Fundamentals of JS-I

Code Structure

The first thing we'll study is the building blocks of code.

Statement

Statements are syntax constructs and commands that perform actions. We've already seen a statement, alert('Hello, world!'), which shows the message "Hello, world!".

We can have as many statements in our code as we want. Statements can be separated with a semicolon.

For example, here we split "Hello World" into two alerts:

```
alert('Hello'); alert('World');
```

Semicolon

A semicolon may be omitted in most cases when a line break exists.

This would also work:

```
alert('Hello')
alert('World')
```

In most cases, a newline implies a semicolon. But "in most cases" does not mean "always"!

There are cases when a newline does not mean a semicolon. For example:

```
alert(3 +
1
+ 2);
```

Comments

As time goes on, programs become more and more complex. It becomes necessary to add comments which describe what the code does and why.

Comments can be put into any place of a script. They don't affect its execution because the engine simply ignores them.

One-line comments start with two forward slash characters //.

The rest of the line is a comment. It may occupy a full line of its own or follow a statement.

```
// This comment occupies a line of its own
alert('Hello');

alert('World'); // This comment follows the statement
```

Multiline comments start with a forward slash and an asterisk `/*` and end with an asterisk and a forward slash `*/`.

Like this:

```
/* An example with two messages.
This is a multiline comment.
*/
alert('Hello');
alert('World');
```

Variables

Most of the time, a JavaScript application needs to work with information. Here are two examples.

- An online shop – the information might include goods being sold and a shopping cart.
- A chat application – the information might include users, messages, and much more. Variables are used to store this information.

A Variables

A variable is a “named storage” for data. We can use variables to store goodies, visitors, and other data.

To create a variable in JavaScript, use the let keyword

The statement below creates (in other words: declares) a variable with the name “message”:

```
let message;
```

Now we can store some information on these variables

```
let message;

message = 'Hello'; // store the string 'Hello' in the variable named message
```

Variable Naming

- Variable names must start with a letter, an underscore (_) or a dollar sign ($).
- Variable names cannot contain spaces.
- Variables cannot be the same as reserved keywords such as if or const.
- By convention, JavaScript variable names are written in camelCase.
- Variables should be given descriptive names that indicate their content and usage (e.g. sellingPrice and costPrice rather than x and y).

- As JavaScript variables do not have set types, it can be useful to include an indication of the type in the name (e.g. orderNumber is obviously a numeric ID, whereas order could be an object, a string or anything else).

LET KEYWORD

Let keyword is used to declare variables in javascript. They were introduced in ECMA6

Syntax:

```
let message;
message = "let are used to store variables"
```

Variables defined by let cannot be redeclared in the same scope but can be reassigned.

```
Let x= 8; x=16;
Console.log (x);
```

The output for the above code will be 16 since you are redeclaring the variable x.

VAR KEYWORD

Var keyword is used to declare variables.

Syntax:

```
Var message = "var is used to declare variables"
console.log("message")
```

Var allows readeclaration as well as reassign of variables.

CONST KEYWORD

Const keyword is used to declare variables.Variables which are defined using const can not be changed after assigning.

Syntax:

```
const message = "var is used to declare variables"
console.log("message")
```

Const does not allows readeclaration as well as reassign of variables.

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:

   - **number**: represents both integer and floating-point numbers in JavaScript.
   - **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!`;
```

- ❖ **boolean**: represents a logical value, which can be either true or false.
- ❖ **null**: represents the intentional absence of any object value.
- ❖ **undefined**: represents a variable that has been declared but has not been assigned a value.
- ❖ **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
  - ❖ 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.

JavaScript Primitive Wrapper Types

JavaScript provides three primitive wrapper types: Boolean, Number, and String types.

The primitive wrapper types make it easier to use primitive values including booleans, numbers, and strings.

See the following example:

```
let language = 'JavaScript';
let s = language.charAt(4);
console.log(s);  // Script
```

Code language: JavaScript (javascript)

In this example, The variable `language` holds a primitive string value. It doesn't have any method like charAt `()`. However, the above code works perfectly.

Methods in Strings

Strings are useful for holding data that can be represented in text form. Some of the most-used operations on strings are to check their length, to build and concatenate them using the + and += string operators, checking for the existence or location of substrings with the indexOf() method, or extracting substrings with the substring() method.

toLowerCase()	Converts a string to lower case
toUpperCase()	Converts a string to Upper Case
includes()	Returns a boolean value if a string is contained by another string
charAt()	Returns the character at a specified inde

- length is property of strings, not a method. It returns the number of characters in a string.
- `let str = "Hello, World!"; console.log(str.length); // Outputs: 13`
- So, it is accessed as str.length without parentheses, unlike methods that are invoked with parentheses like str.toUpperCase().