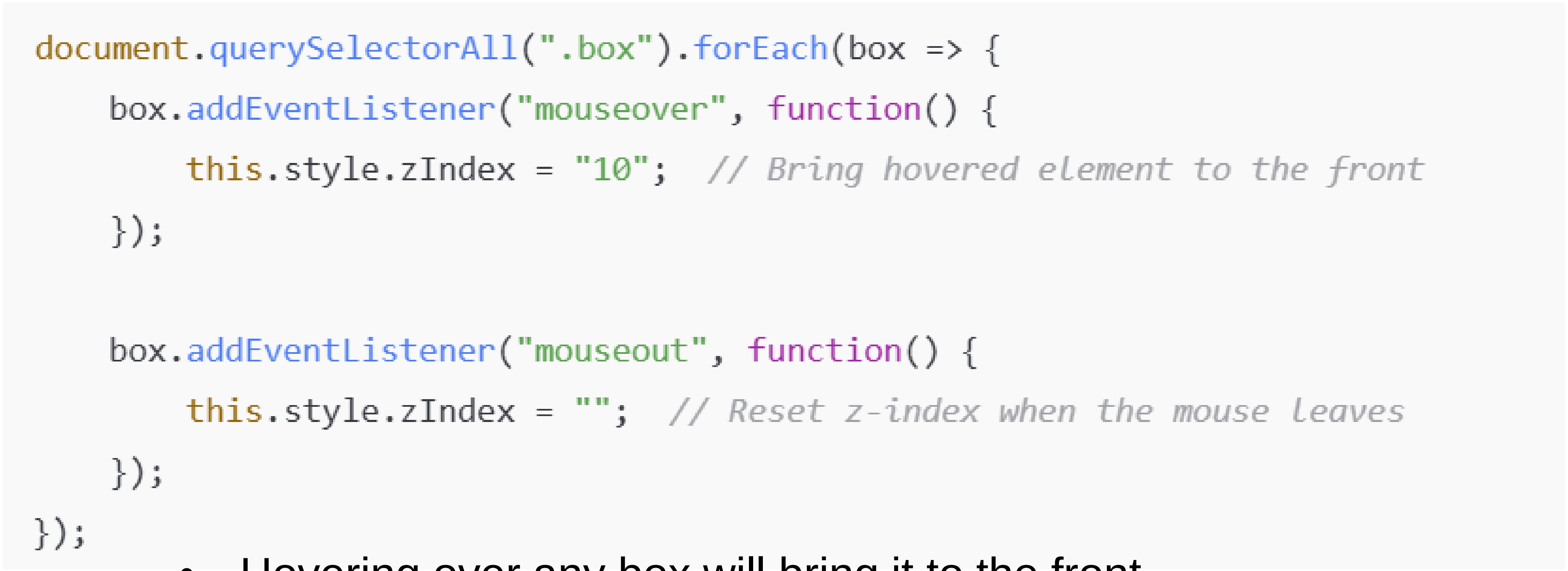


Same Index value

```css
.red {
    background-color: red;
    z-index: 1;
    top: 20px;
    left: 20px;
}

.blue {
    background-color: blue;
    z-index: 1; /* Same as red */
    top: 40px;
    left: 40px;
}

.green {
    background-color: green;
    z-index: 1; /* Same as red and blue
    top: 60px;
    left: 60px;
}
```

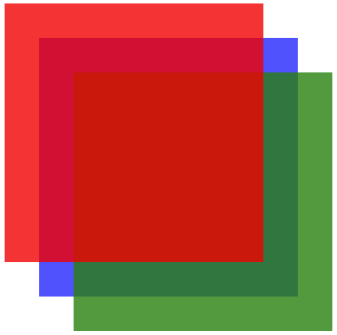


JavaScript to change zIndex



```
document.querySelectorAll(".box").forEach(box => {
    box.addEventListener("mouseover", function() {
        this.style.zIndex = "10";  // Bring hovered element to the front
    });

    box.addEventListener("mouseout", function() {
        this.style.zIndex = "";  // Reset z-index when the mouse leaves
    });
});
```

- Hovering over any box will bring it to the front.
- Once the mouse moves away, it returns to its original order.



Thank You!

JavaScript Canvas

Dr. L.M. Jenila Livingston
Professor
VIT Chennai

What is the Canvas Element?

- The <canvas> element is a part of HTML5.
- It is used to draw graphics on a web page via JavaScript.
- Commonly used for animations, games, and data visualization.

Setting Up Canvas

1. **Add the <canvas> element to your HTML**:

- <canvas id="myCanvas" width="500" height="400"></canvas>

2. **Access the canvas in JavaScript**:

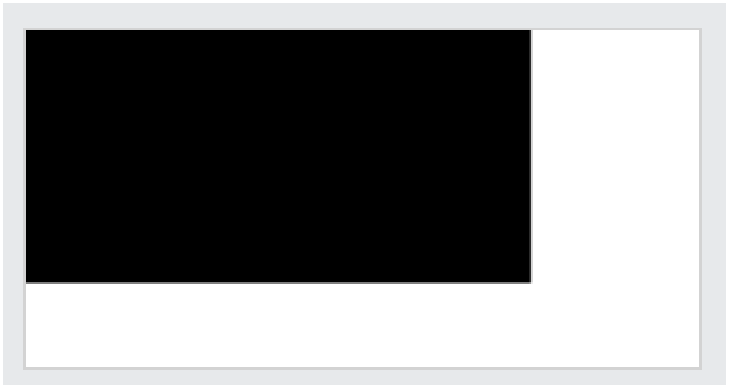
- const canvas = document.getElementById("myCanvas");
- const ctx = canvas.getContext('2d');

beginPath and closePath

- beginPath() ensures that the circle (or any shape) starts a new path, preventing it from connecting to previous shapes and avoiding undesired overlaps.
- Use closePath() to connect the last vertex to the first

Position Inside the canvas

- Define a start-point in position (0,0), and a width and height of 150px and 75px:



Styling and Colors

- fillStyle → Controls the **inside color** of the shape.
- strokeStyle → Controls the **border color** of the shape.
- lineWidth → Adjusts the stroke thickness.
- globalAlpha = value //Controls transparency (0 to 1)
- lineCap = 'butt' | 'round' | 'square' - Defines line end styles –
  - butt (square) ends exactly at the end point (default)/roundedcap/squarecap
- lineJoin = 'bevel' | 'round' | 'miter' - Defines line joining styles (where two lines meet)
  - straight edge/ rounded corner/ sharp corner (default)

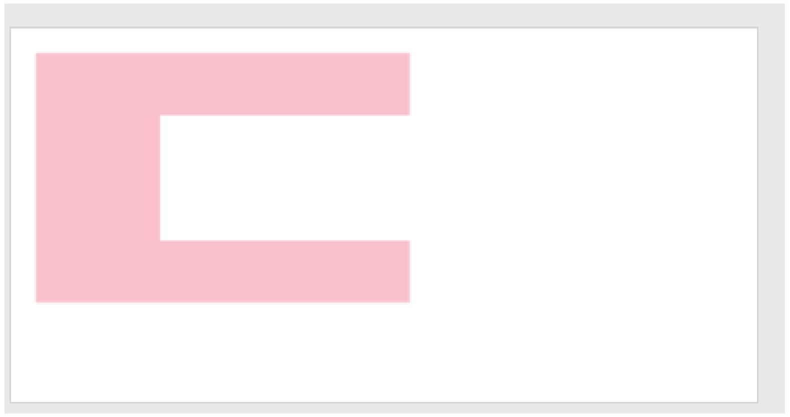
Drawing a Rectangle

- ctx.fillStyle = "green";
- ctx.strokeStyle = "blue";
- ctx.fill**Rect**(50, 50, 200, 100); //filled rectangle
- ctx.strokeRect(50, 50, 200, 100) //rectangle's outline
- ctx.clearRect(50, 50, 50, 25) // clears a specific rectangular area

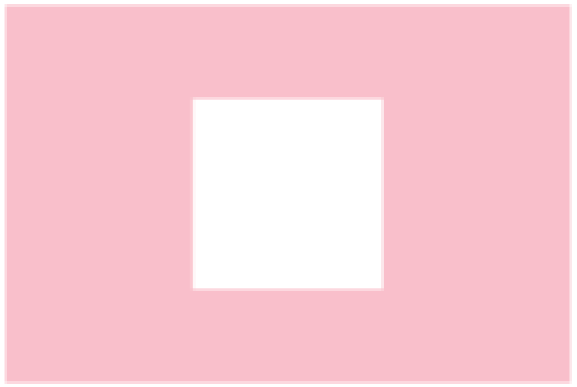
Parameter	Description
x, y	Top-left corner of the rectanglew
w, h	Width and height of the rectangle (same for a square)
ctx.fillRect(x, y, w, h)	Draws and fills the rectangle directly
ctx.strokeRect(x, y, w, h)	Draws the rectangles's outline only

Rectangle

```
<script>
const canvas = document.getElementById("myCanvas");
const ctx = canvas.getContext("2d");

ctx.fillStyle = "pink";
ctx.fillRect(10,10, 150,100);

ctx.clearRect(60,35, 150,50);
</script>
```



```
ctx.fillStyle = "pink";
ctx.fillRect(10,10, 150,100);

ctx.clearRect(60,35, 50,50);
```



Drawing a Circle

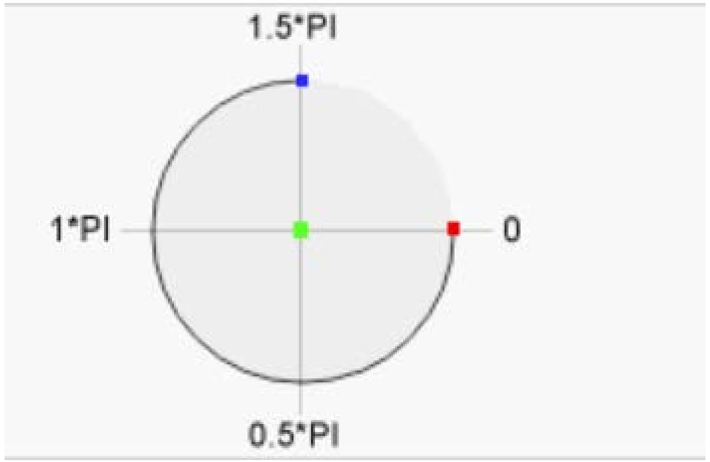
- ctx.beginPath();
- ctx.**arc**(150, 150, 50, 0, **Math.PI * 2**); //full circle (0 to 2π)
- ctx.arc(150, 100, 50, 0, **Math.PI**); // Half-circle (0 to π)
- ctx.fill();

Syntax:

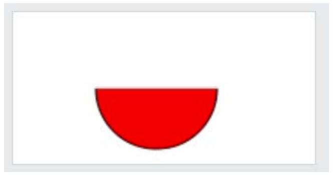
ctx.arc(x, y, radius, startAngle, endAngle, counterclockwise)

Parameter	Description
x, y	Center of the circle
radius	Radius of the circle
startAngle, endAngle	Usually 0 to 2π for a full circle
ctx.beginPath()	Starts a new path to prevent overlap with other shapes
ctx.arc()	Defines the circle's path
counterclockwise	*(Optional)* indicates whether the arc should be drawn counterclockwise (true) or clockwise (*default:* false).

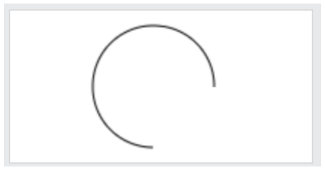
Angle Position of Arc



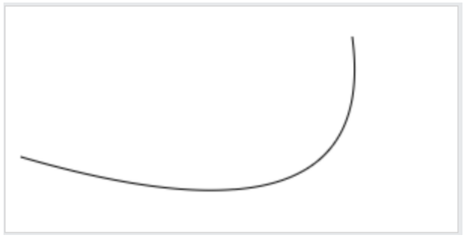
```
ctx.arc(95, 50, 40, 0, Math.PI);
```



```
ctx.arc(95, 50, 40, 0, 0.5 * Math.PI, true);
//true-counter clockwise
```



Other curves

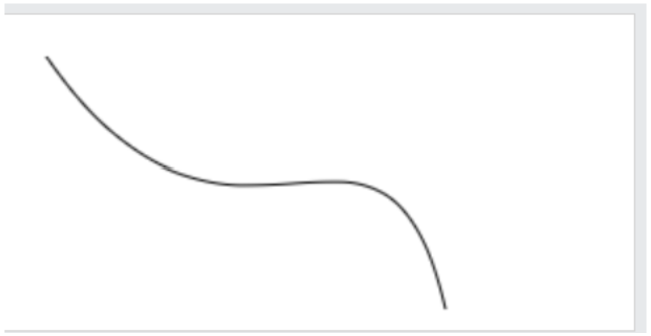


-
  ```
  ctx.moveTo(10, 100);
  ctx.quadraticCurveTo(250, 170, 230, 20);
  ```

  This **quadratic Bezier** curve begins at the point specified by moveTo(): (10, 100). The control point is placed at (250, 170). The curve ends at (230, 20):

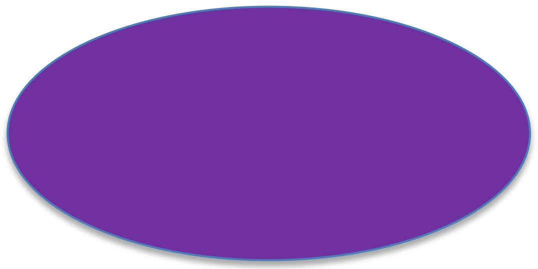
-
  ```
  ctx.moveTo(20, 20);
  ctx.bezierCurveTo(110, 150, 180, 10, 210, 140);
  ```

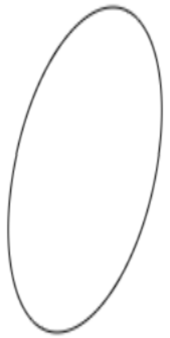
This **cubic Bezier curve** begins at the point specified by moveTo(): (20, 20). The first control point is placed at (110, 150). The second control point is placed at (180, 10). The curve ends at (210, 140):



Drawing an Ellipse

- ctx.beginPath();
- ctx.ellipse(200, 100, 80, 40, 0, 0, Math.PI * 2);
- ctx.fillStyle = 'purple';
- ctx.fill();





With rotation angle

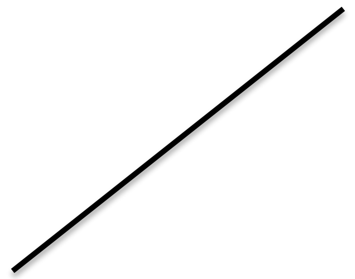
Syntax:

- ctx.ellipse(x, y, radiusX, radiusY, rotation, startAngle, endAngle, counterclockwise)

Parameter	Description
x, y	Center of the ellipse
radiusX, radiusY	Horizontal and vertical radii
rotation	Rotation of the ellipse in radians
startAngle, endAngle	Starting and ending angles in radians

Drawing a Line

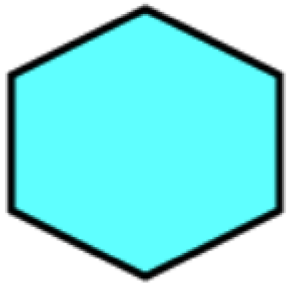
```
ctx.beginPath();
ctx.moveTo(50, 50);
ctx.lineTo(200, 200);
ctx.strokeStyle = 'black';
ctx.lineWidth = 2;
ctx.lineCap = “square";
ctx.stroke();
```



Parameter	Description
moveTo(x, y)	Moves the starting point of the line
lineTo(x, y)	Draws a line to the specified endpoint.
ctx.stroke():	Renders the line
strokeStyle	Color of the line
lineWidth	Thickness of the line
lineCap	The lineCap property defines the cap style of the line ("butt“ (default), "round" or "square").

Drawing a Hexagon

```
ctx.beginPath();
ctx.moveTo(150, 50); // First point
ctx.lineTo(200, 75); // Second point
ctx.lineTo(200, 125); // Third point
ctx.lineTo(150, 150); // Fourth point
ctx.lineTo(100, 125); // Fifth point
ctx.lineTo(100, 75); // Sixth point
ctx.closePath();
ctx.fillStyle = 'cyan';
ctx.fill();
```



Use fill() or stroke()

Stroke(): Draws the line (from the start point, through the sub-points and to the end-point). The default stroke color is black.

Fill() Fills the **entire shape** with a solid color. Default color is black.

Drawing a Hexagon using trigonometric functions

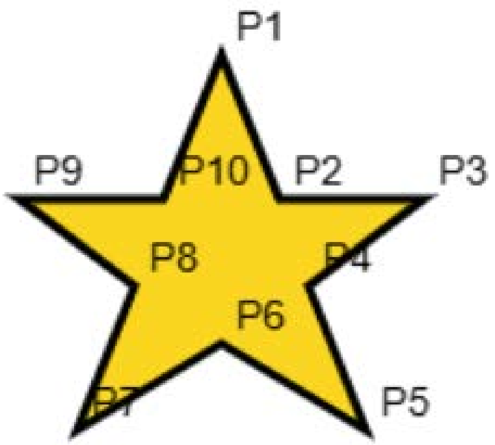
```
ctx.beginPath();
for (let i = 0; i < 6; i++) {
  ctx.lineTo(
    150 + 100 * Math.cos((Math.PI / 3) * i),
    150 + 100 * Math.sin((Math.PI / 3) * i)
  );
}
ctx.closePath();
ctx.fillStyle = 'cyan';
ctx.fill();
```

A hexagon is drawn using six points equally spaced around a circle.
Trigonometric functions calculate the positions of the vertices

Parameter	Description
Number of sides	6 for a hexagon
Radius	Distance from the center to any vertex
Angles	Calculated using (2 * PI) / Number of sides
Trigonometry	Used to calculate each vertex position

Drawing a Star

```
ctx.beginPath();
ctx.moveTo(150, 50); // First outer point
ctx.lineTo(170, 100); // First inner point
ctx.lineTo(220, 100); // Second outer point
ctx.lineTo(180, 130); // Second inner point
ctx.lineTo(200, 180); // Third outer point
ctx.lineTo(150, 150); // Third inner point
ctx.lineTo(100, 180); // Fourth outer point
ctx.lineTo(120, 130); // Fourth inner point
ctx.lineTo(80, 100); // Fifth outer point
ctx.lineTo(130, 100); // Fifth inner point
ctx.closePath();
ctx.fillStyle = 'gold';
ctx.fill();
```



Drawing a Star using trigonometric functions

```
ctx.beginPath();
for (let i = 0; i < 5; i++) {
  ctx.lineTo(
    150 + 100 * Math.cos((18 + i * 72) * Math.PI / 180),
    150 - 100 * Math.sin((18 + i * 72) * Math.PI / 180)
  );
  ctx.lineTo(
    150 + 50 * Math.cos((54 + i * 72) * Math.PI / 180),
    150 - 50 * Math.sin((54 + i * 72) * Math.PI / 180)
  );
}
ctx.closePath();
ctx.fillStyle = 'gold';
ctx.fill();
```

- The star is drawn using alternating points for the outer and inner radii.
- Trigonometric functions calculate the positions of points.

Parameter	Description
Outer radius	Distance from the center to the outer points
Inner radius	Distance from the center to the inner points
Angles	Calculated using trigonometric functions for each point
Number of points	Total number of alternating outer and inner points

Adding Text

Set the font and color:

- ctx.font = '20px Arial';
- ctx.fillStyle = 'black';

Hello World
Hello World

Draw the text:

- ctx.**fillText**('Hello World', 100, 100);//filled text
- ctx.**strokeText**('Hello World', x, y) //outlined text

Text Alignment:

- textAlign = 'left' | 'right' | 'center' - Aligns text horizontally
- textBaseline = 'top' | 'middle' | 'bottom' - Aligns text vertically

Linear Gradient

The `createLinearGradient()` method is used to define a linear gradient.

- A linear gradient changes color along a linear pattern (horizontally/vertically/diagonally).
- The gradient object requires two or more color stops.
- The `addColorStop()` method specifies the color stops, and its position along the gradient. The positions can be anywhere between 0 and 1.
  Syntax: gradient.addColorStop(position, color); 0-start color 1- end color
- To use the gradient, assign it to the `fillStyle` or `strokeStyle` property, then draw the shape

```
// Create linear gradient
const grad=ctx.createLinearGradient(0,0, 280,0);
grad.addColorStop(0, "lightblue");
grad.addColorStop(1, "darkblue");

// Fill rectangle with gradient
ctx.fillStyle = grad;

ctx.fillRect(10,10, 280,130);
ctx.strokeRect(10,10,280,130);

const grad=ctx.createLinearGradient(0,0, 280,0);
grad.addColorStop(0, "lightblue");
grad.addColorStop(0.5, "purple");
grad.addColorStop(1, "darkblue");
```

```
(0,0) -------------> (280,0)
        Horizontal Gradient
```

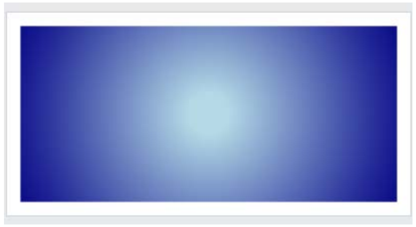
```
(0,0)
  |
  |
  |    Vertical Gradient
  |
  |
(0,280)
```

```
(Light Blue)      (Purple)          (Dark Blue)
0                 0.5                    1
[=============|=============|=============]
```

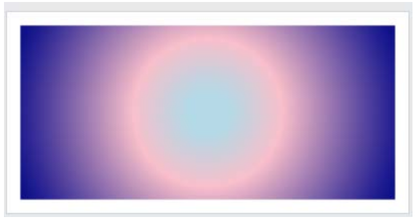
Radial Gradient

- The `createRadialGradient()` method is used to define a radial/circular gradient.
- A radial gradient is defined with two imaginary circles: a start circle and an end circle. The gradient starts with the start circle and moves towards the end circle.

```
// Create radial gradient
const grad=ctx.createRadialGradient(150,75,15,150,75,150);
grad.addColorStop(0,"lightblue");
grad.addColorStop(1,"darkblue");

// Fill rectangle with gradient
ctx.fillStyle = grad;
ctx.fillRect(10,10,280,130);
```

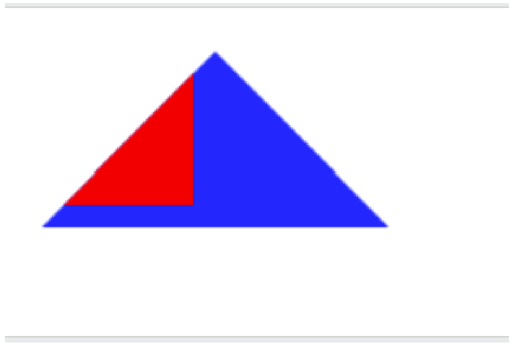


```
const grad=ctx.createRadialGradient(150,75,15,150,75,150);
grad.addColorStop(0,"lightblue");
grad.addColorStop(0.3,"pink");
grad.addColorStop(1,"darkblue");
```



Clipping: Triangular shape

```
<script>
const canvas = document.getElementById("myCanvas");
const ctx = canvas.getContext("2d");

// Create a triangle-shaped clipping region
ctx.beginPath();
ctx.moveTo(100,20);
ctx.lineTo(180,100);
ctx.lineTo(20,100);
ctx.lineTo(100,20);
ctx.clip();

// Draw two rectangles
ctx.fillStyle = "blue";
ctx.fillRect(0, 0, 300, 150);
ctx.fillStyle = "red";
ctx.fillRect(0, 0, 90, 90);
</script>
```



JavaScript Canvas Example Program

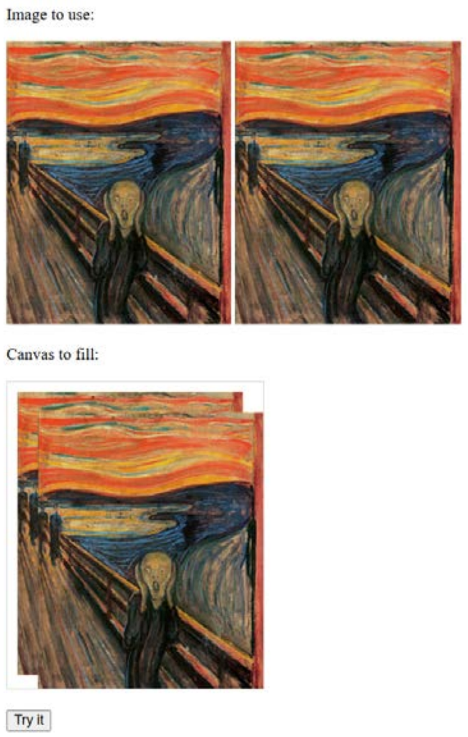
```
<!DOCTYPE html>
<html><head> <title>Canvas Example</title></head>
<body>   <canvas id='myCanvas' width='400' height='300' style='border:1px
solid #000;'></canvas>
  <script>
    const canvas = document.getElementById('myCanvas');
    const ctx = canvas.getContext('2d');
    // Draw a rectangle
    ctx.fillStyle = 'blue';
    ctx.fillRect(50, 50, 150, 100);
    // Add text
    ctx.font = '20px Arial';
    ctx.fillStyle = 'green';
    ctx.fillText('Hello Canvas!', 100, 250);
  </script></body></html>
```

Image Handling

- **drawImage**(image, x, y, width, height) - Draws an image
- **getImageData**(x, y, width, height) - Retrieves pixel data
- **putImageData**(imageData, x, y) - Places pixel data back onto the canvas

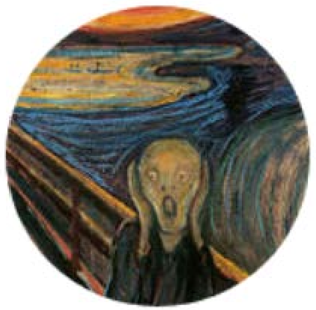
Image

```
<body>

<p>Image to use:</p>
<img id="scream" src="img_the_scream.jpg" alt="The Scream" width="220" height="277">
<img id="scream" src="img_the_scream.jpg" alt="The Scream" width="220" height="277">

<p>Canvas to fill:</p>
<canvas id="myCanvas" width="250" height="300"
style="border:1px solid #d3d3d3;">
Your browser does not support the HTML canvas tag.</canvas>

<p><button onclick="myCanvas()">Try it</button></p>

<script>
function myCanvas() {
  var c = document.getElementById("myCanvas");
  var ctx = c.getContext("2d");
  var img = document.getElementById("scream");
  ctx.drawImage(img,10,10);
  var img = document.getElementById("scream");
  ctx.drawImage(img,30,30);
}
</script>

</body>
```



Clipping: Image

```
<script>
const canvas = document.getElementById("myCanvas");
const ctx = canvas.getContext("2d");
const image = document.getElementById("scream");

image.addEventListener("load", (e) => {
  // Create a circular clipping region
  ctx.beginPath();
  ctx.arc(110, 145, 75, 0, Math.PI * 2);
  ctx.clip();
  // Draw image onto canvas
  ctx.drawImage(image, 0, 0);
});
</script>
```



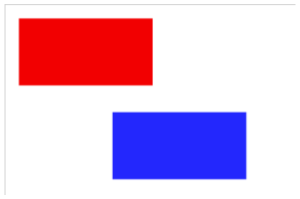
Transformations

- translate(x, y) - Moves the origin to (x, y)
- rotate(angle) - Rotates the canvas (radians)
- scale(x, y) - Scales horizontally & vertically
- save() - Saves the current state
- restore() - Restores the last saved state

```
// Restore original canvas state
ctx.resetTransform();
```

translate

```
<script>
const canvas = document.getElementById("myCanvas");
const ctx = canvas.getContext("2d");

ctx.fillStyle = "red";
ctx.fillRect(10, 10, 100, 50);

ctx.translate(70, 70);

ctx.fillStyle = "blue";
ctx.fillRect(10, 10, 100, 50);
</script>
```

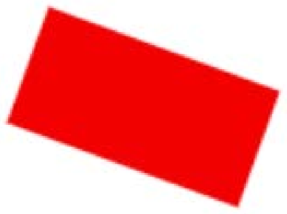


Draws a blue rectangle at (10,10), but due to the translation, it actually appears at (10+70, 10+70) = (80,80).

Output:
- A **red rectangle** at (10,10).
- A **blue rectangle** at (80,80), due to translate(70,70)

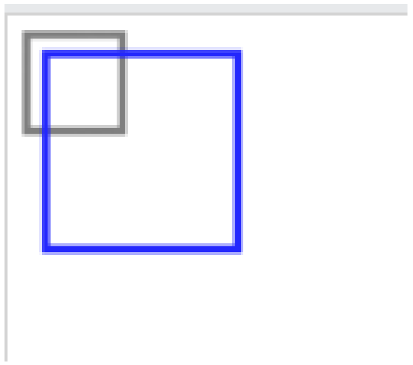
rotate

```
<script>
const canvas =
document.getElementById("myCanvas");
const ctx = canvas.getContext("2d");

ctx.rotate((Math.PI/180)*20);

ctx.fillStyle = "red";
ctx.fillRect(50, 10, 100, 50);
</script>
```



scale

```
<script>
const canvas = document.getElementById("myCanvas");
const ctx = canvas.getContext("2d");

ctx.strokeRect(5, 5, 25, 25);

ctx.scale(2, 2);  // From this point on, everything is twice as large.

ctx.strokeStyle = "blue";
// It actually draws at (10,10) with size 50×50 because all values are multiplied by 2.
ctx.strokeRect(5, 5, 25, 25);
</script>
```



Uniform Scaling

- ctx.scale(2,2) → **Doubles everything** (width and height).

Non-Uniform Scaling

- ctx.scale(2,1) → **Stretches objects horizontally** but keeps the **height unchanged**.
- ctx.scale(1,2) → **Stretches objects vertically** but keeps the **width unchanged**

Animation

Steps to Create an Animation in Canvas

1. **Clear the previous frame** using **ctx.clearRect().**
2. **Update object positions** (e.g., **x & y** coordinates).
3. **Redraw the object** at the new position.
4. **Repeat the process** using **requestAnimationFrame()**.

Animate a ball

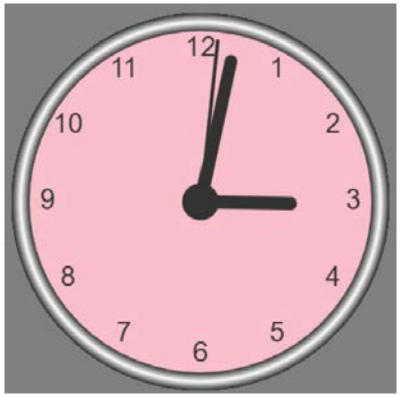
```
 let x = 50, y = 200;   // Ball's initial position
let dx = 3, dy = 2;    // Speed of movement
let radius = 20;       // Ball radius
function drawBall() {
ctx.clearRect(0, 0, canvas.width, canvas.height); // Clear the canvas

 // Draw the ball
ctx.beginPath();
ctx.arc(x, y, radius, 0, Math.PI * 2);
ctx.fillStyle = "red";
ctx.fill();

// Update position
x += dx; y += dy;

 // Bounce off the walls
if (x + radius > canvas.width || x - radius < 0) dx = -dx;
if (y + radius > canvas.height || y - radius < 0) dy = -dy;

requestAnimationFrame(drawBall); // Repeat animation
}
drawBall(); // Start animation
```

- `x + radius > canvas.width` → Ball **hits the right wall**.
- `x - radius < 0` → Ball **hits the left wall**.
- `dx = -dx;` → Reverses **horizontal direction**.
- `y + radius > canvas.height` → Ball **hits the bottom**.
- `y - radius < 0` → Ball **hits the top**.
- `dy = -dy;` → Reverses **vertical direction**.

Clock

```
<canvas id="canvas" width="400" height="400" style="background-
color:gray"> Sorry, your browser does not support canvas.
</canvas>
<script>

const canvas = document.getElementById("canvas");
const ctx = canvas.getContext("2d");
let radius = canvas.height / 2;
ctx.translate(radius, radius);
radius = radius * 0.90

//Calls drawClock() every 1000ms (1 second) using setInterval(),
setInterval(drawClock, 1000);

function drawClock() {
  drawFace(ctx, radius);
  drawNumbers(ctx, radius);
  drawTime(ctx, radius);
}
```

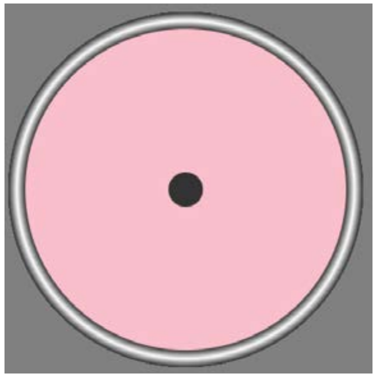
12
1
2
3
4
5
6
7
8
9
10
11



- **drawClock()** is the **main function** that draws the clock.
- It calls **drawFace(ctx, radius)** → Draws the circular clock face.
- It calls **drawNumbers(ctx, radius)** → Draws the numbers (1 to 12) on the clock.
- Calls **drawTime(ctx, radius)** → Draws the **moving clock hands**.

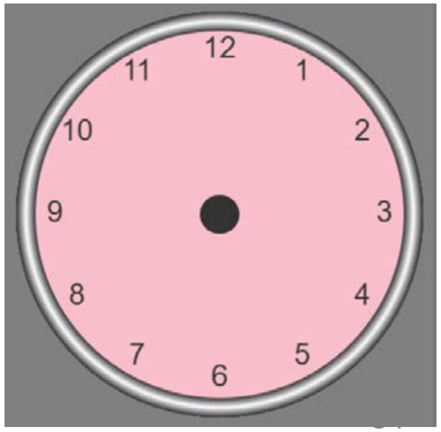
Clock Face

```
function drawFace(ctx, radius) {
  const grad = ctx.createRadialGradient(0, 0, radius * 0.95, 0, 0, radius * 1.05);
  grad.addColorStop(0, '#333');   // Dark gray outer ring
  grad.addColorStop(0.5, 'pink'); // Pink middle area
  grad.addColorStop(1, '#333');   // Dark gray inner ring

  ctx.beginPath();
  ctx.arc(0, 0, radius, 0, 2 * Math.PI);  // Draws a full circle
  ctx.fillStyle = 'white';  // Sets fill color to white
  ctx.fill();

  ctx.strokeStyle = grad;   // Uses gradient for border 3d appearance
  ctx.lineWidth = radius * 0.1;  // Border thickness
  ctx.stroke();

  ctx.beginPath();
  ctx.arc(0, 0, radius * 0.1, 0, 2 * Math.PI);  // Draws a small circle in the center
  ctx.fillStyle = '#333';  // Sets center dot color
  ctx.fill();
}
```



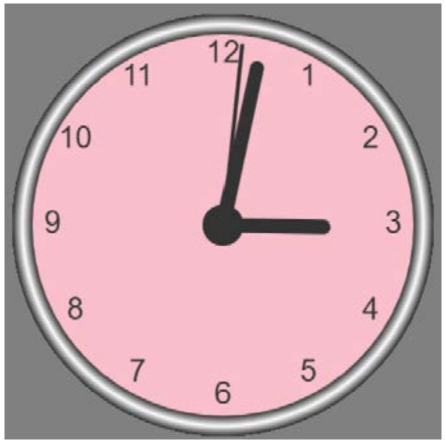
Clock Numbers

```
function drawNumbers(ctx, radius) {
  ctx.font = radius * 0.15 + "px arial"; // Sets font size relative to clock size
  ctx.textBaseline = "middle";
  ctx.textAlign = "center";
  for(let num = 1; num < 13; num++){
    let ang = num * Math.PI / 6; // Convert number position to angle
    ctx.rotate(ang); // Rotate canvas to the correct position
    ctx.translate(0, -radius * 0.85); // Move text outward from the center
    ctx.rotate(-ang); // Rotate back to upright text position
    ctx.fillText(num.toString(), 0, 0); // Draw number
    ctx.rotate(ang); // Restore rotation
    ctx.translate(0, radius * 0.85); // Move text back to original position
    ctx.rotate(-ang);}
}
```



Draw Hands

```
function drawTime(ctx, radius) {
  const now = new Date();
  let hour = now.getHours();
  let minute = now.getMinutes();
  let second = now.getSeconds();
  //hour hand position (Converted time to angles (radians) to move correctly)
  // Calls drawHand(ctx, pos, length, width); for each hand.
  hour = hour%12;
  hour = (hour*Math.PI/6)+(minute*Math.PI/(6*60))+(second*Math.PI/(360*60));
  drawHand(ctx, hour, radius*0.5, radius*0.07);
  //minute hand position
  minute = (minute*Math.PI/30)+(second*Math.PI/(30*60));
  drawHand(ctx, minute, radius*0.8, radius*0.07);
  // second hand position
  second = (second*Math.PI/30);
  drawHand(ctx, second, radius*0.9, radius*0.02); }

function drawHand(ctx, pos, length, width) {
  ctx.beginPath();
  ctx.lineWidth = width;
  ctx.lineCap = "round";
  ctx.moveTo(0,0);
  ctx.rotate(pos); // Rotate to the correct position
  ctx.lineTo(0, -length);
  ctx.stroke();
  ctx.rotate(-pos);} // Rotate back to prevent misalignment
```



Start the clock

```
//Start the Clock
const canvas = document.getElementById("canvas");
const ctx = canvas.getContext("2d");
let radius = canvas.height / 2;
ctx.translate(radius, radius);
radius = radius * 0.90

//drawClock();
setInterval(drawClock, 1000);
```

- The setInterval() function is used to repeatedly call a function **at fixed time intervals**.
- Calls the drawClock function **every 1000 milliseconds (1 second) -** to update the clock hands **every second**.
- Used for **real-time updates**, such as a **clock, animations, or dynamic UI change**

Refer:
https://www.w3schools.com/graphics/canvas_clock_start.asp

Games

Example

Reference:
https://www.w3schools.com/graphics/tryit.asp?filename=trygame_default_gravity

Applications of Canvas

- Creating dynamic charts and graphs
- Designing animations and games
- Image processing and manipulation
- Data visualization and dashboards

Thank You!

Charts and Graphs: Plotly

Dr. L.M. Jenila Livingston
Professor
VIT Chennai

Plotly library

<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>

- This script **loads the Plotly library** into your webpage.

- It enables you to create interactive charts and graphs using **JavaScript**.

- **Plotly.js** is a charting library that comes with many different chart types:
  - Horizontal and Vertical Bar Charts
  - Pie and Donut Charts
  - Line Charts
  - Scatter and Bubble Plots
  - Equation Plots
  - 3D Charts
  - Statistical Graphs
  - etc.

Plotly Offline

How to Use Plotly Without the Internet (Offline Mode)
If you **don't have internet access**, you can **download the Plotly library** and use it locally.

Steps to Use Plotly Offline

1. **Download Plotly Library**
   - Go to Plotly GitHub. https://github.com/plotly/plotly.js/releases
   - Download the latest plotly-latest.min.js file.
2. **Save It Locally**
   - Place the downloaded file in your project folder (e.g., libs/plotly-latest.min.js).
3. **Update Your HTML File**

   Instead of using the online CDN, **use the local file**:

```
<script src="libs/plotly-latest.min.js"></script>
```

Using Plotly with NPM

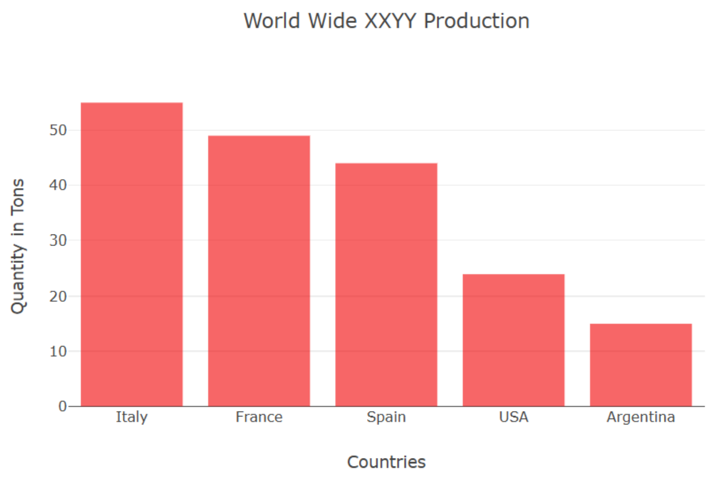
- If you are using **Node.js**, you can install Plotly via NPM:
- npm install plotly.js-dist
- Then, import it in your JavaScript file:
- const Plotly = require('plotly.js-dist');

Steps to draw Plotly graphs/Charts

1. **Import Plotly Library**: Use the Plotly script in your HTML file.
2. **Specify Data Points**: Define the data for the plot.
3. **Specify Title and Layout**: Configure the title and axis labels.
4. **Map Data and Layout**: Use Plotly.newPlot() to generate the chart.

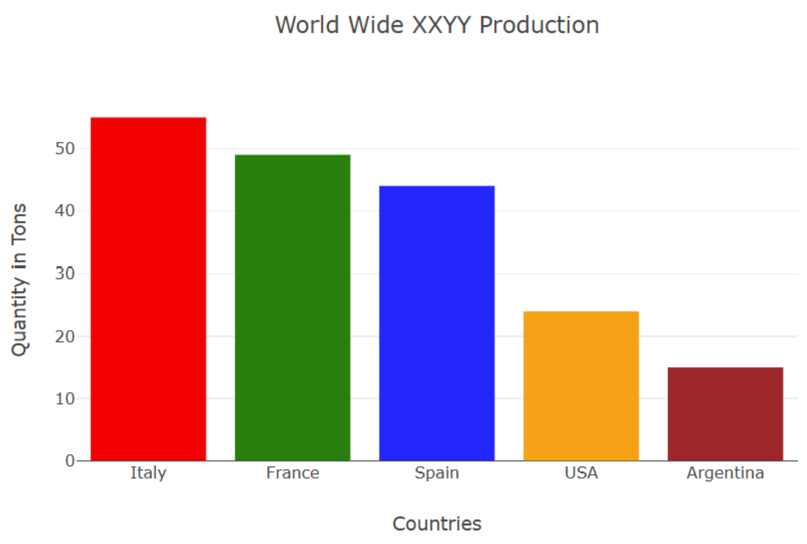
Vertical Bar chart

```
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
<body>
<div id="myPlot" style="width:100%;max-width:700px"></div><script>
const xArray = ["Italy","France","Spain","USA","India"];
const yArray = [55, 49, 44, 24, 15];

const data = [{
  x: xArray,
  y: yArray,
  type: "bar",
  orientation:"v",
  marker: {color:"rgba(255,0,0,0.6)"}
}];

const layout = {
  xaxis: { title: "Countries" },
  yaxis: { title: "Quantity in Tons" },
  title:"World Wide XXYY Production“
};
Plotly.newPlot("myPlot", data, layout);
</script></body>
```



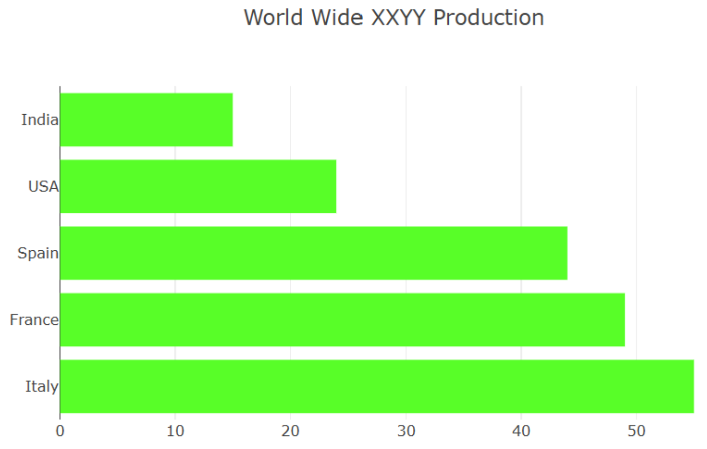
Vertical Bar chart

```
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
<body>
<div id="myPlot" style="width:100%;max-width:700px"></div><script>
const xArray = ["Italy","France","Spain","USA","India"];
const yArray = [55, 49, 44, 24, 15];
const barColors = ["red", "green","blue","orange","brown"];

const data = [{
  x: xArray,
  y: yArray,
  type: "bar",
  orientation:"v",
  marker: { color: barColors }
}];

const layout = {
  xaxis: { title: "Countries" },
  yaxis: { title: "Quantity in Tons" },
  title:"World Wide XXYY Production"
};
Plotly.newPlot("myPlot", data, layout);
</script></body>
```



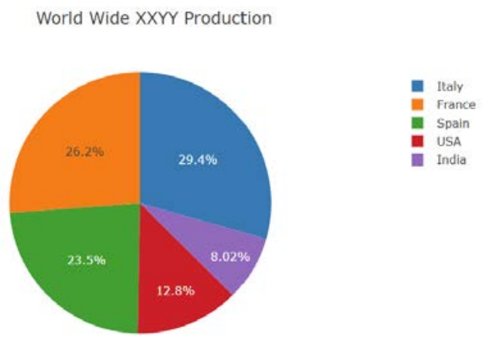
Horizontal Bar chart

```
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
<body>
<div id="myPlot" style="width:100%;max-width:700px"></div><script>
const xArray = ["Italy","France","Spain","USA","India"];
const yArray = [55, 49, 44, 24, 15];

const data = [{
  x: yArray,
  y: xArray,
  type: "bar",
  orientation:“h",
  marker: {color:"rgb(0,255,0)"}
}];

const layout = {title:"World Wide XXYY Production"};

Plotly.newPlot("myPlot", data, layout);
</script>
</body>
```



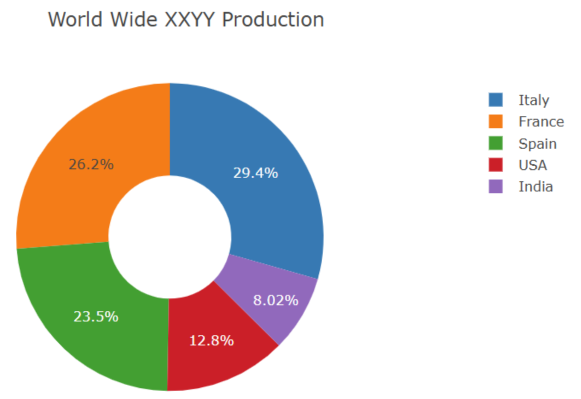
Pie chart

```
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
<body>
<div id="myPlot" style="width:100%;max-width:700px"></div><script>
const xArray = ["Italy","France","Spain","USA","India"];
const yArray = [55, 49, 44, 24, 15];

const data = [{
  labels: xArray,
  values: yArray,
  type: “pie",
}];

const layout = {title:"World Wide XXYY Production"};

Plotly.newPlot("myPlot", data, layout);
</script>
</body>
```



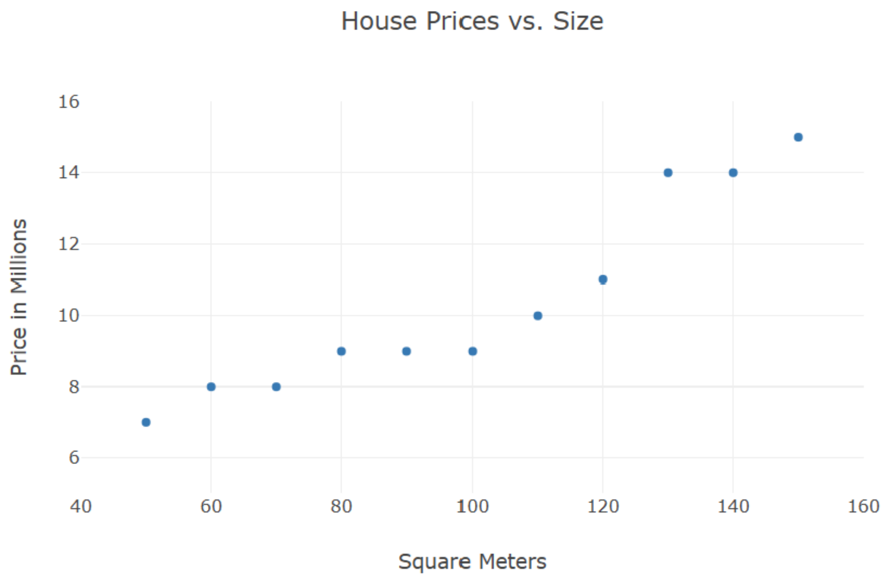
Donut chart

```
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
<body>
<div id="myPlot" style="width:100%;max-width:700px"></div><script>
const xArray = ["Italy","France","Spain","USA","India"];
const yArray = [55, 49, 44, 24, 15];

const data = [{
  labels: xArray,
  values: yArray,
  hole:.4,
  type: “pie",
}];

const layout = {title:"World Wide XXYY Production"};

Plotly.newPlot("myPlot", data, layout);
</script>
</body>
```



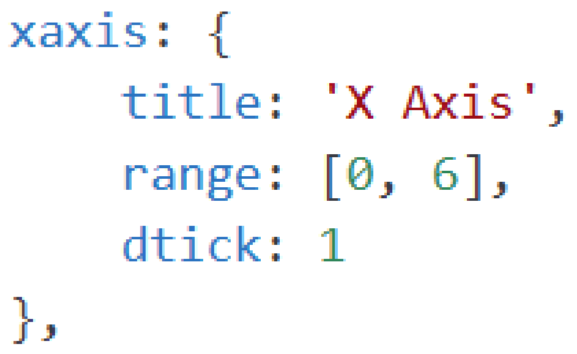
Scatter Plots

```
const xArray = [50,60,70,80,90,100,110,120,130,140,150];
const yArray = [7,8,8,9,9,9,10,11,14,14,15];

// Define Data
const data = [{
  x: xArray,
  y: yArray,
  mode:"markers",
  type:"scatter"
}];

// Define Layout
const layout = {
  xaxis: {range: [40, 160], title: "Square Meters"},
  yaxis: {range: [5, 16], title: "Price in Millions"},
  title: "House Prices vs. Size"
};

Plotly.newPlot("myPlot", data, layout);
```



range - dtick

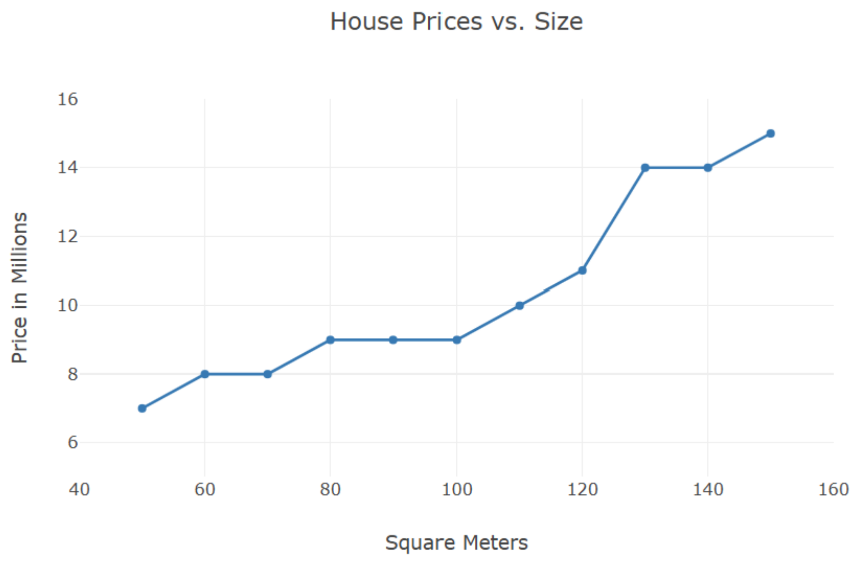
- Dtick Property: By default, **Plotly automatically determines the best tick interval (dtick)** based on the data points range and axis limits.
- The ticks **adjust automatically/ dynamically** (auto scaling behaviour) when zooming in or out (interactive) - **Preferable**
- The axis ticks are **dynamically adjusted** to fit the available space.
- If the range is large, Plotly **increases the interval** to avoid overcrowding.
- If the range is small, **more ticks are displayed** for better readability.
- **Manual Mode of dtick (not preferable):**



```
xaxis: {
    title: 'X Axis',
    range: [0, 6],
    dtick: 1
},
```

Line Graph

```
const xArray = [50,60,70,80,90,100,110,120,130,140,150];
const yArray = [7,8,8,9,9,9,10,11,14,14,15];

// Define Data
const data = [{
  x: xArray,
  y: yArray,
  mode:"lines",
  type:"scatter"
}];

// Define Layout
const layout = {
  xaxis: {range: [40, 160], title: "Square Meters"},
  yaxis: {range: [5, 16], title: "Price in Millions"},
  title: "House Prices vs. Size"
};

Plotly.newPlot("myPlot", data, layout);
```



Thank You!