JavaScript Week01 – Operations

Introduction to JavaScript:

JavaScript is a high-level programming language that is widely used in web development. It is designed to add interactivity and dynamic behavior to web pages. JavaScript code is executed on the client-side, which means that it runs directly in the browser of the user. This allows web developers to create dynamic and interactive user interfaces that can respond to user actions in real-time, without requiring the page to be refreshed. It is a very versatile language that can be used to create a wide range of interactive web applications, from simple calculators to complex online games, and it has a large and active developer community. There are many frameworks, libraries, and tools available that make it easier to develop complex web applications with JavaScript such as Reach and jQuery. As with anything, we will start at the bottom with foundational skills and build up from there.

Variables:

In JavaScript, variables are used as storage containers for data that a program needs to use. A variable is declared using the keyword var and a named typically in lower camel case. The variable can then be used to store a value, which can be changed or updated as needed during the execution of the program. For example, let’s say we want to create a program that calculates the area of a rectangle in JavaScript. We can create two variables, length and width, using the var keyword, which will be used to store the length and width of the rectangle, respectively. We can then use these variables to calculate the area of the rectangle by multiplying them together and storing the result in a third variable called areaOfRectangle. Here is an example of how we can implement this in JavaScript:

Var length = 10; // declare a variable called length and assign it a value of 10

Var width = 5; // declare a variable called width and assign it a value of 5

Var areaOfRectangle = length \* width; // assign the area value to the areaOfRectangle variable

Console.log():

In JavaScript, the console is a built-in object that provides developers with a way to output messages and data to the console window. The most widely used console method is console.log(). It is an essential tool for debugging and troubleshooting JavaScript code, and it is widely used by developers to inspect and understand their programs. Here is an example of how to use console.log() to print a message:

Console.log(“Hello, world!”);

This would output Hello World! To the console window.

Now lets take the areaofRectangle example and print the value of the area to the console:

Var length = 10;

Var width = 5;

Var areaOfRectangle = length \* width;

Console.log(areaOfRectangle); // this will print: 50

When comparing values, what prints to the console will either be the value true or false. For Example:

Console.log(7 > 2); // prints true to the console

Console.log(7 < 2); // prints false to the console

Reassign variables:

Variables can be reassigned, or in other words, assigned to different values. For example:

// Declare a variable, assign it a value, and print it to the console

Var myVariable = “Hello, world!”;

Console.log(myVariable); // this will print: Hello, world!

// Reassign the value of myVariable to “Goodbye, world!”, notice the var keyword is missing

myVariable = “Goodbye, world!”;

// log the new value of myVariable to the console

Console.log(myvariable); // this will print: Goodbye, world!

Vocabulary Terms:

Comparison operator – used to compare two values and returns a value true or false. Examples of comparison operators in JavaScript include greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=).

Concatenate – The process of joining two or more strings together to create a new, longer string. In JavaScript, you can concatenate strings using the + operator.

Equality Operator – An equality operator is used to compare two values for equality and returns a Boolean value of true or false. The double equals sign (==) is used to check for loose equality, which means that the two values are equal in value but not necessarily in type. The triple equal sign (===) is used to check for strict equality, which means that the two values are equal in value and type.

Math Operations – JavaScript provides a set of built-in math operations that can be used to perform calculations on numbers, such as addition (+), subtraction (-), multiplication (\*), division (/), and modulo (%), among others.

Null: The null type represents an intentional absence of any object value and has exactly one value: null. In JavaScript, null is a primitive value and is not an object but can be used where an object is expected in JavaScript.

Number: a data type that represents both integers and floating-point numbers.

Strict Equality Operators: the strict Equality operator, or ===, is a comparison operator that tests if two values are equal without performing type coercion, If the two values being compared are of different types, the result will be false.

String: A data type that represents a sequence of characters. Strings can be created using single or double quotes and can be concatenated using the (+) operator.

Type coercion: The automatic or implicit conversion of a value from one data type to another by the programming language, often in order to perform an operation or comparison.

Undefined: The undefined type represents a variable that has been declared but has not been assigned a value. It only has one value: undefined.

Variable – a container that holds a value, which can be a string, number, Boolean, or other data type.

Var – a keyword that was originally used to declare variables in JavaScript.

Concatenation

In Programming, the + operator can be used in the addition of numbers and also in the joining of strings. When strings are put together it is called concatenation and the + operator is then referred to as the concatenation operator. For example, let’s say you have two strings “Hello” and “world” stored in the variables greeting and subject. To concatenate them together, you can use the + operator and quotes with a space in between like this:

Var greeting = “Hello”;

Var subject = “World”;

Var message = greeting + “ “ + subject;

Console.log(message);

// This is what will print out

Hello World

// without the quotes and space, this will print to the console instead

HelloWorld

Concatenation can get tricky when mixing data types. Adding a string and a number using the + operator, will result in a string. For example, the expression “5” + 3 would result in the string “53”.

Var x = 5; // this is a number

Var y = “3”; // this is a string because it is in quotes

Console.log(x + y); // this will print “53”

Similarly, if you concatenate a number and a string such as 3 + “Apples”, the result is the string “3apples”.

When you add an undefined value to a number in JavaScript, the result will be NaN (Not a number). This is because undefined is not a number and cannot be added to a number. For example:

Var a; // this variable has been declared but not assigned a value, therefore it is undefined

Var b = 4; // this is a number

Console.log(a + b); // this will result in NaN

Concatenation can be useful in a variety of programming tasks, such as building dynamic text strings for use in HTML or other text-based formats or generating output for logging or debugging purposes. However, it’s worth noting that these behaviors are well-defined and consistent within the language, and there are reasons why they are designed this way.

**Equality Operators**

In JavaScript, both the **equality** (==) operator and the **strict equality** (===) operator are used to compare values, but they work in slightly different ways. The == operator compares two values for equality after converting them to a common type. This means that if the two values being compared are of different types, JavaScript will attempt to convert them to a common type before making the comparison. For example:

console.log(5 == "5"); // prints True to the console

In this example, the == operator compares the number 5 and the string "5" for equality. Since the two values are of different types, JavaScript converts the string to a number before making the comparison, and the result is true.

On the other hand, the === operator compares two values for equality without any type conversion. This means that if the two values being compared are of different types, the result will be false. For example:

console.log(5 === "5"); // prints False to the console

In this example, the === operator compares the number 5 and the string "5" for strict equality. Since the two values are of different types, the result is false. In general, it is a good practice to use the === operator instead of the == operator in JavaScript, as it is more strict and can help avoid unexpected results due to type coercion. However, there may be cases where the == operator is more appropriate, such as when comparing values of different types that may need to be coerced to a common type for a valid comparison.

# **JavaScript Week 1 - Variables**

## Variables

Variables are essential components in any programming language. In **JavaScript**, variables serve as storage containers for data that a program needs to use. They allow us to identify specific data so that we can refer to the data by its variable name and write instructions telling the computer what to do with the data by using its variable name.

## Primitive Data Types:

Variables in JavaScript can refer to different types of data to be used in many different ways. **Primitive Data Types** represent data that is not an object, and have no methods or properties defined on them.  
  
  
The seven primitive data types in JavaScript are:

* **String** - A sequence of characters, such as letters or numbers, enclosed within single or double quotes in JavaScript.

* **Number** - A numeric value, which can be either positive or negative, with or without decimals.

* **Boolean** - A logical value that can either be true or false.

* **Undefined** - A variable that has been declared but has not been assigned a value, or a property that does not exist in an object. This type has one value: undefined

* **Null** - A special value that represents a deliberate non-value or absence of any object value. This type has one value: null

* **Bigint** - A numeric value that can represent numbers larger than the maximum safe integer of (2^53)-1, with the suffix n at the end.

* **Symbol** - A primitive data type that represents a unique identifier, often used to create property keys for objects.

**Note**: All **Primitive Types** in JavaScript, except null and undefined, do have corresponding object **wrapper** types, and these **wrapper** types provide useful methods for working with the primitive values.

## Variables and Data Types:

In JavaScript, we don't have to tell a variable what type of data it will hold. The data type is determined implicitly based on the data assigned to the variable. This is called **dynamic typing**. JavaScript is also **loosely typed** (or **weakly typed**), meaning that the data type of a variable can change. For example, a variable could be pointing to a String value and then be changed to hold a Number value.

## Declaring Variables:

To declare variables in JavaScript, we start with a keyword: var or let, followed by the name or identifier of the variable. After declaring the variable name/identifier, we use an **assignment operator** (=) followed by the value we want to assign to the variable. Finally, we end the statement (the line of code) with a semicolon.

For example, suppose we have customer data we are working with, we may have a variable called **customerFirstName** that holds String data representing the first name of a customer. We can declare this variable as follows:

**var** customerFirstName = "Sam";

Anywhere we refer to **customerFirstName** after that line, the computer will substitute in the value "Sam".

## Naming Variables:

It is important to name variables appropriately so that the code is easy to read and understand. In JavaScript, we follow a naming convention called **Camel Case** or **camelCase**, where the first letter of the first word is lowercase, and the first letter of each following word is uppercase. Below are some examples of variables that have values assigned to them:

**var** bankAccountBalance = 100.54;

**var** numberOfFriends = 10;

**var** isHotOutside = false;

**var** isHappy = true;

Notice how there are no quotes around the Number or Boolean values. Strings are denoted with single or double quotes, but Numbers and Booleans do not use quotes.

## ****Examples of Variables****

**String**: A sequence of characters that represents text. Strings are enclosed in either single or double quotes. For example:

**var** greeting = "Hello, world!";

**var** myName = "John";

console.log(myName); // prints: John

**Number**: A numeric value, which can be an integer or a floating-point number. For example:

**var** myAge = 38;

**var** temperature = 98.6;

**console**.log(myAge); // prints: 38

**Boolean**: A value that represents either true or false. Booleans are often used in conditional statements and logical operations. For example:

**var** isSunny = true;

**var** isRainy = false;

console.log(isRainy); //prints: false

**Undefined**: A value that is assigned to a variable that has not been initialized. Undefined is also returned by functions that do not have a return statement. For example:

**var** lastName;

console.log(lastName); // prints: undefined

## ****Reassign Variables****

Variables can be reassigned, or in other words, assigned to different values. For example:

// Declare a variable, assign it a value, and print it to the console

**var** myVariable = "Hello, world!";

console.log(myVariable); // this will print: Hello, world!

// Reassign the value of myVariable to "Goodbye, world!", notice the var keyword is missing

myVariable = "Goodbye, world!";

// Log the new value of myVariable to the console

console.log(myVariable); // this will print: Goodbye, world!

## More Data Type Examples

**Null**: A value that represents the intentional absence of any object value. Null is often used to indicate a deliberate non-value. For example:

**var** y = null;

console.log(y); // logs "null"

**Symbol**: A unique and immutable value that may be used as the key of an Object property. Symbols are often used to create private members in Objects. For example:

**const** key1 = Symbol("key");

**const** key2 = Symbol("key");

**const** obj = {};

obj[key1] = "value1";

obj[key2] = "value2";

console.log(obj[key1]); // logs "value1"

console.log(obj[key2

**BigInt**: A numeric value that can represent integers larger than 2^53 - 1. BigInts are denoted by appending the letter "n" to the end of an integer literal or by calling the BigInt()function. For example:

**var** bigNumber = 9007199254740993n;

**var** largeNumber = BigInt(123456789012345678901234567890);