Connecting Intelligence

Institute of Computer Engineering Technology

Plus Content of iCET Certified Master Program

Project Class

JavaScript Revesion Notes

Week 01

Prepared By - Ravindu Samarawickrama

❖ What is JavaScript?

JavaScript is a ***lightweight***, ***interpreted***, or ***just-in-time compiled*** programming language with ***first class functions***. As well as it is ***prototype-based***, ***multi-paradigm***, **single-threaded**, **dynamic language**, supporting **object oriented**, **imperative** and **declarative** (functional programming) language.

(Source - developer.mozilla.org)

❖ Special Points

- JavaScript ignores the spaces, tabs and newline that appear in JavaScript programs.
- Simple statements are generally followed by a semincolon character, just as they are in java. However, JavaScript allows us to omit the semicolon when we write statments in seprate lines.
- JavaScript is case-sensitive language. This means that the language, keywords, variables, function names and any other identifiers must always be typed with a consistent capitalize of a letter.

❖ JavaScript Data Types

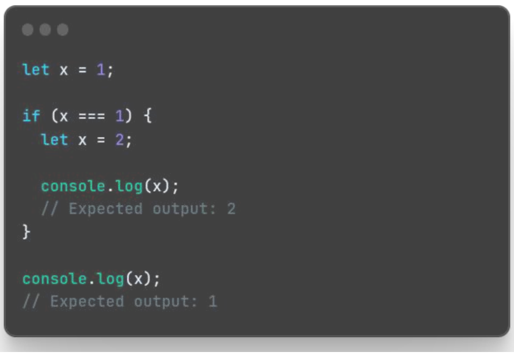
Data Types are the values that can be represented and manipulated in a programming language. JavaScript allows us to work with three primitive data types.

01. Numbers -> 100,102.36,
02. Strings -> “This is sample text”
03. Boolean -> true or false
04. null -> A special keyword denoting a null value
05. undefined -> property whoose value is not defined

❖ Variables in JavaScript.

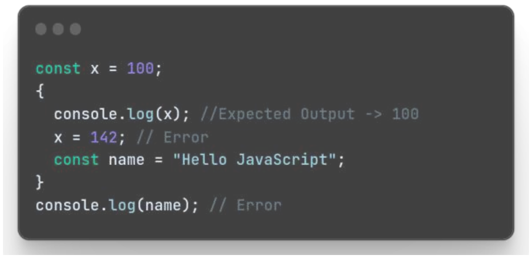
There are 03 types of varibles in JavaScript. `let` and `const` basically replace `var`.

01. let - the let declaration declares a block-scoped local variable, optionally initializing it to a value.



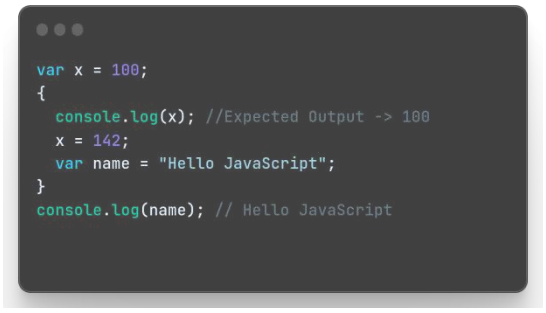
```
let x = 1;

if (x === 1) {
  let x = 2;

  console.log(x);
  // Expected output: 2
}

console.log(x);
// Expected output: 1
```

02. const - const declaration create block-scoped constant, much like variables declares using the let keyword. The value of the constant variable can't be changed through reassignment.



```
const x = 100;
{
  console.log(x); //Expected Output -> 100
  x = 142; // Error
  const name = "Hello JavaScript";
}
console.log(name); // Error
```

03. var - the var statement declares a function-scoped variable, optionally initializing it to a value.



```
var x = 100;
{
  console.log(x); //Expected Output -> 100
  x = 142;
  var name = "Hello JavaScript";
}
console.log(name); // Hello JavaScript
```

❖ Special Points on JavaScript variables

- We should not use any javaScript keywords when declaring variables.
- JavaScript variables should not start with numerals or any other character except letters and underscore.
- JavaScript variables are case senstive. ("Name" and "name" two different variables.)

❖ Reserved Keywords in JavaScript (From W3Schools)

abstract	arguments	await
break	byte	case
char	class*	const
debugger	default	delete
double	else	enum*
export*	extends*	false
finally	float	for
goto	if	implements

in	instanceof	int
let	long	native
null	package	private
public	return	short
super*	switch	synchronized
throw	throws	transient
try	typeof	var
volatile	while	with

❖ JavaScript Operators

Operators are special symbols used to perform special operations on operands in values and variables. Javascript supports the following types of operators.

01. Arithmatic Operators
02. Comparison Operators
03. Logical or Relational Operators
04. Assignment Operators.
05. Condition (or Ternary) Operators

❖ Arithmatic Operators

+ (Addition)	Add two operands (Numbers, String, Variables) **Ex** 10 **+** 20 gives 30
- (Subtratcion)	Subtarct second opearand from the first **Ex** 20 - 10 will give 10
*** (Multiplication)**	Multiply One operand by other **Ex** 10 * 2 gives 20
/ (Division)	Divide one operad by other **Ex** 20 / 2 gives 10
% (Modules)	Gives the reminder of division **Ex** 10 % 6 gives 4
++ (Increment)	Increase integer value by one **Ex** A++ gives 10.

-- (Decrement)	Decrease integer value by one **Ex** A- - gives 8

- Special Point
  - JavaScript uses “+” Operator for both addition and concatenation. Numbers are added and Strings are concatenation.

- Comparison Operators

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

- Logical Operators

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true

||	or	(x == 5 || y == 5) is false
!	not	!(x == y) is true

❖ Bitwise Operators

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shifts left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off
>>>	Zero fill right shift	Shifts right by pushing zeros in from the left, and let the rightmost bits fall off

❖ Assignemnt Operators

Assignment operators assign values to the javascript variables.

Operator	Example	Same As
= (Simple Assignment)	x = y	x = y
+= (Add and Assignment)	x += y	x = x + y
-= (Subtract and Assigment)	x -= y	x = x - y
***= (Multiply and Assignment)**	x *= y	x = x * y
/= (Divide and Assignment)	x /= y	x = x / y
%= (Modules and Assignment)	x %= y	x = x % y
****= (Powered and Assignment)**	x **= y	x = x ** y

❖ typeof Operator

The **typeof** operator is used to determine the data type of a single operand. It takes a single operand of any type and returns a string indicating what type it is. For instance, if the operand is a number, the **typeof** operator will return "number". If it's a string, the operator will return "string", and if it's a boolean value, it will return "boolean". The operator returns true or false based on the evaluation of the operand's data type.

❖ String Declaration, Useful methods in String, String concatenation

Strings are text. In JavaScript, there are many ways to manipulate text/Strings.

- *Creating string in JavaScript.*

```
const string = "The revolution will not be televised.";
console.log(string);
```

- *Single Quotes vs Double Quotes*

In JavaScript, you can choose single quote or double quotes to wrap your strings in Both of the following will work okay :

```
const sgl = 'Single quotes.';
const dbl = "Double quotes";
console.log(sgl);
console.log(dbl);
```

But you cam’t mix single quote and double quote in the begining and end.

```
const text = “Hello World’;
```
This will give error.

So It means, You should end the quote that you start. As if you start begining from the single quote then you should end the text from the single quote. in the middle you can use double quote as below.

```
const sglDbl = 'Would you eat a "fish supper"?';
const dblSgl = "I'm feeling blue.";
console.log(sglDbl);
console.log(dblSgl);
```

- *Escaping characters in a String.*

If we want to add one single quote in the middle of single quoted String we should do it like below.

```
const text = ‘I\’m Ravindu’;
console.log(text);
```

We should use \ (backslash) before the single quote or any escape character.

- *Concatenating Stirngs*

Concatenating strings means just adding texts together. But in advanced we can use *template literals* to add Strings. Template Literals are just strings but instead of using single or double quote we use backticks (`) to declare it.

```
const greetinsg = `Hello`;
```

This is normal string but we can insert variables inside the strings by wrpping from ${} characters. When the program is executed, the varible value will automatically inserted.

```
const name = “Ravindu”
const hello = `My name is ${name}.`;
console.log(hello); // My name is Ravindu.
```

- *Useful String methods.*

  `const text1 = “Hello World”;`
  consider above code line for below examples.

a. **finding the length,**
```
console.log(text1.length); // 12
```

b. **Retreive a special character in string**
```
console.log(text1[1]); // e
```

c. **Testing of string contacins substring.**
```
console.log(text1.includes(‘Hello’)); // true
```

d. **Finding position of a substring in a string.**
```
console.log(text.indexOf(‘World’));  // 7
```

e. **Extracting substring from a string**
```
console.log(text1.slice(5)) //w World
```

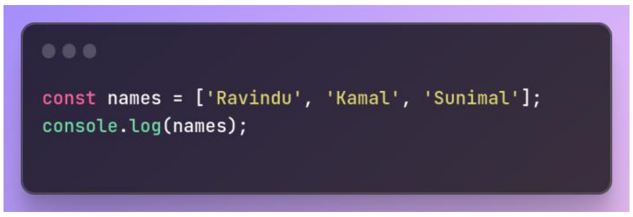
f. **Changing case**
```
console.log(text1.toLowerCase()); // hellow world
console.log(text1.toUpperCase()); // HELLOW WORLD
```

❖ Arrays in JavaScript

Arrays means storing list of data items under a single variable name. After stroing many data items in a single varible name, we can retreive, update, delete thoese data.

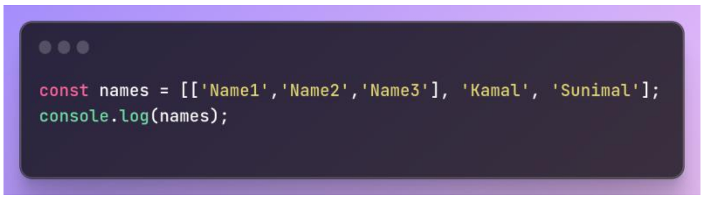
▪ Creating Array

Arrays consist of square brackets([]) and items that are seprated by commans.



```
const names = ['Ravindu', 'Kamal', 'Sunimal'];
console.log(names);
```

We can create array of arrays as below.



```
const names = [['Name1','Name2','Name3'], 'Kamal', 'Sunimal'];
console.log(names);
```

❖ Control Flows in JavaScript

01. Block Statements

We use block statements to group statements. The block statment is delimited by {} Curly Braces.

```
{
    statement01...
    statement 02...
    statement 03...
}
```

02. Conditional Statements

conditional statements are set of commands that are use to specified condition.

a) *if...else statement*

We can use if else statement to check logical conditions. if the condtion true, the statements inside the if block will run and we can use optional else statement to run specified codes if the condition is false.

```
if (condition) {
  statement1;
} else {
  statement2;
}
```

b) *Nested if else*

We can compound the statements using *"else if"* have multiple conditions tested in sequence, as follows.

```
if (condition1) {
  statement1;
} else if (condition2) {
  statement2;
} else if (conditionN) {
  statementN;
} else {
  statementLast;
}
```

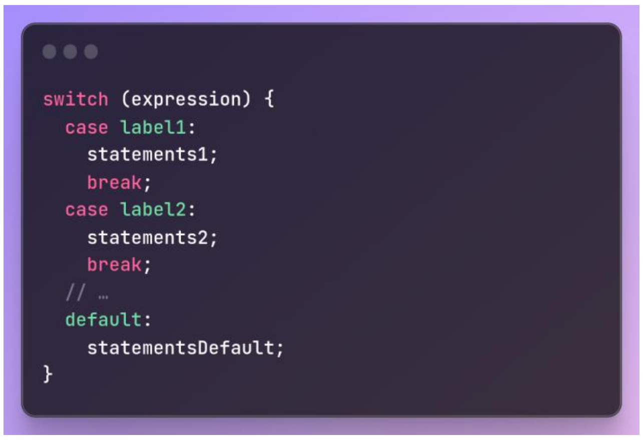
c) *Flasy Values*

Following value evaluate to false. (Also known as falsy values)

false, undefined, null, 0, NaN, empty string (“”)

d) *switch Statement*

Switch statement allows us to listdown many posibilities of a condition and if one of the case meet the condition result, it will execute.



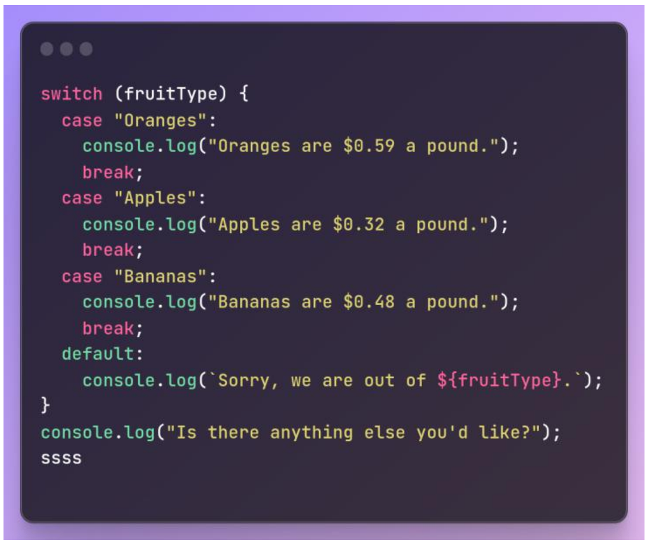
```
switch (expression) {
  case label1:
    statements1;
    break;
  case label2:
    statements2;
    break;
  // …
  default:
    statementsDefault;
}
```

Pattern of JavaScript switch execution

01. JavaScript first look for the case clause which match with the condition result.
02. If the matching clause found, then it will execute the statments inside the clause.
03. If no matching clase found, program will look for “default” clause and it will execute.
04. If no “default” clause found, program will contineu the execution after end of the switch.

e) Break statement

The optional ‘break’ statement linked to each case option ensure that once the appropriate case has been executed, the program will immediately exit the ‘switch’ statement and proceed to next line of code.



```
switch (fruitType) {
  case "Oranges":
    console.log("Oranges are $0.59 a pound.");
    break;
  case "Apples":
    console.log("Apples are $0.32 a pound.");
    break;
  case "Bananas":
    console.log("Bananas are $0.48 a pound.");
    break;
  default:
    console.log(`Sorry, we are out of ${fruitType}.`);
}
console.log("Is there anything else you'd like?");
ssss
```

❖ Loops and Iterations

01. For statement

The for loop in JavaScript will contineu to repeat until the set of conditions evalutates to false. This loop structure is similar to those used in Java and C, providing a familiar experience for developers.

```
for (initialization; condition; afterthought){
  statement
}
```

02. Do-while statement

The do-while statement repeats until a specified condition evaluates to false.

```
do
  statement
while (condition);
```

In here, statement is always run/executed before the condition is checked. If the condtion true, the statement will execute again and again until condition is false.

03. While statement

While statement executes until its condition is false.

```
while (condition){
      statement
}
```

When the condition of the loop is no longer true, the statements within it will cease execution, and control will be transferred to the next statement after the loop. Before each iteration, the condition is evaluated. If it is true, the statements within the loop will be executed and the evaluation will be repeated. If the condition is false, the loop will end and the program will continue with the next statement after it.

04. For...in Statement

The for...in statement iterates a specified variable over all the enumerable properties of an object.

```
for (variable in object)
  statement
```

05. For...of Statement

The for...of statement creates a loop iterating over iterable object (Array, Map, Set) invoking a custom iteration hook with statement to be executed for the value of each distinct.

```
for (variable of object)
  statement
```

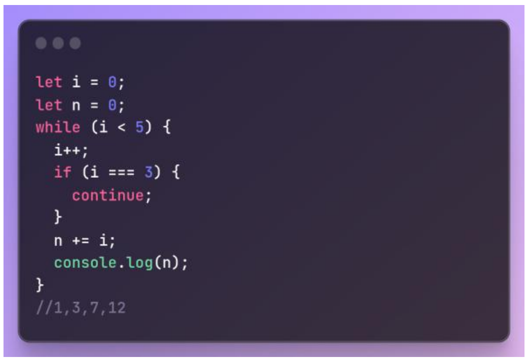
07. Labeled Statement

We can use labeled statement to label a loop or switch. Then we can use contineu, break keyword along with these labeled loops's name.

```
label :
       statement
```

08. Contineu Statement

The contineu statement can be used to restart while,do0whiel,for or labeled statement.



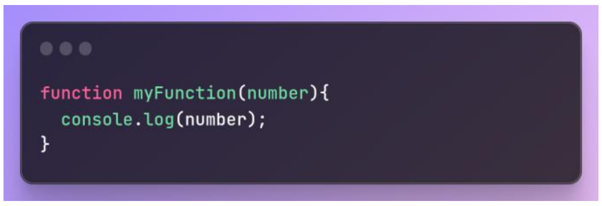
```
let i = 0;
let n = 0;
while (i < 5) {
  i++;
  if (i === 3) {
    continue;
  }
  n += i;
  console.log(n);
}
//1,3,7,12
```

❖ Functions in JavaScript

A JavaScript function is a block of code that performs a specific task. It's defined using the function keyword, a name, and parentheses that can include parameters. Functions can be called to execute the code inside, and can return a value using return. Functions make code reusable, modular, and easier to maintain.

- Declaring Function

function should be declared using `'function'` keyword and should have a name for the funtions. below shows the sample declaration of a function in javascript.



```
function myFunction(number){
  console.log(number);
}
```

- Function Expression / Arrow Functions

Arrow functions are a different way of creating functions in JavaScript. Besides a shorter syntax, they offer advantages when it comes to keeping the scope of the this keyword(see here). Arrow function syntax may look strange but it's actually simple.

```
function callMe(name) {
      console.log(name);
}
```

which you could also write as:

```
const callMe = function(name) {
      console.log(name);
}
```

becomes:

```
const callMe = (name) => {
      console.log(name);
}
```

Important:

When having no arguments, you have to use empty parentheses in the function declaration:

```
const callMe = () => {
      console.log('Max!');
}
```

When having exactly one argument, you may omit the parentheses:

```
const callMe = name => {
      console.log(name);
}
```

When just returning a value, you can use the following shortcut:

```
const returnMe = name => name
```

That's equal to:

```
const returnMe = name => {
      return name;
}
```