Introduction of JavaScript

1. What is a data type, and what are the different data types in JavaScript?

In JavaScript, a data type refers to the kind of value a variable holds. A variable is a named container that stores a value of a specific data type, allowing you to reference and manipulate it throughout your code. Data types help the program understand the kind of data it's working with, whether it's a number, text, or something else.

A Number represents both integers and floating-point values. Examples of integers include 7, 19, and 90. Examples of floating point numbers include 3.14 and 5.2. A floating point number is a number with a decimal point.

The next data type is a String.

A String is a sequence of characters, or text, enclosed in quotes. Here are two examples:

"Hello, world"

'JavaScript'

Another data type used in JavaScript is the Boolean type.

A Boolean represents one of two possible values: true or false. You can use a boolean to check if a page is loading, or if a user is logged in or not.

The next two data types used in JavaScript are undefined and null.

undefined means a variable has been declared but hasn't been given a value yet. null means the variable has been intentionally set to "nothing" and does not hold any value. We will explore more on how this works in future lectures.

The next data type we will look at is the Object type.

An Object is a more complex data type that can hold collections of key-value pairs. Let's break this down. The key (also called a property name), is like a label for the data, whereas the value, is the actual data you want to store. Here is an example:

let book = {

title: "The Great Gatsby",

author: "F. Scott Fitzgerald",

year: 1925

};

Each key-value pair in an object is called a property. So we can say that this book object has three properties.

The last two data types are the Symbol and BigInt data types.

A Symbol is a special type of value in JavaScript that is always unique and cannot be changed. It's often used to create unique labels or identifiers for properties:

// Creating two symbols

const symbol1 = Symbol('mySymbol');

const symbol2 = Symbol('mySymbol');

console.log(symbol1 === symbol2); // Outputs: false

BigInt is used for very large numbers that exceed the limit of the Number type:

const bigNumber = 1234567890123456789012345678901234567890n;

console.log(bigNumber);

// Outputs: 1234567890123456789012345678901234567890n

1. What are variables, and what are guidelines for naming JavaScript variables?

JavaScript, variables act as containers for storing data that you can access and modify throughout your program.

You can think of variables as boxes that hold values. With variables, you can keep track of things like numbers or text and refer to these values whenever you need them in your program.

One way to declare a variable in JavaScript, is to use the let keyword.

let age;

Right now, the age variable does not have a value assigned to it. To assign a value to a variable you will need to use the assignment operator like this:

let age = 25;

The assignment operator looks like an equals sign (=) but it doesn't check for equality. The assignment operator is used to assign a value to a variable. This process of assigning a value to a variable is known as initialization

One advantage of using the let keyword to declare variables is that you can reassign values to them. In programming, reassignment means giving a new value to a variable that already has one.

Here is an example of reassigning the value for the age variable:

age = 30;

Now the age variable holds the value of 30. Notice that the let keyword wasn't needed again because the age variable was already declared, so there's no need to declare it a second time.

When using reassignment, you only need to reference the variable name. Reassignment is useful because it allows you to update and change the value stored in a variable as your program runs. A good example of this would be updating points in a game.

Naming variables may seem straightforward, but there are some rules and best practices to ensure your code is readable and functional.

Your variable names should describe what the data represents. For example, instead of using a name like x, a more descriptive name such as age or points makes your code easier to understand.

Variables in JavaScript must begin with a letter, an underscore (\_), or a dollar sign ($). They cannot start with a number.

Variable names are case-sensitive, meaning the word age in all lowercase and the word Age with a capital A are considered different variables.

This is why it's important to stick with a consistent naming convention like camelCase. camelCase is where the first word is all lowercase and each subsequent word starts with an uppercase letter.

Here is an example of using the camelCase naming convention for a variable:

let thisIsCamelCase;

There are certain keywords in JavaScript that you cannot use as variable names, such as let, const, function, or return, as they are reserved for the language itself.

You should also avoid using special characters like exclamation points (!) or at (@) symbols, in your variable names. It is best to keep variable names readable by using letters, numbers, underscores, or dollar signs.

By following these guidelines, your code will be cleaner and more manageable as it grows in complexity.

1. How do let and const differ when is comes to variable declaration, assignment, and reassignment?

The let keyword allows you to declare variables that can be updated or reassigned later. You can think of let as a flexible container – once you've stored a value in it, you can change that value as needed throughout your program.

On the other hand, const is used to declare variables that are constant. Once you assign a value to a variable declared with const, you cannot reassign it.

This makes const ideal for values that you don't want to change accidentally during the execution of your program.

Variables declared with const must be assigned a value at the time of declaration. If you try to declare a const variable without assigning it a value, you will get an error:

1. What is a string in JavaScript, and what is string immutability?

In JavaScript, a string is a sequence of characters used to represent text data. Strings are one of the primitive data types in the language, along with numbers, booleans, null, and undefined.

To create a string in JavaScript, you can use single quotes ('), or double quotes (").

Here is an example of creating two variables that hold string values:

let singleQuotes = 'This is a string';

let doubleQuotes = "This is also a string";

Even though you can use single or double quotes to create strings, it's important to be consistent. If a string begins with a single quote, it must also end with a single quote.

The same applies to double quotes. The following example will throw an error because it starts with a double quote and ends with a single quote:

const improperStr = "Do not do this';

Another thing to take into account is that strings are immutable. In programming, immutability means that once something is created, it cannot be changed. So, when you create a string, you can't change its characters directly. Instead, you would create a new string if you want to make changes.

Here is an example of assigning a new string to a developer variable:

let developer = "Jessica";

developer = "Quincy";

Since strings are immutable, we can't update the first string directly. That is why we are assigning a new string to the developer variable.

1. What is string concatenation, and how can you concatenate strings with variables?

In JavaScript, working with text is an essential part of coding, and often, you'll need to combine or join pieces of text together. This process is called string concatenation.

In this lecture, we'll focus on how string concatenation works, specifically using the + operator, the += operator, and the concat() method.

The + operator is one of the simplest and most frequently used methods to concatenate strings. It allows you to join multiple strings or combine strings with variables that hold text.

Here's an example:

let firstName = "John";

let lastName = "Doe";

let fullName = firstName + " " + lastName; // John Doe

If you need to add or append to an existing string, then you can use the += operator. This is helpful when you want to build upon a string by adding more text to it over time.

Here's an example of appending one string to another using the += operator:

let greeting = 'Hello';

greeting += ', John!';

// greeting is now the string Hello, John!

It is important to remember that strings are immutable which means once a string is created you can not alter it.

In this case, the original string of Hello is not modified. Instead, greeting now references the new string of Hello, John!

Another way you can concatenate strings is to use the concat() method.

Before we begin learning about the concat() method, it is important to first understand what a method and a function are at a higher level.

In programming, a function is a reusable block of code that performs a specific task and can be called with various inputs. A method, on the other hand, is a type of function that is associated with an object, meaning it operates on the data contained within that object.

let str1 = 'Hello';

let str2 = 'World';

let result = str1.concat(' ', str2); // Hello World

In this example, we use the concat() method to join str1, a space (' '), and str2 into a single string.

To conclude, + operator is best for simple concatenation, especially when you need to combine a few strings or variables.

The += operator is useful when building up a string step by step or appending new content to an existing string variable.

Finally, the concat() method is beneficial when you need to concatenate multiple strings together.

1. What is the role of semicolons in JavaScript, and programming in general?

In JavaScript, and many other programming languages, semicolons help delineate statements and improve code readability. In JavaScript, a semicolon (;) is used to indicate the end of a statement.

Just as a period (.) marks the end of a sentence in English, a semicolon signifies the end of an executable line of code. This allows the JavaScript engine to distinguish between separate commands.

For example:

let variableOne = 5; // Declare a variable and assign a value

let variableTwo = 10; // Declare another variable and assign a value

In this code, the semicolons at the end of each line mark the end of each statement. Without them, the JavaScript engine might have trouble interpreting where one statement ends and another begins.

Semicolons are primarily used to mark the end of a statement. This helps the JavaScript engine understand the separation of individual instructions, which is important for correct execution.

let a = 1; // Statement ends here

let b = 2; // Another statement starts here

While JavaScript has Automatic Semicolon Insertion (ASI) that can add semicolons automatically, explicitly including them improves code clarity and helps prevent errors that may arise from unexpected ASI behavior.

Semicolons are used in many programming languages, including C, C++, and Java.

Semicolons mark the end of statements or instructions, helping the compiler or interpreter parse the code correctly. A compiler translates high-level programming language code into machine-readable code, which creates an executable file.

By clearly delineating statements, semicolons contribute to the readability and maintainability of code. Semicolons help prevent ambiguities in code execution and ensure that statements are correctly terminated.

Semicolons are a fundamental part of JavaScript and many other programming languages.

They mark the end of statements, improve code readability, and help avoid errors related to Automatic Semicolon Insertion. By understanding and using semicolons properly, you can write more reliable and maintainable code.

1. What are comments in JavaScript, and when should you use them?

Comments in programming are used to provide additional context for the code or leave notes for yourself and others. Comments are lines or blocks of text that are ignored by the JavaScript engine when your code is executed. They are there solely for the benefit of people reading the code, whether that's you or someone else.

JavaScript provides two ways to add comments to your code: single-line comments and multi-line comments. Single-line comments are created using two forward slashes (//)

Comments like this are important for those working on teams for two reasons:

Other developers working on the project will understand the purpose of this code.

It helps prevent unnecessary changes or deletions without consulting the team, which could lead to bugs or issues.

Another type of comment is multi-line comments.

Multi-line comments are useful when you need to write longer descriptions, explanations, or notes in your code.

While comments are useful in programming, it is important to avoid over-commenting. You don't need to comment on every single line of code, especially if the code is straightforward and self-explanatory.

It is also important not to use comments to help explain away confusing, overly complicated, or poorly written code. In those situations, it is best to refactor, or change, your code so other developers will better understand what is going on.