Basic Concepts

Variables

Variables are containers for storing data values. The value of a variable can change throughout the program.
Use the var keyword to declare a variable:

```
var x = 10;
```

In the example above, the value **10** is assigned to the variable **x**.

> JavaScript is case sensitive. For example, the variables *lastName* and *lastname*, are two different variables.

The Equal Sign

In JavaScript, the equal sign (=) is called the "**assignment**" operator, rather than an "equal to" operator.
For example, **x = y** will assign the value of **y** to **x**.

> A variable can be declared without a value. The value might require some calculation, something that will be provided later, like user input.
> A variable declared without a value will have the value **undefined**.

Using Variables

Let's assign a value to a variable and output it to the browser.

```
var x = 100;
document.write(x);
```

Try It Yourself

Result:

Using variables is useful in many ways. You might have a thousand lines of code that may include the variable x. When you change the value of x **one time**, it will automatically be changed in **all places** where you used it.

> Every written "instruction" is called a **statement**. JavaScript statements are separated by **semicolons**.

Naming Variables

JavaScript variable names are case-sensitive.
In the example below we changed x to uppercase:

```
var x = 100;
document.write(X);
```

Try It Yourself

This code will not result in any output, as x and X are two different variables.

Naming rules:
- The first character **must be** a letter, an underscore (_), or a dollar sign ($). Subsequent characters may be letters, digits, underscores, or dollar signs.
- Numbers are **not allowed** as the first character.
- Variable names **cannot** include a **mathematical or logical operator** in the name. For instance, *2*something* or *this+that*;
- JavaScript names **must not contain spaces**.

> Hyphens are not allowed in JavaScript. It is reserved for subtractions.

Naming Variables

There are some other rules to follow when naming your JavaScript variables:

- You **must not** use any **special symbols**, like *my#num, num%*, etc.
- Be sure that you do not use any of the following JavaScript reserved words.

Reserved Words in JavaScript			
abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

> When you get more familiar with JavaScript, remembering these keywords will be much easier.

Data Types

The term **data type** refers to the types of values with which a program can work. JavaScript variables can hold many data types, such as **numbers**, **strings**, **arrays**, and more.

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point, etc.

JavaScript numbers can be written with or without decimals.

```
var num = 42; // A number without decimals
```

This variable can be easily changed to other types by assigning to it any other data type value, like num = 'some random string'.

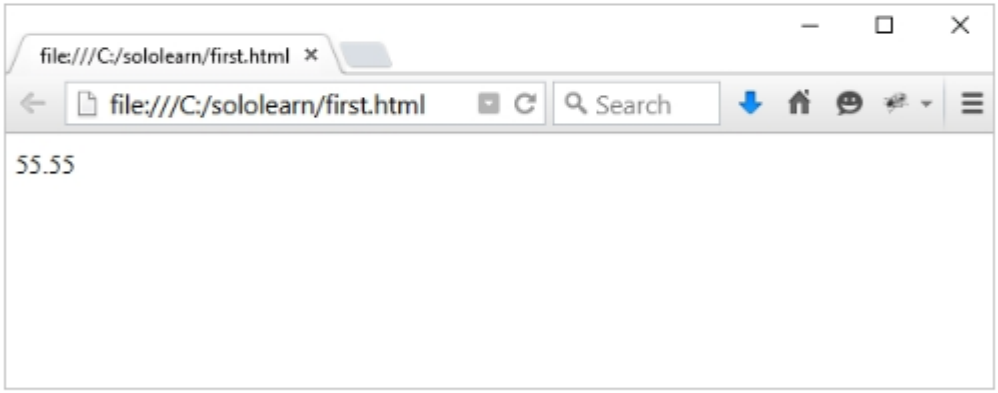
Floating-Point Numbers

JavaScript numbers can also have decimals.

```
<script>
  var price = 55.55;
  document.write(price);
</script>
```

Try It Yourself

Result:



JavaScript numbers are always stored as **double precision floating point numbers**.

Strings

JavaScript **strings** are used for storing and manipulating text.
A string can be any text that appears within **quotes**. You can use single or double quotes.

```
var name = 'John';
var text = "My name is John Smith";
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string.

```
var text = "My name is 'John' ";
```

You can get double quotes inside of double quotes using the escape character like this: \" or \' inside of single quotes.

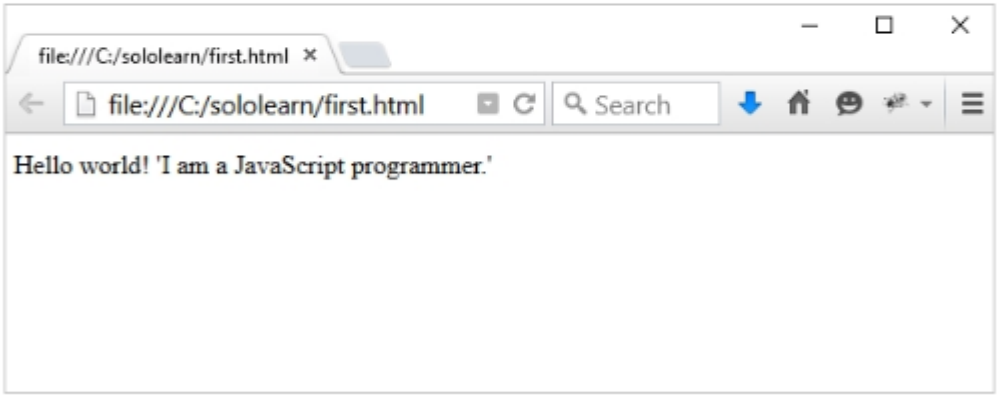
Strings

As strings must be written within quotes, quotes inside the string must be handled. The **backslash (\) escape character** turns special characters into string characters.

```
var sayHello = 'Hello world! \'I am a JavaScript programmer.\' ';
document.write(sayHello);
```

Try It Yourself

Result:



The escape character (\) can also be used to insert other special characters into a string. These special characters can be added to a text string using the backslash sign.

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

If you begin a string with a single quote, then you should also end it with a single quote. The same rule applies to double quotes. Otherwise, JavaScript will become confused.

Booleans

In JavaScript Boolean, you can have one of two values, either **true** or **false**.
These are useful when you need a data type that can only have one of two values, such as Yes/No, On/Off, True/False.

Example:

```
var isActive = true;
var isHoliday = false;
```

The Boolean value of 0 (zero), null, undefined, empty string is **false**.
Everything with a "real" value is **true**.

Arithmetic Operators

Arithmetic operators perform arithmetic functions on numbers (literals or variables).

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

In the example below, the addition operator is used to determine the sum of two numbers.

```
var x = 10 + 5;
document.write(x);

// Outputs 15
```

Try It Yourself

You can add as many numbers or variables together as you want or need to.

```
var x = 10;
var y = x + 5 + 22 + 45 + 6548;
document.write(y);

//Outputs 6630
```

Try It Yourself

You can get the result of a string expression using the eval() function, which takes a string expression argument like eval("10 * 20 + 8") and returns the result. If the argument is empty, it returns undefined.

Multiplication

The multiplication operator (*) multiplies one number by the other.

```
var x = 10 * 5;
document.write(x);

// Outputs 50
```

Try It Yourself

10 * '5' or '10' * '5' gives the same result. Multiplying a number with string values like 'sololearn' * 5 returns NaN (Not a Number).

Division

The / operator is used to perform division operations:

```
var x = 100 / 5;
document.write(x);

// Outputs 20
```

Try It Yourself

Remember to handle cases where there could be a division by 0.

The Modulus

Modulus (%) operator returns the division remainder (what is left over).

```
var myVariable = 26 % 6;

//myVariable equals 2
```

Try It Yourself

In JavaScript, the modulus operator is used not only on integers, but also on floating point numbers.

Increment & Decrement

Increment ++
The increment operator increments the numeric value of its operand by one. If placed before the operand, it returns the incremented value. If placed after the operand, it returns the original value and then increments the operand.
Decrement --
The decrement operator decrements the numeric value of its operand by one. If placed before the operand, it returns the decremented value. If placed after the operand, it returns the original value and then decrements the operand.

Some examples:

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

As in school mathematics, you can change the order of the arithmetic operations by using parentheses.
Example: var x = (100 + 50) * 3;

Assignment Operators

Assignment operators assign values to JavaScript variables.

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

You can use multiple assignment operators in one line, such as x -= y += 9.

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values. They return **true** or **false**.

The **equal to** (==) operator checks whether the operands' values are equal.

```
var num = 10;
// num == 8 will return false
```

Try It Yourself

You can check all types of data; comparison operators always return true or false.

Comparison Operators

The table below explains the comparison operators.

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

When using operators, be sure that the arguments are of the same data type; numbers should be compared with numbers, strings with strings, and so on.

Logical Operators

Logical Operators, also known as **Boolean** Operators, evaluate the expression and return **true** or **false**.

The table below explains the logical operators (**AND**, **OR**, **NOT**).

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

You can check all types of data; comparison operators always return true or false.

Logical Operators

In the following example, we have connected two Boolean expressions with the **AND** operator.

```
(4 > 2) && (10 < 15)
```

Try It Yourself

For this expression to be **true**, both conditions must be **true**.
- The first condition determines whether 4 is greater than 2, which is **true**.
- The second condition determines whether 10 is less than 15, which is also **true**.

Based on these results, the whole expression is found to be **true**.

Conditional (Ternary) Operator

Another JavaScript conditional operator assigns a value to a variable, based on some condition.
Syntax:

```
variable = (condition) ? value1: value2
```

For example:

```
var isAdult = (age < 18) ? "Too young": "Old enough";
```

Try It Yourself

If the variable *age* is a value below 18, the value of the variable *isAdult* will be "Too young". Otherwise the value of *isAdult* will be "Old enough".

Logical operators allow you to connect as many expressions as you wish.

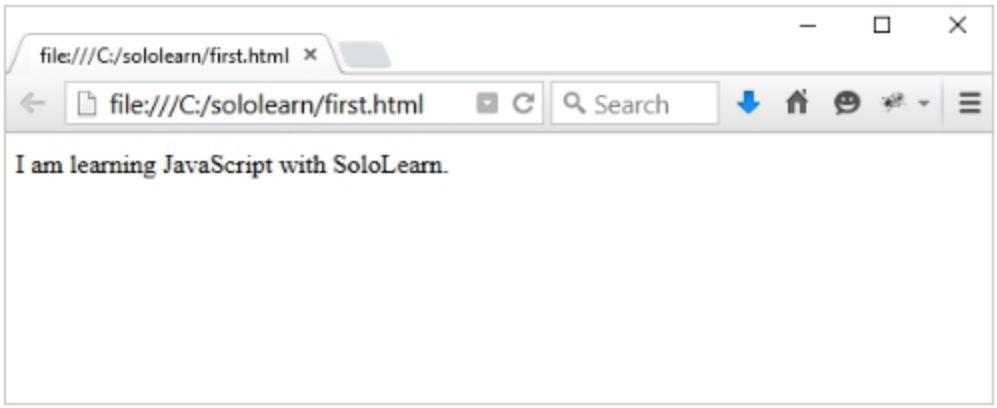
String Operators

The most useful operator for strings is *concatenation*, represented by the + sign.
Concatenation can be used to build strings by joining together multiple strings, or by joining strings with other types:

```
var mystring1 = "I am learning ";
var mystring2 = "JavaScript with SoloLearn.";
document.write(mystring1 + mystring2);
```

Try It Yourself

The above example declares and initializes two string variables, and then concatenates them.



Numbers in quotes are treated as strings: "42" is not the number 42, it is a string that includes two characters, 4 and 2.

End.