Core Objects

JavaScript Arrays

Arrays store multiple values in a single variable.

To store three course names, you need three variables.

```
var course1 ="HTML";
var course2 ="CSS";
var course3 ="JS";
```

But what if you had 500 courses? The solution is an **array**.

```
var courses = new Array("HTML", "CSS", "JS");
```

> This syntax declares an array named **courses**, which stores three values, or elements.

Accessing an Array

You refer to an array element by referring to the **index number** written in **square brackets**.
This statement accesses the value of the first element in **courses** and changes the value of the second element.

```
var courses = new Array("HTML", "CSS", "JS");
var course = courses[0]; // HTML
courses[1] = "C++"; //Changes the second element
```

> [0] is the first element in an array. [1] is the second. Array indexes start with **0**.

Accessing an Array

Attempting to access an index outside of the array, returns the value **undefined**.

```
var courses = new Array("HTML", "CSS", "JS");
document.write(courses[10]);
//Outputs "undefined"
```

Try It Yourself

> Our **courses** array has just 3 elements, so the 10th index, which is the 11th element, does not exist (is undefined).

Creating Arrays

You can also declare an array, tell it the number of elements it will store, and add the elements later.

```
var courses = new Array(3);
courses[0] = "HTML";
courses[1] = "CSS";
courses[2] = "JS";
```

Try It Yourself

> An array is a special type of **object**.
> An array uses **numbers** to access its elements, and an object uses **names** to access its members.

Creating Arrays

JavaScript arrays are dynamic, so you can declare an array and not pass any arguments with the Array() constructor. You can then add the elements dynamically.

```
var courses = new Array();
courses[0] = "HTML";
courses[1] = "CSS";
courses[2] = "JS";
courses[3] = "C++";
```

Try It Yourself

> You can add as many elements as you need to.

Array Literal

For greater simplicity, readability, and execution speed, you can also declare arrays using the **array literal** syntax.

```
var courses = ["HTML", "CSS", "JS"];
```

Try It Yourself

This results in the same array as the one created with the **new Array()** syntax.

> You can access and modify the elements of the array using the index number, as you did before.
> The **array literal** syntax is the recommended way to declare arrays.

The length Property

JavaScript arrays have useful **built-in** properties and methods.

An array's **length** property returns the number of it's elements.

```
var courses = ["HTML", "CSS", "JS"];
document.write(courses.length);
//Outputs 3
```

Try It Yourself

> The **length** property is always one more than the highest array index.
> If the array is empty, the length property returns **0**.

Combining Arrays

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

Example:

```
var c1 = ["HTML", "CSS"];
var c2 = ["JS", "C++"];
var courses = c1.concat(c2);
```

Try It Yourself

The **courses** array that results contains 4 elements (HTML, CSS, JS, C++).

> The **concat** operation does not affect the *c1* and *c2* arrays - it returns the resulting concatenation as a new array.

Associative Arrays

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.
For example:

```
var person = []; //empty array
person["name"] = "John";
person["age"] = 46;
document.write(person["age"]);
//Outputs "46"
```

Try It Yourself

Now, person is treated as an object, instead of being an array.
The named indexes "name" and "age" become properties of the person object.

> As the person array is treated as an object, the standard array methods and properties will produce incorrect results. For example, **person.length** will return 0.

Associative Arrays

Remember that JavaScript **does not** support arrays with named indexes.
In JavaScript, arrays always use numbered indexes.

It is better to use an object when you want the index to be a string (text).
Use an array when you want the index to be a **number**.

> If you use a named index, JavaScript will redefine the array to a standard object.

The Math Object

The Math object allows you to perform mathematical tasks, and includes several properties.

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

For example:

```
document.write(Math.PI);
//Outputs 3.141592653589793
```

Try It Yourself

> Math has no constructor. There's no need to create a Math object first.

Math Object Methods

The Math object contains a number of methods that are used for calculations:

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

For example, the following will calculate the **square root** of a number.

```
var number = Math.sqrt(4);
document.write(number);
//Outputs 2
```

Try It Yourself

> To get a random number between 1-10, use Math.random(), which gives you a number between 0-1. Then multiply the number by 10, and then take Math.ceil() from it: Math.ceil(Math.random() * 10).

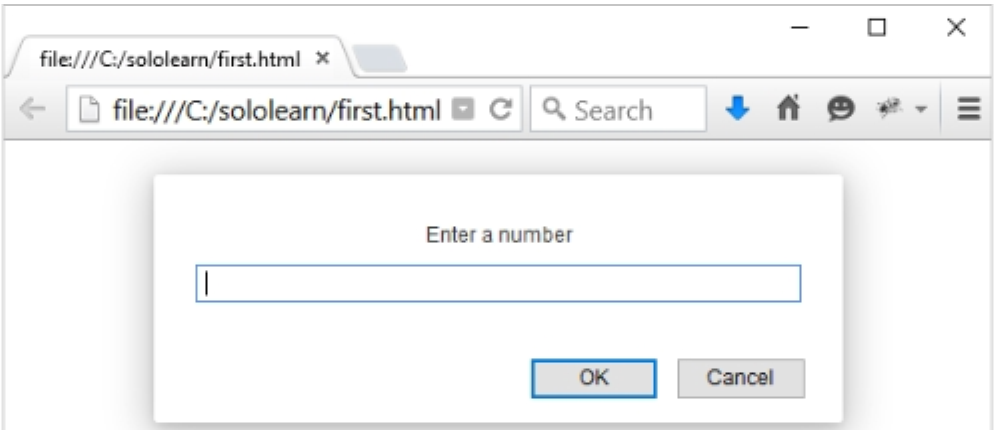
The Math Object

Let's create a program that will ask the user to input a number and alert its square root.

```
var n = prompt("Enter a number", "");
var answer = Math.sqrt(n);
alert("The square root of " + n + " is " + answer);
```

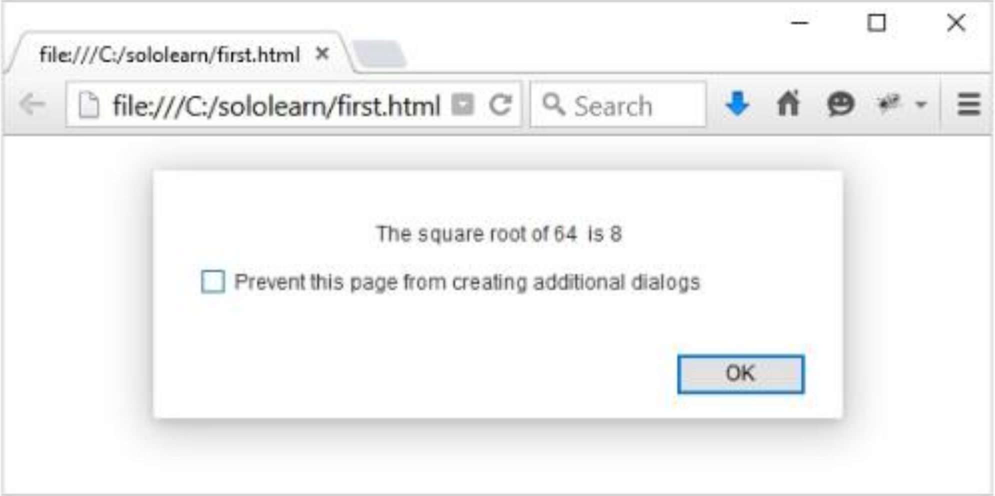
Try It Yourself

Result:


file:///C:/sololearn/first.html
file:///C:/sololearn/first.html
Search
OK
Cancel

Enter a number

Enter a number, such as 64.


file:///C:/sololearn/first.html ×
file:///C:/sololearn/first.html
Search
OK

The square root of 64 is 8

□ Prevent this page from creating additional dialogs

> Math is a handy object. You can save a lot of time using Math, instead of writing your own functions every time.

setInterval

The setInterval() method calls a function or evaluates an expression at specified intervals (in milliseconds).
It will continue calling the function until **clearInterval()** is called or the window is closed.

For example:

```
function myAlert() {
  alert("Hi");
}
setInterval(myAlert, 3000);
```

Try It Yourself

This will call the myAlert function every 3 seconds (1000 ms = 1 second).

> Write the **name** of the function without parentheses when passing it into the **setInterval** method.

The Date Object

The **Date** object enables us to work with dates.
A date consists of a year, a month, a day, an hour, a minute, a second, and milliseconds.

Using **new Date**(), create a new date object with the **current date and time**.

```
var d = new Date();
//d stores the current date and time
```

The other ways to initialize dates allow for the creation of new date objects from the **specified date and time**.

```
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

> JavaScript dates are calculated in milliseconds from 01 January, 1970 00:00:00 Universal Time (UTC). One day contains 86,400,000 millisecond.

For example:

```
//Fri Jan 02 1970 00:00:00
var d1 = new Date(86400000);

//Fri Jan 02 2015 10:42:00
var d2 = new Date("January 2, 2015 10:42:00");

//Sat Jun 11 1988 11:42:00
var d3 = new Date(88,5,11,11,42,0,0);
```

> JavaScript counts months from 0 to 11. January is 0, and December is 11.
> Date objects are static, rather than dynamic. The computer time is ticking, but date objects don't change, once created.

Date Methods

When a Date object is created, a number of **methods** make it possible to perform operations on it.

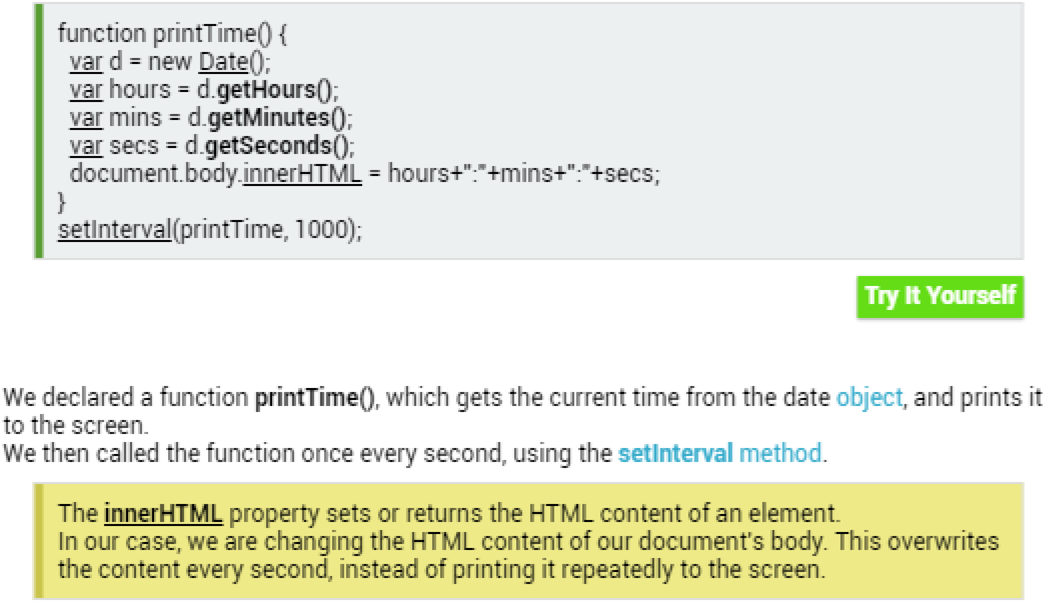
Method	Description
getFullYear()	gets the year
getMonth()	gets the month
getDate()	gets the day of the month
getDay()	gets the day of the week
getHours()	gets the hour
getMinutes()	gets the minutes
getSeconds()	gets the seconds
getMilliseconds()	gets the milliseconds

For example:

```
var d = new Date();
var hours = d.getHours();
//hours is equal to the current hour
```

Try It Yourself

Let's create a program that prints the current time to the browser once every second.



```
function printTime() {
  var d = new Date();
  var hours = d.getHours();
  var mins = d.getMinutes();
  var secs = d.getSeconds();
  document.body.innerHTML = hours+":"+mins+":"+secs;
}
setInterval(printTime, 1000);
```

We declared a function **printTime()**, which gets the current time from the date object, and prints it to the screen.
We then called the function once every second, using the **setInterval** method.

> The **innerHTML** property sets or returns the HTML content of an element.
> In our case, we are changing the HTML content of our document's body. This overwrites the content every second, instead of printing it repeatedly to the screen.

END.