Running code on Visual Studio code:

Run code: Ctrl+Alt+N
Stop Run: Ctrl+Alt+M

Javascript Notes:

```
<script>
        document.write("<h1>Hello World</h1>");
</script>
```

Output to the browser console: **console.log()**

Multi-line comment /* and */

Code	Outputs
\'	single quote
\"	double quote
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	form feed

Operator	Description	Example
+	Addition	25 + 5 = 30
-	Subtraction	25 - 5 = 20
*	Multiplication	10 * 20 = 200
/	Division	20 / 2 = 10
%	Modulus	56 % 3 = 2
++	Increment	var a = 10; a++; Now a = 11
--	Decrement	var a = 10; a--; Now a = 9

Operator	Description	Example	Result
var++	Post Increment	var a = 0, b = 10; var a = **b++**;	a = 10 and b = 11
++var	Pre Increment	var a = 0, b = 10; var a = **++b**;	a = 11 and b = 11
var--	Post Decrement	var a = 0, b = 10; var a = **b--**;	a = 10 and b = 9
--var	Pre Decrement	var a = 0, b = 10; var a = **--b**;	a = 9 and b = 9

Operator	Example	Is equivalent to
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x -y
*=	x *= y	x= x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Operator	Description	Example
==	Equal to	5 == 10 false
===	Identical (equal and of same type)	5 === 10 false
!=	Not equal to	5 != 10 true
!==	Not Identical	10 !== 10 false
>	Greater than	10 > 5 true
>=	Greater than or equal to	10 >= 5 true
<	Less than	10 < 5 false
<=	Less than or equal to	10 <= 5 false

Logical Operators	
&&	Returns true, if both operands are true
||	Returns true, if one of the operands is true
!	Returns true, if the operand is false, and false, if the operand is true

For (i=1, text=””; i<:5; i++){
	Text = i;
	document.write(i + “
”);
}

Print EVEN values:
Var x = 0;
for(; x<=20; x+=2){
	document.write(x);
}

```
function main() {
   var depth = parseInt(readLine(), 10);
}
```

<script src=”script.js”>

```
var amount = parseFloat(readLine(), 10);
```

objectName.propertyName
//or
objectName['propertyName']

objectName.methodName()

Var John = new person (“John”, 25)

```
function contact(name, number)
{
   this.name = name;
   this.number = number;
   this.print = print;
}

function print()
{
   console.log(this.name + ": " + this.number);
}

var a = new contact("David", 12345);
var b = new contact("Amy", 987654321)
a.print();
b.print();
```

ARRAYS:

Var course = **new Array(**“HTML”, “CSS”, “JS”**);**

```
var course = courses[0]; // HTML
courses[1] = "C++"; //Changes the second element
```

JavaScript's concat() method allows you to join arrays and create an entirely new array.

```
Var c1 = [“HTML”, “CSS”];
Var c2 = [“JS”, “C++”];
Var courses = c1.concat(c2);
```

While many programming languages support arrays with named indexes (text instead of numbers), called **associative arrays** JavaScript **does not**.
However, you still can use the named array syntax, which will produce an object.

```
Var person = []; // empty array
```

```
Person [“age”] = 46;
document .write(person[“age”]);
```

Property	Description
E	Euler's constant
LN2	Natural log of the value 2
LN10	Natural log of the value 10
LOG2E	The base 2 log of Euler's constant (E)
LOG10E	The base 10 log of Euler's constant (E)
PI	Returns the constant PI

```
document.write(Math.PI);
```

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
asin(x)	Returns the arcsine of x, in radians
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
atan2(y,x)	Returns the arctangent of the quotient of its arguments
ceil(x)	Returns x, rounded upwards to the nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of E^x
floor(x)	Returns x, rounded downwards to the nearest integer
log(x)	Returns the natural logarithm (base E) of x
max(x,y,z,...,n)	Returns the number with the highest value
min(x,y,z,...,n)	Returns the number with the lowest value
pow(x,y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds x to the nearest integer
sin(x)	Returns the sine of x (x is in radians)
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle

Var answer = Math.sqrt(x);
clearInterval() is called or the window is closed.
setInterval(myAlert, 3000);

Var d = new Date(); -> new Date(milliseconds)

Method	Description
getFullYear()	gets the year
getMonth()	gets the month
getDate()	gets the day of the month
getDay()	gets the day of the week
getHours()	gets the hour
getMinutes()	gets the minutes
getSeconds()	gets the seconds
getMilliseconds()	gets the milliseconds

Var hours = d.getHourse();va

```
function main() {
    var increase = parseInt(readLine(), 10);
    var prices = [98.99, 15.2, 20, 1026];
    //your code goes here
    for(var i=0;i<=prices.length-1;i++){
        prices[i]=prices[i]+increase;
    }
    console.log(prices);
}
```

DOM - Document Object Model
document.body.innerHTML = “Some text”;

document.getElementById(id)

Var elem = document.getElementBtId(“demo”);
elem.innerHTML = “Hello World”;

```
var arr = document.getElementsByClassName("demo");
```

```
<p>hi</p>
<p>hello</p>
<p>hi</p>
<script>
var arr = document.getElementsByTagName("p");
for (var x = 0; x < arr.length; x++) {
arr[x].innerHTML = "Hi there";
}
</script>
```

Each element in the DOM has a set of properties and methods that provide information about their relationships in the DOM:
element.**childNodes** returns an array of an element's child nodes.
element.**firstChild** returns the first child node of an element.
element.**lastChild** returns the last child node of an element.
element.**hasChildNodes** returns true if an element has any child nodes, otherwise false.
element.**nextSibling** returns the next node at the same tree level.
element.**previousSibling** returns the previous node at the same tree level.
element.**parentNode** returns the parent node of an element.

As we have seen in the previous lessons, we can change the text content of an element using the **innerHTML** property.
Similarly, we can change the attributes of elements.
For example, we can change the **src** attribute of an image:

```
<img id="myimg" src="orange.png" alt="" />
<script>
var el = document.getElementById("myimg");
el.src = "apple.png";
</script>
```

```
<a href = “http example.com”> Some link </a>
<script>
Var el = document.getElementByTagName(“a”);
El[0].href = “http sololearn.com”
</script>
```

```
<img id="myimg" src="orange.png" alt="" />
<script>
var el = document.getElementById("myimg");
el.src = "apple.png";
</script>
```

```
Changing the background color of all the span elements of the page

Var s = document
          getElementByTageName(“span”);
for(var x=0; x<s.length; x++){
          s[x].style.backgroundColor = “33EA73”

}

var node = document.createTextNode("Some new text");
```

```
<div id = “demo”>some content</div>

<script>
	//creating a new paragraph
	Var p = document.createElement(“p”);
	Var node = document.createTextNode(“Some new text”);
	// adding the text to the paragraph
	p.appendChild(node);

	Var div = document.getElementById(“demo”);
	//adding the paragraph to the div
	div.appendChilod(p);
</script>

//Creating a simple animation
<style>
#containter{
	Width: 200px;
	Height: 200px
	Background: green;
	Position: relative;
}

#box{
	Width:50px;
	Height: 50px;
	Background: red;
	Position: absolute;
}
</style>
<div id = “container”>
	<div id=”box”></div>
</div>
```

To create an animation, we need to change the properties of an element at small intervals of time. We can achieve this by using the **setInterval**() method, which allows us to create a timer and call a function to change properties repeatedly at defined intervals (in milliseconds).

For example:

```
var t = setInterval(move, 500);
// starting position
var pos = 0;
//our box element
var box = document.getElementById("box");

function move() {
```

```
pos += 1;
box.style.left = pos+"px"; //px = pixels
}
```

However, this makes our box move to the right forever. To stop the animation when the box reaches the end of the container, we add a simple check to the move() function and use the clearInterval() method to stop the timer.

```
function move() {
if(pos >= 150) {
clearInterval(t);
}
else {
pos += 1;
box.style.left = pos+"px";
}
}
```

```
<button onclick=”show”>click me</button>
<script>
Function show(){
        alert(“Hi there”);
}
</script>
```

The onload and onunload events are triggered when the user enters or leaves the page. These can be useful when performing actions after the page is loaded.

```
<body onload="doSomething()">

//after the whole page is loaded
Window.onload = function(){
        //code
}
```

The first parameter is the event's type (like "click" or "mousedown").
The second parameter is the function we want to call when the event occurs.
The third parameter is a Boolean value specifying whether to use event bubbling or event capturing. This parameter is optional, and will be described in the next lesson.

```
element.addEventListener(event, function, useCapture);
```

Event Propogation Explained.

1 = Capturing - Capture phase where everything above this element is notified

2 = Target - Target phase where we reach the actual element where the event occurred
3 = Bubbling - bubbling phase where everything above it is notice
0 = None

In bubbling, the innermost element's event is handled first and then the outer element's event is handled. The <p> element's click event is handled first, followed by the element's click event.

In capturing, the outermost element's event is handled first and then the inner. The element's click event is handled first, followed by the <p> element's click event.

The addEventListener() method allows you to specify the propagation type with the "useCapture" parameter.

```
addEventListener(event, function, useCapture)
```

The default value is false, which means the bubbling propagation is used; when the value is set to true, the event uses the capturing propagation.

```
//Capturing propagation
elem1.addEventListener("click", myFunction, true);

//Bubbling propagation
elem2.addEventListener("click", myFunction, false);
```

Now we can create a sample image slider project. The images will be changed using "Next" and "Prev" buttons.
Now, let’s create our HTML, which includes an image and the two navigation buttons:

```
<div>
        <button>Prev</button>
        <img id = “slider”
src=”http://www.sololearn.com/uploads/slider/1.jpg”
        width=”200px” height = “100px”/>
<button> Next </button>
</div>
```

```
var images = [
"http://www.sololearn.com/uploads/slider/1.jpg",
"http://www.sololearn.com/uploads/slider/2.jpg",
"http://www.sololearn.com/uploads/slider/3.jpg"
];

<div>
<button onclick="prev()"> Prev </button>
<img id="slider" src="http://www.sololearn.com/uploads/slider/1.jpg"
width="200px" height="100px"/>
<button onclick="next()"> Next </button>
</div>

<form onsubmit="return validate()" method="post">
Number: <input type="text" name="num1" id="num1" />
<br />
Repeat: <input type="text" name="num2" id="num2" />
<br />
<input type="submit" value="Submit" />
</form>
```