Fundamentals of JS

Variables in Javascript

A variable is a named container that stores data or values in memory that can be accessed and manipulated later in the program.

Variable Declaration:

- A declaration is the first step in creating a variable, where the developer sets up a space in memory to store the value that will later be assigned to the variable. The JavaScript keywords "`let`", "`const`", or "`var`" can be used to declare variables.
- Variables in JS are dynamically typed, i.e., The variables do not have any type, but the values have type, and we can reassign values to them at runtime.

Variable Definition:

In JavaScript, a variable definition refers to the process of assigning a value to a variable. A variable is a named container that stores data or values in memory, which can be used later in the program.

```
let age=10;
console.log(age);

age='ten';
console.log(age);
```

Naming Convention

- Variable names can contain letters, digits, underscores, and dollar signs.
- A variable name must begin with a letter, underscore, or dollar sign.

- Variable names are case-sensitive.
- It is recommended to avoid using reserved keywords as variable names, such as `var`, `let`, `const`, `function`, `return`, etc.

Best practices to name a variable

- Use descriptive names that convey the purpose of the variable. This will help you remember what the variable is for later on and make it simpler for other developers to grasp what the variable is for.
- Avoid using single-letter variable names unless they are widely used conventionally. For example: using i in a loop or x, y, and z for coordinates is okay, but it's not okay to use a or b to represent a complex concept.
- Use meaningful names for constants (variables whose value doesn't change). Conventionally, constant names are written in ALL_CAPS_WITH_UNDERSCORES, for example, MAX_WIDTH, COLOR_RED, PI.
- Use consistent naming conventions throughout your code. If you have established a naming convention for variables, stick with it throughout your codebase. This will make your code more readable and easier to maintain.
- Use only acronyms and abbreviations that are well-known and understood. For example, it's okay to use ‘btn’ for buttons if it's a widely used convention, but it's not okay to use ‘nm’ for names.
- Be mindful of the length of variable names. Long variable names can make your code harder to read, so aim for concise but still descriptive names.

The most commonly used naming convention includes:

- **Camel Casing**:
    - This means starting with a lowercase letter and capitalising the first letter of each subsequent word. For example, firstName, totalSales, employeeSalary.
- **Snake Casing:**
    - When writing in "snake case," the initial letter of each word is written in lowercase, and each space is substituted with the underscore (_) character. For example: first_child_age.

let, const, and var

- `let` and `const` are two new methods for declaring variables in JavaScript that were added to ES6. The var keyword was the sole way to declare a variable prior to this introduction.
- We will discuss various differences between these keywords in upcoming lectures. Currently, we may distinguish between these keywords based on their level of strictness.
    1) `var` is the least strict of the three, allowing reassigning values to the variable. This declaration can lead to unexpected results and is generally considered bad practice, so it is rarely used
    2) `let` and `const` are more strict than var. Variables declared using let can be reassigned values similar to the var declarations.
    3) `const` is the most strict of the three. When a variable is declared with const, it cannot be reassigned to a new value. This prevents accidental reassignments and makes the code more predictable.
- let and const are recommended instead of var because they have more predictable behaviour and can help prevent bugs in your code.
- An example to depict the declarations using let, var, and const is shown below.

```
var number = 10;

let text = "Coding Ninjas";

const PI = 3.14;
```

Comments

Comments in JavaScript are lines of text that are not executed as part of the program. They are used to add descriptive notes to the code for the programmer or to temporarily remove code without deleting it.

Single-line and multi-line comments:

1. Single-line comments start with // and continue to the end of the line. For example,

```
// This is a single-line comment in JavaScript
```

2. Multi-line comments start with /* and end with */. They can span across multiple lines. For example,

```
/* This is a
multi-line
comment in JavaScript */
```

Data types

- A data type is a classification of the type of data that can be stored and manipulated within a program.
- Each data type has its own characteristics and methods for manipulation.

Classification of data types:

Data Types are broadly classified into the following two categories:

1. **Primitive data types**: These basic data types represent a single value. JavaScript has seven primitive data types. They include:

   A. **number:** represents both integer and floating-point numbers in JavaScript.

   B. **string:** represents textual data in JavaScript and can be enclosed in single quotes, double quotes, or backticks. In JavaScript, there are several ways to represent strings:

      - Single quotes ('string'): Strings can be enclosed in single quotes.
      - Double quotes ("string"): Strings can also be enclosed in double-quotes.

- Template literals/backticks (`string`): Template literals were introduced in ECMAScript 6 and allowed strings to be enclosed in backticks. They are mostly used to represent a string literal that can contain placeholders for variables, making it easier to concatenate strings and variables. In addition, the backtick notation also allows for multiline strings.

Example:

```
// using Single Quote('')
let myString='Coding Ninjas!';

// using Double Quotes("")
let my_String="Coding Ninjas!";

// using Backticks(``)
let String=`Coding Ninjas!`;
let myTemplateString=`Coding
Ninjas!`;
```

C. **boolean:** represents a logical value, which can be either true or false.

D. **null:** represents the intentional absence of any object value.

E. **undefined:** represents a variable that has been declared but has not been assigned a value.

F. **bigint:** represents integers with arbitrary precision. These values usually have 'n' at the end of the number. For example: `let num = 10n;` Here, num is of bigint data type

G. **symbol:** represents a unique value that can be used as a key for object properties.

2. **Object data types:**

An object is a non-primitive data type representing a collection of related properties and methods. It can be thought of as a container for related data and behaviour, similar to an object in the real world.

`typeof` operator

In JavaScript, the `typeof` operator can be used to determine the data type of a value.
The use of `typeof` operator has been described in the examples mentioned below.

```
let myNumber = 42;
console.log(typeof myNumber); // Outputs "number"

let myString = "Hello, world!";
console.log(typeof myString); // Outputs "string"

let myBoolean = true;
console.log(typeof myBoolean); // Outputs "boolean"

let myNull = null;
console.log(typeof myNull); // Outputs "object" (this is a known bug in JavaScript)

let myUndefined;
console.log(typeof myUndefined); // Outputs "undefined"

let myBigInt = BigInt(9007199254740991);
console.log(typeof myBigInt); // Outputs "bigint" (introduced in ES2020)

let mySymbol = Symbol("mySymbol");
console.log(typeof mySymbol); // Outputs "symbol" (introduced in ES6)
```

Special features of JavaScript

In JavaScript, `Number.MAX_VALUE` and `Number.MIN_VALUE` are properties of the Number object that represent the maximum and minimum finite values that can be represented in JavaScript.

- **`Number.MAX_VALUE`** is the largest finite number that can be represented in JavaScript. It has a value of approximately 1.7976931348623157e+308, which is equivalent to 2^1024 - 2^971.
- **`Number.MIN_VALUE`** is the smallest positive number that can be represented in JavaScript. It has a value of approximately 5e-324, which is equivalent to 2^-1074.
- It's important to note that the `Number.MIN_VALUE` is not the smallest number that can be represented in JavaScript. In fact, JavaScript also has a special

value called -`Infinity` that represents negative infinity, which is smaller than any finite number.

Here's an example of how to use `Number.MAX_VALUE` and `Number.MIN_VALUE` in JavaScript:

```
console.log(Number.MAX_VALUE); // Outputs 1.7976931348623157e+308
console.log(Number.MIN_VALUE); // Outputs 5e-324
```

Basic Operators

Operators are symbols or keywords that perform operations on one or more values, called operands and produce results.

Different types of operators:

1. **Arithmetic Operators:** These operators perform basic arithmetic operations on numerical values. Examples include addition (+), subtraction (-), multiplication (*), division (/), increment(++), decrement(--), and modulus (%).
    - The **modulus operator (%)** is represented by the percent sign (%) and is used to find the remainder of the division operation between two numbers.
    - The ++ and -- operators are called increment and decrement operators, respectively. They are used to increase or decrease the value of a variable by 1. Here's how they work:
    - **Increment Operator (++):** The increment operator ++ is used to increase the value of a variable by 1. It can be used before or after the variable name. When used before the variable name, it is called the pre-increment operator; when used after the variable name, it is called the post-increment operator.
    - **Decrement Operator (--):** Similarly, the decrement operator -- is used to decrease the value of a variable by 1. It can also be used before or after the variable name. When used before the variable name, it is

called the pre-decrement operator, and when used after the variable name, it is called the post-decrement operator.

- Example:

```
// increment operator
let x = 5;
x++; // equivalent to x = x + 1;
console.log(x); // Output: 6

// decrement operator
let y = 10;
y--; // equivalent to y = y - 1;
console.log(y); // Output: 9
```

2. **Assignment Operators:** These operators are used to assign a value to a variable. Examples include assignment (=), addition assignment (+=), subtraction assignment (-=), multiplication assignment (*=), division assignment (/=), and modulus assignment (%=).

    - The +=, -=, *=, /=, and %= operators are shorthand assignment operators that combine an arithmetic operation with an assignment. They are used to modify the value of a variable in a concise and readable way.

    - Examples:

```
let x = 5;

let x = 5;
x += 2; // It means x=x+2. x is now 7

let x = 5;
x -= 3; // It means x=x-3. x is now 2

let x = 5;
x *= 4; // It means x=x*2. x is now 20

let x = 10;
x /= 2; // It means x=x/2. x is now 5

let x = 10;
x %= 3; // // It means x=x%3. x is now 1
```

3. **Comparison operators:** These operators compare two values and return a Boolean value (true or false) based on the comparison result. Examples include equality (==), strict equality (===), inequality (!=), strict inequality (!==), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=).

equality (==) vs strict equality (===)

- There are two equality operators in JavaScript: == (equality operator) and === (strict equality operator). While both operators are used to compare values, they have some differences.
- The == (equality operator) compares the values of two operands for equality after converting them to a common type. If the values are equal, it returns true.
- The === (strict equality operator) also compares the values of two operands for equality but without any type conversion. It returns true if the values are equal and have the same type.
- In general, using the strict equality operator === in JavaScript is recommended, as it avoids any unexpected type conversions that can occur with the equality operator ==.
- Example:

```
// Using equality operator
5 == "5"; // true

// Using strict equality operator
5 === "5"; // false
```

Falsy values

- Falsy values are values that evaluate as false when coerced to a boolean. This can be useful in conditional statements or in boolean operations.

- In JavaScript, the values `false`, `0`, `0n` (BigInt zero), `""` (empty string), `null`, `undefined`, and `NaN` (Not-a-Number) are considered falsy.

4. Logical Operators:

- These operators perform logical operations on Boolean values and return a Boolean value as the result. Examples include logical AND (`&&`), logical OR (`||`), and logical NOT (`!`).
    - **&& (logical AND)** - returns true if both operands are true; otherwise returns false.
    - **|| (logical OR)** - returns true if at least one operand is true; otherwise, returns false.
    - **! (logical NOT)** - returns true if the operand is false and returns false if the operand is true.
- Logical operators are often used in conditional statements to evaluate multiple conditions or invert a condition's result. They can also be used to create complex boolean expressions.

4. Bitwise Operators:

- Bitwise operators perform operations on the binary representation of numeric values. These operators are often used to manipulate the individual bits in a number. Examples include bitwise AND (`&`), bitwise OR (`|`), bitwise XOR (`^`), bitwise NOT (`~`), left shift (`<<`), right shift (`>>`), and zero-fill right shift (`>>>`).
    - **& (bitwise AND) -** performs a bitwise AND operation on two values, resulting in a new value with a 1 in each bit position where both values have a 1.
    - **| (bitwise OR) -** performs a bitwise OR operation on two values, resulting in a new value with a 1 in each bit position where at least one of the values has a 1.
    - **^ (bitwise XOR) -** performs a bitwise XOR operation on two values, resulting in a new value with a 1 in each bit position where the two **values** have different bits.
    - **~ (bitwise NOT) -** performs a bitwise NOT operation on a value, resulting in a new value where each bit is inverted.

- **<< (left shift) -** shifts the bits of a value to the left by a specified number of positions, resulting in a new value.
- **>> (right shift) -** shifts the bits of a value to the right by a specified number of positions, resulting in a new value.

- While bitwise operators can be useful in certain situations, they are less commonly used in JavaScript than in other programming languages.

Precedence and Associativity: Link

Notes on Operators: Link

Type coercion

- Type coercion occurs when JavaScript automatically converts a value from one type to another during an operation. It looks for the feasibility of the operation and performs conversion accordingly.
- '+' operator: concatenates string with numbers by converting number to string whereas '-', '*' ,'/', '%',++ etc. operator converts strings to numbers.
- For example, if you try to add a string and a number together using the + operator, JavaScript will automatically convert the number to a string and concatenate the two values.

  Example:

```
let a = "Hello";
let b = 123;
let c = a + b; // Result: "Hello123"
```

  In this example, the number `123` is coerced into a string so that it can be combined with the string value "`Hello`".

Type conversion

- Type conversion is the process of changing the type of a value from one type to another. Several methods can be used in JavaScript to convert values to different types.

1. **Converting to String**

    - You can use the `String()` function or the `toString()` method to convert a value to a string.
    - For example:

```
let num = 123;
let str1 = String(num); // Result: "123"
let str2 = num.toString(); // Result: "123"
```

    - "Both `String()` and `toString()` are used for type conversion to string in JavaScript, but `String()` can convert any data type to a string while `toString()` cannot convert null and undefined values to a string."
    - Example:

```
// using String()
let myNull1 = null;
let myUndefined1 = undefined;
let nullString1 = String(myNull);
let undefinedString1 = String(myUndefined);
console.log(nullString); // output: null
console.log(undefinedString); // output: undefined


// using toString()
let myNull = null;
let myUndefined = undefined;
let nullString = myNull.toString();
let undefinedString = myUndefined.toString();
console.log(nullString); // TypeError: Cannot read property 'toString' of null
console.log(undefinedString); //TypeError: Cannot read property 'toString' of undefined
```

2. **Converting to Number**

    To convert a value to a number, you can use the

    - `Number()`

- `parseInt()`
- `parseFloat()`
- (+)Unary operator

For example:

```
let str = "123";
let num1 = Number(str); // Result: 123
let num2 = parseInt(str); // Result: 123
let num3 = parseFloat("123.45"); // Result: 123.45
```

Note that `parseInt()` and parseFloat() are used for converting strings specifically to integers or floating point numbers, respectively.

- In JavaScript, the **unary '+' operator** can be used to convert a value to a number. When used before a value, it attempts to convert the value to a number type.
  Here's an example:

  ```
  let str = "123";
  let num = +str; // Result: 123
  ```

  Note that the unary + operator can also be used to perform mathematical operations on values, such as addition or subtraction. However, it's important to be careful when using the + operator for concatenation, as it can sometimes lead to unexpected behaviour.

3. **Converting to Boolean**

To convert a value to a boolean, you can use the `Boolean()` function or the !! operator. For example:

```
let num = 123;
let bool1 = Boolean(num); // Result: true
let bool2 = !!num; // Result: true

let str = "";
let bool3 = Boolean(str); // Result: false
let bool4 = !!str; // Result: false
```

Note that any value that is "falsy" (such as 0, "", null, undefined, and NaN) will convert to false when converted to a boolean, while any other value will convert to true.

Conditional statements

Conditional statements in JavaScript are a type of control flow statement that allows you to execute different blocks of code based on different conditions or inputs. JavaScript's most common conditional statements are the `if`, `if-else`, `else-if`, and `switch` statements.
By using conditional statements, you can make your code more dynamic and responsive to user input and other factors.

if-else:

A brief explanation of this conditional statement:

- **if statement:** The if statement executes a block of code if a specified condition is true.
  Example:

```
let num = 5;

if (num > 0) {
  console.log("The number is positive.");
}
```

- **if-else statement**: The if-else statement executes one block of code if a condition is true and another block of code if the condition is false.
  Example:

```
let num = 4;

if (num % 2 === 0) {
  console.log("The number is even.");
} else {
  console.log("The number is odd.");
}
```

- **else-if statement:** The else-if statement specifies a new condition to test if the first condition in the if statement is false.
  Example:

```
let num = 75;

if (num < 50) {
  console.log("The number is less than 50.");
} else if (num < 100) {
  console.log("The number is between 50 and 100.");
} else {
  console.log("The number is greater than or equal to 100.");
}
```

- Note that an if statement can exist alone in JavaScript. It allows you to execute a block of code if a certain condition is true. If the condition is false, nothing happens.
- However, an else statement cannot exist alone, and an else-if statement must always be part of an if statement or an if-else statement, as mentioned earlier.

Ternary operators:

- The ternary operator, also known as the conditional operator, is a shorthand way of writing an if-else statement in JavaScript. It is a concise way of writing a conditional statement that returns one of two values depending on a condition.
- **The general syntax of a ternary operator is:**

```
condition ? value1 : value2;
```

Here, the condition is a boolean expression that is evaluated. If the condition is true, value1 is returned; otherwise, value2 is returned.

if-else vs ternary operator:

- Consider the following if-else statement:

```
let isSunny = true;
let message;

if (isSunny) {
  message = "It's a sunny day!";
} else {
  message = "It's not a sunny day.";
}
```

- This example can be simplified using a ternary operator as shown below:

```
let isSunny = true;
let message = isSunny ? "It's a sunny day!" : "It's not a sunny day.";
```

The ternary operator can make your code more concise and easier to read, but it should be used judiciously. It can become difficult to read if the condition and values are too complex.

Switch statements:

The `switch` statement is a control flow statement that allows you to execute different blocks of code depending on the value of a variable. It is an alternative to using a series of if-else statements.

Implementation of switch statements:

- The switch statement starts with a `switch` keyword followed by a set of parentheses. Inside the parentheses, you specify the variable that you want to test.
- After the parentheses, you add a set of curly braces that contain a series of `case` statements and a `default` statement.
- Each case statement tests for a specific value of the variable, and if the value matches, the corresponding block of code is executed. If none of the case statements matches the variable's value, the code's default block is executed.

Break statement in JavaScript:

In JavaScript, the `break` statement in switch is used to terminate the switch statement. When the `break` keyword is encountered inside the switch statement, the program control immediately exits the statement and continues with the code after the statement.

Here's an example:

```
let color = "red";

switch (color) {
  case "red":
    console.log("The color is red");
    break;
  case "blue":
    console.log("The color is blue");
    break;
  case "green":
    console.log("The color is green");
    break;
  default:
    console.log("The color is not red, blue, or green");
    break;
}
```

Default statement in JS:

- In a switch statement, the `default` statement serves as a backup choice. When none of the cases in the switch statement matches the given condition, the code inside the default statement is executed.
- The default statement is optional and can be placed anywhere in the switch statement, but it is usually placed at the end.

Summarizing it:

In this lecture, we covered several topics related to variables and data types in JavaScript. They are listed below:

- We learned about variables in JavaScript and how they can be used.
- We explored the different types of variables available in JavaScript.
- We learned how to add comments in JavaScript to improve the readability of our code.
- We learned about different data types available in JavaScript, such as Number, String, Boolean, Null, Undefined, and Object.
- We learned some interesting facts about the Number data type.
- We learned about Basic operators in JavaScript, including arithmetic, comparison, and logical operators were discussed.
- We learned the concept of operator precedence, which determines the order in which operators are evaluated.
- Finally, we learned about conditional statements in JavaScript, such as the if-else and switch statements, which allow us to execute different code blocks based on certain conditions.