

# Canvas Events

# Window Events

## addEventListener

- ▶ Listen to key presses in the window:  
`window.addEventListener( "keydown", doKeyDown)`
- ▶ “keydown” is the event name
- ▶ doKeyDown is the function we want to call
  - Can be a named or anonymous function

# Function Type Reminder

Named function:

```
function doKeyDown(event) {  
  console.log("Key down: " + event);  
}  
window.addEventListener("keydown", doKeyDown);
```

Anonymous function:

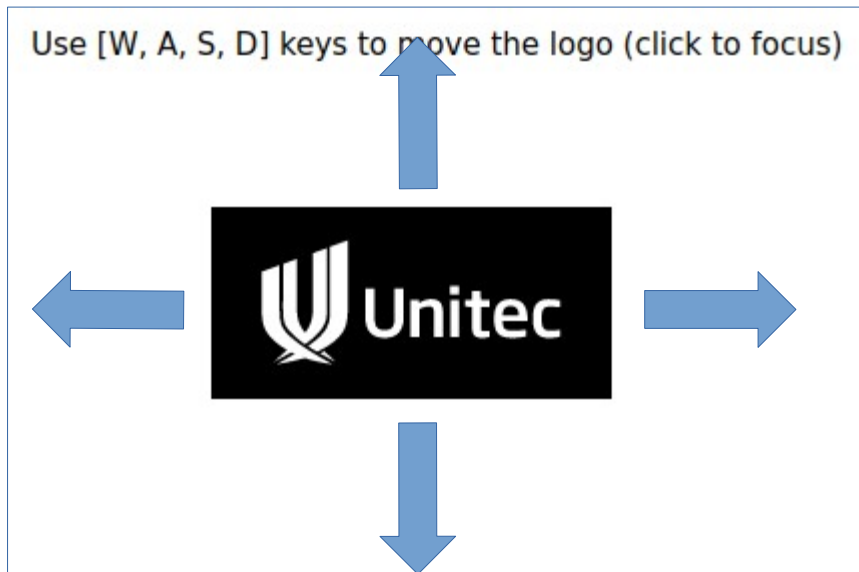
```
window.addEventListener("keydown", (event) => {  
  console.log("Key down: " + event);  
}))
```

# Example:

```
let container = document.getElementById("logoBox");
let logo = container.querySelector("img");

container.addEventListener("keydown", event => {
  doKeyDown(event);
});

let x = 0;
let y = 0;
```



<https://jschollitt.github.io/week7.html>

```
function moveLogo(x, y) {
  logo.style.left = x + 'px';
  logo.style.top = y + 'px';
}

function doKeyDown(event){
  switch(event.key) {
    case "a": {
      x = x - 10;
      moveLogo(x, y);
      break;
    }
    case "w": {
      y = y - 10;
      moveLogo(x, y);
      break;
    }
    case "d": {
      x = x + 10;
      moveLogo(x, y);
      break;
    }
    case "s": {
      y = y + 10;
      moveLogo(x, y);
      break;
    }
  }
}
```

# Canvas-Events

To make the canvas element accept key input:

- ▶ `var canvas = document.getElementById("myCanvas");  
canvas.addEventListener( "keydown", doKeyDown,  
true);`
- ▶ **Detecting a Key Press**
- ▶ Things like buttons, text boxes, images and hyperlinks have events such as `onClick` attached to them.
- ▶ Unfortunately, the canvas tag doesn't. Instead we use the preferred event triggering method, `addEventListener`. We can write our own events and attach them to the canvas tag. Here's the syntax for the method:
- ▶ `addEventListener( event_type, event_handler,  
capture )`

# element.addEventListener(*event*, *function*, *useCapture*)

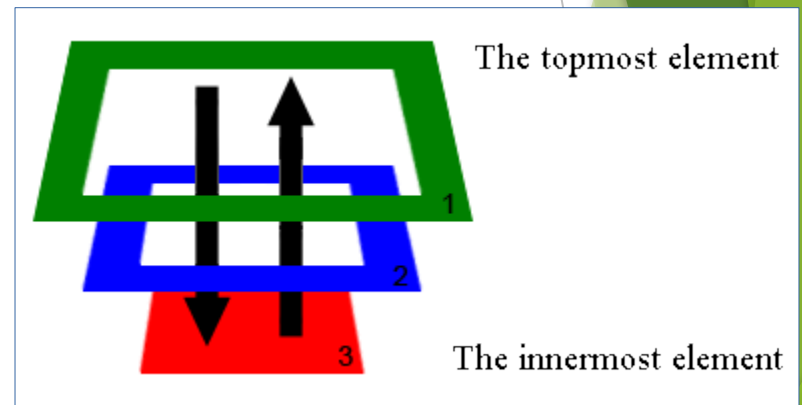
Parameter	Description
<i>event</i>	<p>Required. A String that specifies the name of the event.</p> <p><b>Note:</b> Do not use the "on" prefix. For example, use "click" instead of "onclick".</p> <p>For a list of all HTML DOM events, look at the complete <a href="#">HTML DOM Event Object Reference</a>.</p>
<i>function</i>	<p>Required. Specifies the function to run when the event occurs.</p> <p>When the event occurs, an event object is passed to the function as the first parameter. The type of the event object depends on the specified event. For example, the "click" event belongs to the MouseEvent object.</p>
<i>useCapture</i>	<ul style="list-style-type: none"><li>•Optional. A Boolean value that specifies whether the event should be executed in the capturing or in the bubbling phase.</li></ul> <p>Possible values:true - The event handler is executed in the capturing phase</p> <ul style="list-style-type: none"><li>•false- Default. The event handler is executed in the bubbling phase</li></ul>

# Event Propagation

- ▶ In all browsers, except IE<9, there are two stages of event processing.
- ▶ The event first goes down - that's called *capturing*, and then *bubbles* up. This behaviour is standardized in W3C specification.

According to this model, the event:

1. Captures down - through 1 -> 2 -> 3.
2. Bubbles up - through 3 -> 2 -> 1.



`elem.addEventListener( type, handler, phase )`

**phase = true**

The handler is set on the capturing phase.

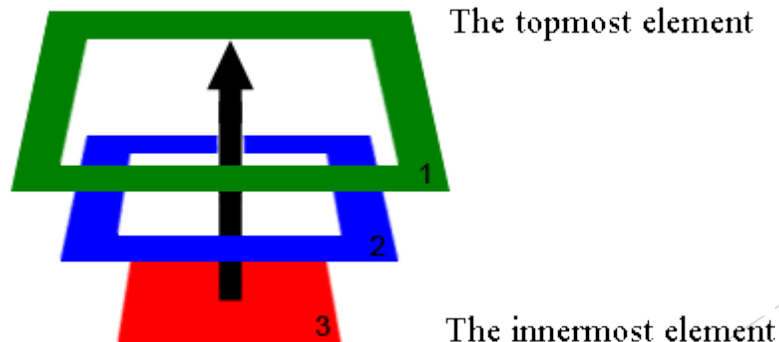
**phase = false**

The handler is set on the bubbling phase.

# Bubbling

- ▶ Default behaviour of events that support propagating.
- ▶ The main principle of bubbling states:  
**After an event triggers on the deepest possible element, it then triggers on parents in nesting order.**
- ▶ Test for bubble support with `event.bubbles` readonly boolean
- ▶ For example:

<http://javascript.info/tutorial/bubbling-and-capturing>

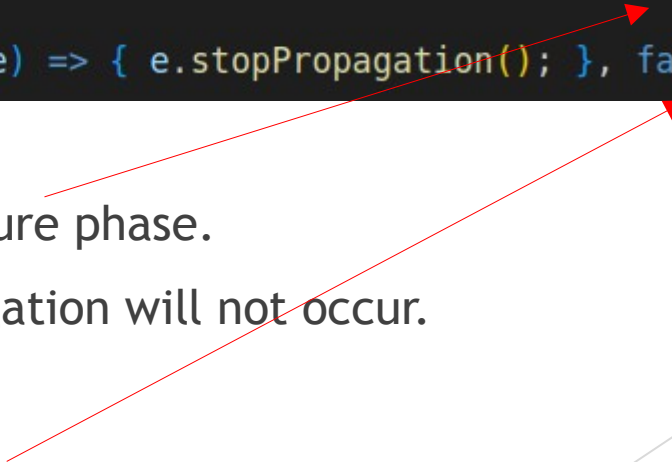




# Stop Event Propagation

- ▶ Capture and Bubbling can be cancelled
- ▶ `event.stopPropagation()`

```
element.addEventListener("click", (e) => { e.stopPropagation(); }, true);  
element.addEventListener("click", (e) => { e.stopPropagation(); }, false);
```



Line 1 will run in the capture phase.

Further **downward** propagation will not occur.

Line 2 will run in the bubble phase.

Further **upward** propagation will not occur

# Stop Event Propagation

- ▶ We can be selective about allowed propagation:
- ▶ `event.preventDefault()`
  - Allow our event code to run, but stop web browser events from running
  - Useful for stopping page scrolling when using arrow keys for input, etc.

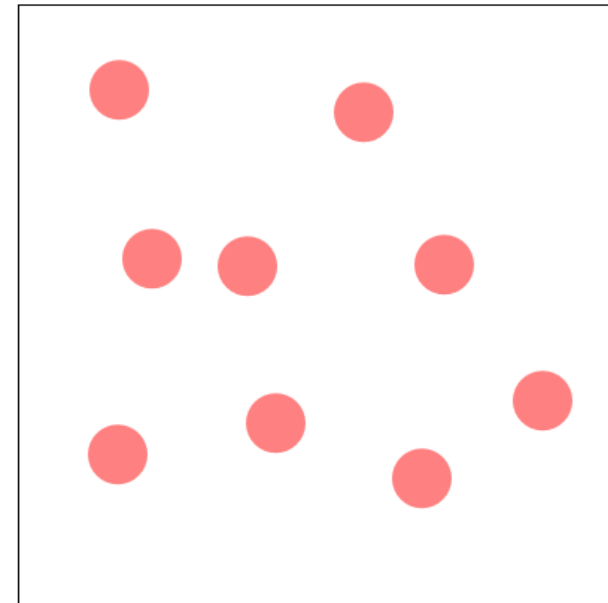
`event.stopImmediatePropagation()`

- Stops all listeners for an event for all elements

# Mouse click canvas

```
function drawOnEvent() {  
  let canvas = document.getElementById("w7canvas4");  
  let ctx = canvas.getContext("2d");  
  canvas.addEventListener("mousedown", click, false);  
  
  function click(event) {  
    let rect = canvas.getBoundingClientRect();  
    let x = event.clientX - rect.left;  
    let y = event.clientY - rect.top;  
    draw(x, y);  
  }  
  
  function draw(x, y) {  
    ctx.beginPath();  
    ctx.arc(x, y, 20, 0, 2 * Math.PI, true);  
    ctx.closePath();  
    ctx.fillStyle = "#ff8080";  
    ctx.fill();  
  }  
}  
drawOnEvent();
```

Draw on Mouse Click Event



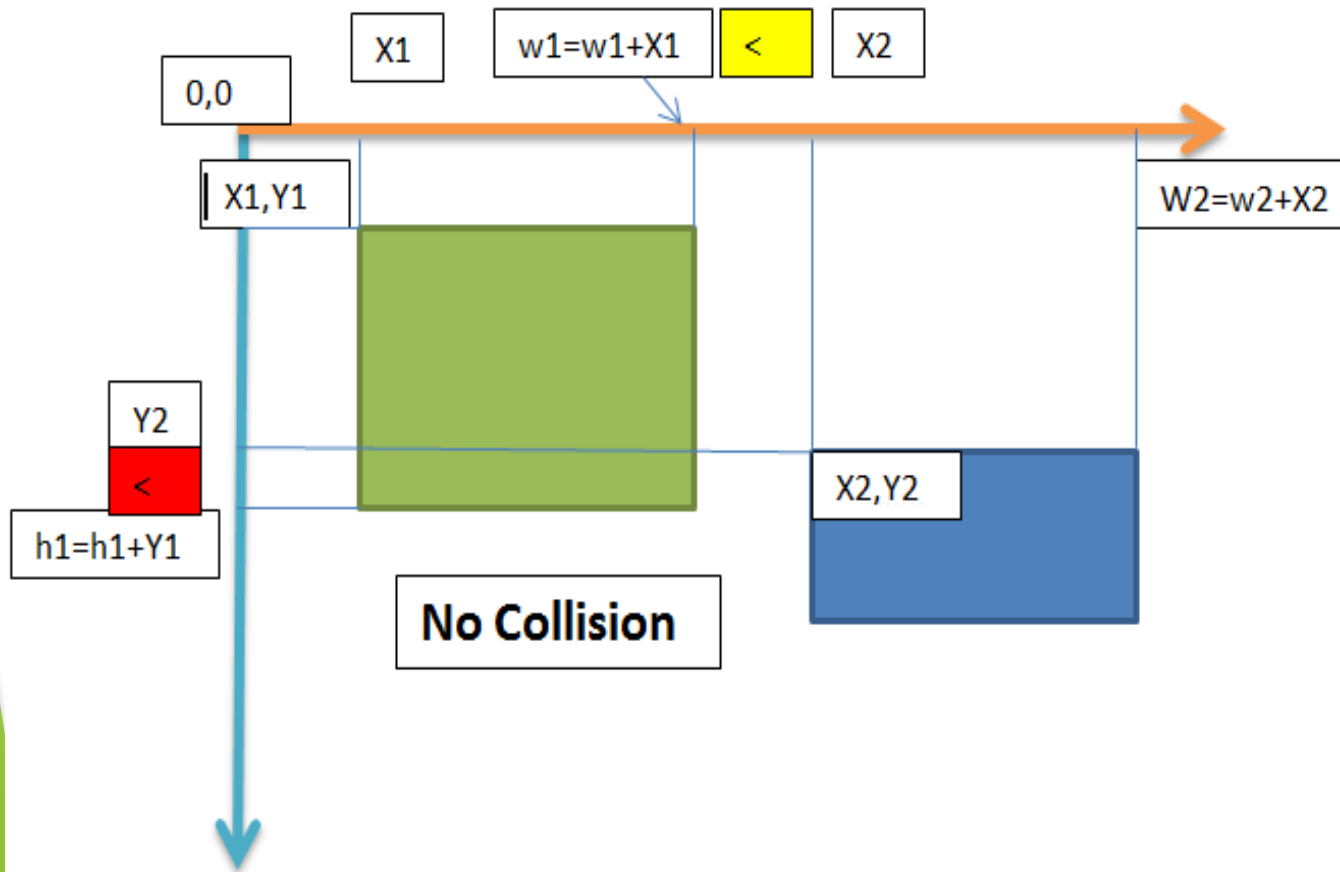
# Collision detection

- ▶ Some use cases for animation depend on boundary awareness - both elemental and environmental
- ▶ Collision detection is the ability to identify when two elements have made contact or are overlapping
  - Collision detection can be performed in real-time or predictively. Some video game physics engines can “step” forward in time to detect if collision will occur before it happens
- ▶ Convex shape overlap is relatively simple to calculate

# Collision detection - Rect

- ▶ Rectangle collision can be calculated by checking for gaps between any of the 4 sides of the rectangles.
- ▶ If a gap is found, a collision has not occurred.
- ▶ Gap checks:
  - $\text{Rect2.X} > (\text{Rect1.Width} + \text{Rect1.X})$
  - $\text{Rect1.X} > (\text{Rect2.Width} + \text{Rect2.X})$
  - $\text{Rect2.Y} > (\text{Rect1.Height} + \text{Rect1.Y})$
  - $\text{Rect1.Y} > (\text{Rect2.Height} + \text{Rect2.Y})$

# Collision detection:



# Collision detection:

```
function rectIntersect(x1, y1, w1, h1, x2, y2, w2, h2) {  
  if (x2 > w1 + x1 || x1 > w2 + x2 || y2 > h1 + y1 || y1 > h2 + y2){  
    return false;  
  }  
  return true;  
}
```

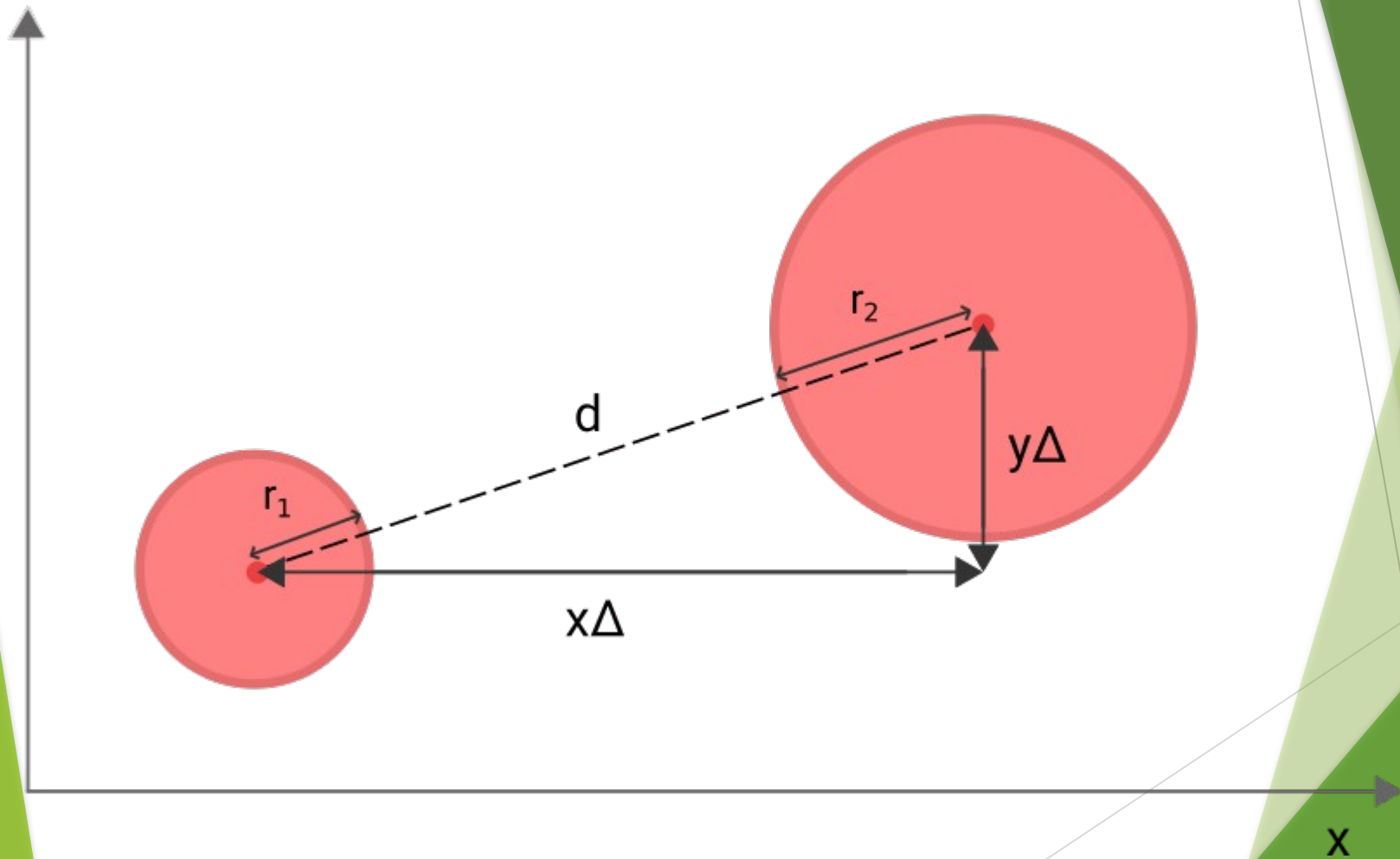


# Collision detection - Circle

- ▶ Circle collision can be calculated by checking the distance between the circle centres and their combined radii.
- ▶ If the distance between circle centres is less than the radii sum, a collision has occurred.
- ▶ Distance calculation ( $A^2 + B^2 = C^2$ ):
  - $\Delta X = \text{circle1.X} - \text{circle2.X}$
  - $\Delta Y = \text{circle1.Y} - \text{circle2.Y}$
  - $\text{Distance} = \sqrt{\Delta X^2 + \Delta Y^2}$

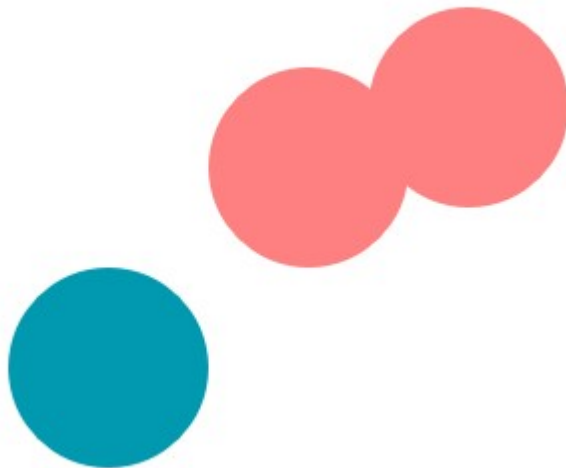


# Collision detection:



# Collision detection:

```
function circleIntersect(x1, y1, r1, x2, y2, r2) {  
  const distanceX = x1 - x2;  
  const distanceY = y1 - y2;  
  const distance = Math.sqrt(distanceX * distanceX + distanceY * distanceY);  
  
  if (distance < r1 + r2) {  
    return true;  
  }  
  return false;  
}
```



# Array declaration in JS:

```
var box1=[x1, y1, 30, 30];
```

```
var box2=[x2, y2, 30, 30];
```

```
var box3=[x3, y3, 30, 30];
```

```
var boxs = [box1,box2, box3]; // static declaration
```

```
var boxs = []; // dynamic declaration
```

```
boxs.push(box1);
```

```
boxs.push(box2);
```

```
boxs.push(box3);
```

# Removing element from array

- ▶ After collision was detected we want to remove the element that collided. There are multiple ways to achieve element removal from an array:
  - `pop()` *removes from end*
  - `shift()` *removes from front*
  - `Splice()` *removes from index, optional replacement elements*
    - The `splice()` method adds/removes items to/from an array, and returns the removed item(s).
    - **Note:** This method replaces the original array with an updated copy

# Array Splice

index	Required. An integer that specifies at what position to add/remove items, Use negative values to specify the position from the end of the array
howmany	Required. The number of items to be removed. If set to 0, no items will be removed
item1, ..., itemX	Optional. The new item(s) to be added to the array

[http://www.w3schools.com/jsref/jsref\\_splice.asp](http://www.w3schools.com/jsref/jsref_splice.asp)



► `Math.floor(x )` -

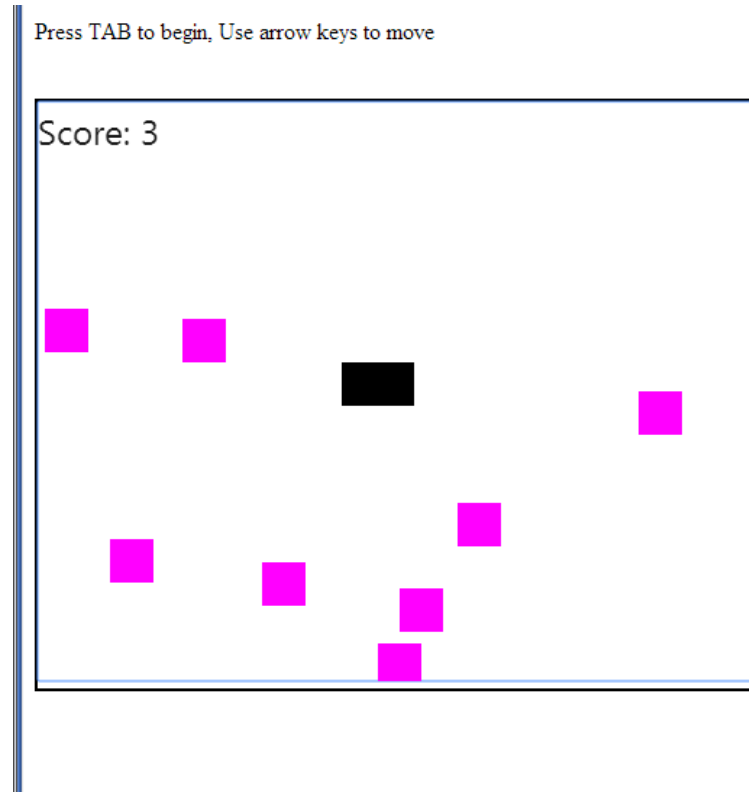
Round a number downward to its nearest integer

► `Math.random()` -

Return a random number between 0 (inclusive) and 1 (exclusive).

# Simple game:

- ▶ Collision detection
- ▶ Score
- ▶ Animation
- ▶ Sound



# Audio

- ▶ Audio can be added to a website in multiple ways
  - Audio tag <audio>      *recommended*
  - Embed tag <embed>      *no longer recommended*
- ▶ Techniques for handling audio vary, usually by number of channels, source/output control, and timing requirements.
  - Web Audio API Provides much greater control at the cost of a more complicated implementation



# Audio

```
<audio src="music.wav" type="audio/wav" preload="auto" controls loop></audio>
```

- ▶ Audio element takes attributes to specify how audio is going to be used:
  - src: the file path. Can be local or URL
  - type: Helps the browser know which file type. Recommended but not mandatory.
  - preload: Whether the audio file or metadata should be loaded before being needed
  - controls: If specified, will generate playback controls for the audio file
  - loop: If specified, will loop the audio track at the end of track playback
  - muted: If specified, will default to muting playback until changed

# Audio Type Support

- ▶ Audio file type for web is usually MP3, WAV, or OGG
- ▶ File type support varies by browser:

Browser	MP3	WAV	OGG
Edge/IE	YES	YES*	YES*
Chrome	YES	YES	YES
Firefox	YES	YES	YES
Safari	YES	YES	NO
Opera	YES	YES	YES

[https://developer.mozilla.org/en-US/docs/Web/Media/Audio\\_and\\_video\\_delivery/Cross-browser\\_audio\\_basics](https://developer.mozilla.org/en-US/docs/Web/Media/Audio_and_video_delivery/Cross-browser_audio_basics)

# Audio Implementation HTML

```
<audio src="music.wav" type="audio/wav" preload="auto" controls loop></audio>
```

```
<audio controls>
  <source src="audiofile.mp3" type="audio/mpeg" />
  <source src="audiofile.ogg" type="audio/ogg" />
  <!-- fallback for non-supporting browsers goes here -->
  <p>
    Your browser does not support HTML audio, but you can still
    <a href="audiofile.mp3">download the music</a>.
  </p>
</audio>
```

# Audio Implementation JS

```
function Sound(src) {  
    this.sound = document.createElement("audio");  
    this.sound.src = src;  
    this.sound.crossOrigin = "anonymous";  
    this.sound.setAttribute("preload", "auto");  
    this.sound.setAttribute("controls", "none");  
    this.sound.setAttribute("looping", "true");  
    this.sound.style.display = "none";  
    document.body.appendChild(this.sound);  
    this.play = function(){  
        if (!this.sound.paused || !this.sound.currentTime) {  
            this.sound.load();  
        }  
        this.sound.play();  
    }  
    this.stop = function(){  
        this.sound.pause();  
    }  
}
```

```
// Credit: p0ss on opengameart.org.  
// https://opengameart.org/content/interface-sounds-starter-pack  
let actionAudio = new Sound("./assets/appear-online.ogg");  
actionAudio.play();
```

# Game Sound

[https://www.w3schools.com/graphics/game\\_sound.asp](https://www.w3schools.com/graphics/game_sound.asp)

Run »

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<style>
canvas {
  border:1px solid #d3d3d3;
  background-color: #f1f1f1;
}
</style>
</head>
<body onload="startGame()">
<script>

var myGamePiece;
var myObstacles = [];
var mySound;

function startGame() {
  myGamePiece = new component(30, 30, "red", 10, 120);
  mySound = new sound("bounce.mp3");
  myGameArea.start();
}

var myGameArea = {
  canvas : document.createElement("canvas"),
  start : function() {
    this.canvas.width = 480;
    this.canvas.height = 270;
    this.context = this.canvas.getContext("2d");
    document.body.insertBefore(this.canvas, document.body.childNodes[0]);
    this.frameNo = 0;
    this.interval = setInterval(updateGameArea, 20);
  },
  stop : function() {
    clearInterval(this.interval);
  },
  clear : function() {
    this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
  }
}
```

# Resources:

- ▶ <http://ie.microsoft.com/testdrive/Graphics/CanvasPad/Default.html>
- ▶ [http://www.w3schools.com/jsref/jsref\\_splice.asp](http://www.w3schools.com/jsref/jsref_splice.asp)
- ▶ <http://www.webbingways.com/bookfiles/chapter3/>
- ▶ [http://www.homeandlearn.co.uk/JS/html5\\_canvas\\_keyboard\\_keys.html](http://www.homeandlearn.co.uk/JS/html5_canvas_keyboard_keys.html)
- ▶ <http://javascript.info/tutorial/bubbling-and-capturing#capturing>
- ▶ [http://www.w3schools.com/js/js\\_math.asp](http://www.w3schools.com/js/js_math.asp)
- ▶ [https://www.w3schools.com/graphics/game\\_sound.asp](https://www.w3schools.com/graphics/game_sound.asp)
- ▶ <https://jschollitt.github.io/week7.html>