# Javascript-19th june and 20th June

18 June 2024 10:18

1. Introduction to Javascript
2. Introduction about NodeJs
3. Setting up NodeJS environment to run programs
4. How to run a Basic javascript program
5. Variable Declarations in Javascript
6. Scopes of Variable declarations
7. DataTypes in Javascript
8. What are functions in javascript and how to write a function.

==========

Javascript

----> language
-------> Its helps us add interactivity
--------> HTML and CSS
    ----> Layout
--->functionality
    ---validation
    ---scrolling
    ---clicking
    ---dragging
    ---more….

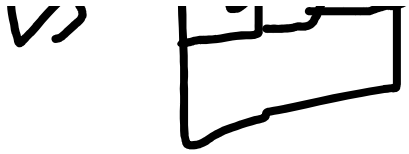

==========

Javacript

--browser engine

--Angular/React/vue
---> compiled--> Javascript

==> Google--> engine--Parser



==> Google--> engine--Parser
---NodeJs
--Javascript run time environment

1.Install NodeJs
2.CodeRunner
==========

Variable Declarations
---

Let
Const
Var
---
Not Defined  vs Undefined

Not Defined---> Variable Not even declared
Undefined---> value hasn't been assigned

```
var a=220
let aa=223
const aaab=225
let value222
console.log(a)
console.log(aa)
console.log(aaab)
console.log(value222)
```

---

Let ---block Scoped
Var---globalScope
Const ---block Scoped

Var

--allows re-declaration
---allow re-assigning
---> globalScope

Let
--->allows re-assigning
-->block Scope

Const
---> block scope
----> It doesn't allow re-declaration and re-assigning

| | global scoped | function scoped | block scoped | reassignable | redeclarable | can be hoisted |
|---|---|---|---|---|---|---|
| var | + | + | - | + | + | + |
| let | - | + | + | + | - | - |
| const | - | + | + | - | - | - |

==========

## Variable Types

**Primitive DataTypes**
**----SingleValue**

**Non-primitiveDataTypes**

**--Complex --Objects,Arrays**

```
// primitive dataTypes
```

```
// non-primitive dataTypes

//primitive
let name="Sirish"// data
let num=203
let bool=true
// non primitive
let inpArray=[2,24,5,6] //--1mb
// 50 mb
// out of space
let inpArray1=["sa","ss","w33"]
let obj={
        name:"Sirish",
        place:"Hyderbad"
}
```

Let var const

DataTypes
Bool
String
Number
Undefined
Null
Symbol

```
let a=123;
console.log( typeof a)
let name1="Sirish";
console.log(typeof name1)
let isPresent =true;
console.log(typeof isPresent)
let randomValue
console.log(typeof randomValue)
console.log(first)
```


Stack
--fixed memory
--last in first out

Heap
---dynamically memory will be allocated
--- It uses reference pointer to the objects created in the memory

```
let a=123;
console.log( typeof a) //Number
let name1="Sirish";
console.log(typeof name1) //string
let isPresent =true;
console.log(typeof isPresent)//boolean
let randomValue
console.log(typeof randomValue) //undefined

console.log(first)
ReferenceError: first is not defined
```

Not Defined  vs Undefined

Not Defined---> Variable Not even declared
Undefined---> value hasn't been assigned

==========

ReferenceError: first is not defined
SyntaxError: Unexpected token 'const'

==========

Functions

--A function is a re-usable code
Which helps us to do a task or complete a program
+ ---> function (1 ,3) ==> 4
* --> function(5,6)===> 30

Login /logout / change settings
Calling an api

==========

Notes

Pre-requisites for setting up local environment

1. Install Nodejs locally from https://nodejs.org/en/download/source-code
2. Install VS code
3. Install code runner from extensions tab

4. Create a file with .js extension
5. Write the code and run the program

==========

## Var vs let vs Const

- var declarations are globally scoped or function scoped while let and const are block scoped.
- var variables can be updated and re-declared within its scope; let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.
- They are all hoisted to the top of their scope. But while var variables are initialized with undefined, let and const variables are not initialized.
- While var and let can be declared without being initialized, const must be initialized during declaration.

**Reference link**: https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/

## Primitive and reference values

JavaScript including primitive and reference values.
JavaScript has two different types of values:

- Primitive values
- Reference values

Primitive values are atomic pieces of data while reference values are objects that might consist of multiple values.

Stack and heap memory
When you declare variables, the JavaScript engine allocates the memory for them on two memory locations: stack and heap.
Static data is the data whose size is fixed at compile time. Static data includes:

- **Primitive values** (null, undefined, boolean, number, string, symbol, and BigInt)

- **Reference values** that refer to objects.

Because static data has a size that does not change, the JavaScript engine allocates a fixed amount of memory space to the static data and stores it on the stack.
For example, the following declares two variables and initializes their values to a literal string and a number:

```
let name = 'John';
let age = 25;Code language: JavaScript (javascript)
```

Because name and age are primitive values, the JavaScript engine stores these variables on the stack as shown in the following picture:

Note that strings are objects in many programming languages, including Java and C#. However, strings are primitive values in JavaScript.

Unlike the stack, JavaScript stores objects (and functions) on the heap. The JavaScript engine doesn't allocate a fixed amount of memory for these objects. Instead, it'll allocate more space as needed.
The following example defines the name, age, and person variables:

```
let name = 'John';
let age = 25;
let person = {
  name: 'John',
  age: 25,
};Code language: JavaScript (javascript)
```

Internally, the JavaScript engine allocates the memory as shown in the following picture:
In this picture, JavaScript allocates memory on the stack for the three variables name, age, and person.
The JavaScript engine creates a new object on the heap memory. Also, it links the person variable on the stack memory to the object on the heap memory.
Because of this, we say that the person variable is a reference that refers to an object.

## Dynamic Properties

A reference value allows you to add, change, or delete properties at any time. For example:

```
let person = {
  name: 'John',
  age: 25,
```

};
*// add the ssn property*
person.ssn = '123-45-6789';
*// change the name*
person.name = 'John Doe';
*// delete the age property*
delete person.age;
console.log(person); Code language: JavaScript (javascript)

Output:
{ name: 'John Doe', ssn: '123-45-6789' }Code language: CSS (css)

Unlike a reference value, a primitive value cannot have properties. This means that you cannot add a property to a primitive value.
JavaScript allows you to add a property to a primitive value. However, it won't take any effect. For example:
let name = 'John';
name.alias = 'Knight';
console.log(name.alias); *// undefined*
Code language: JavaScript (javascript)
Output:
undefinedCode language: JavaScript (javascript)
In this example, we add the alias property to the name primitive value. But when we access the alias property via the name primitive value, it returns undefined.
==========

JavaScript has the primitive data types:
1. null
2. undefined
3. boolean
4. number
5. string
6. symbol – available from ES2015
7. bigint – available from ES2020

and a complex data type object.

JavaScript is a dynamically typed language, meaning that a variable isn't associated with a specific type. In other words, a variable can hold a value of different types. For example:
let counter = 120; *// counter is a number*

counter = false;  *// counter is now a boolean*
counter = "foo";  *// counter is now a string*Code language: JavaScript (javascript)
To determine the current type of the value stored in a variable, you use the typeof operator:
let counter = 120;
console.log(typeof(counter)); *// "number"*
counter = false;
console.log(typeof(counter)); *// "boolean"*
counter = "Hi";
console.log(typeof(counter)); *// "string"*Code language: JavaScript (javascript)
Output:
"number"
"boolean"
"string"Code language: JSON / JSON with Comments (json)

## The undefined type

The undefined type is a primitive type that has only one value undefined. By default, when a variable is declared but not initialized, it defaults to undefined.
Consider the following example:
let counter;
console.log(counter);  *// undefined*
console.log(typeof counter); *// undefined*Code language: JavaScript (javascript)
In this example, the counter is a variable. Since counter hasn't been initialized, it is assigned the value undefined. The type of counter is also undefined.
It's important to note that the typeof operator also returns undefined when you call it on a variable that hasn't been declared:
console.log(typeof undeclaredVar); *// undefined*Code language: Java

## The boolean type

The boolean type has two literal values: true and false in lowercase. The following example declares two variables that hold the boolean values.

let inProgress = true;
let completed = false;
console.log(typeof completed); // *boolean*Code language: JavaScript (javascript)
JavaScript allows values of other types to be converted into boolean values of true or false.
To convert values of other types into boolean values, you use the Boolean()function.
The following table displays the conversion rules:

| **Type** | **true** | **false** |
|---|---|---|
| string | non-empty string | empty string |
| number | non-zero number and Infinity | 0, NaN |
| object | non-null object | null |
| undefined | | undefined |
| | | |

**For example:**
console.log(Boolean('Hi'));// *true*
console.log(Boolean('')); // *false*
console.log(Boolean(20)); // *true*
console.log(Boolean(Infinity)); // *true*
console.log(Boolean(0)); // *false*
console.log(Boolean({foo: 100})); // *true on non-empty object*
console.log(Boolean(null));// *false*Code language: JavaScript (javascript)

The symbol type
JavaScript introduced a new primitive type in ES6: the symbol. Unlike other primitive types, the symbol type does not have a literal form.
To create a symbol, you call the Symbol function as follows:
let s1 = Symbol();Code language: JavaScript (javascript)
The Symbol function creates a new unique value every time you call it.
console.log(Symbol() == Symbol()); // *false*Code language: JavaScript (javascript)

The bigint type
The bigint type represents the whole numbers that are larger than $2^{53} – 1$.
To form a bigint literal number, you append the letter n at the end of the number:
let pageView = 9007199254740991n;

```
console.log(typeof(pageView)); // 'bigint'
```

## The object type

In JavaScript, an object is a collection of properties, where each property is defined as a key-value pair.

The following example defines an empty object using the object literal.

syntax:
let emptyObject = {};Code language: JavaScript (javascript)
The following example defines the person object with two properties: firstName and lastName.

```
let person = {
    firstName: 'John',
    lastName: 'Doe'
};
```

A property name of an object can be any string. You can use quotes around the property name if it is not a valid identifier.
For example, if the person object has a property first-name, you must place it in the quotes such as "first-name".

A property of an object can hold an object. For example:

```
let contact = {
    firstName: 'John',
    lastName: 'Doe',
    email: 'john.doe@example.com',
    phone: '(408)-555-9999',
    address: {
        building: '4000',
        street: 'North 1st street',
        city: 'San Jose',
        state: 'CA',
        country: 'USA'
    }
}Code language: JavaScript (javascript)
```

The contact object has the firstName, lastName, email, phone,

and address properties.
The address property itself holds an object that
has building,  street, city, state, and country properties.
To access an object’s property, you can use

- The dot notation (.)
- The array-like notation ([]).

The following example uses the dot notation (.) to access
the firstName and lastName properties of the contact object.
console.log(contact.firstName);
console.log(contact.lastName);Code language: CSS (css)
If you reference a property that does not exist, you’ll get
an undefined value. For example:
console.log(contact.age); *// undefined*Code language: JavaScript
(javascript)
The following example uses the array-like notation to access
the email and phone properties of the contact object.
console.log(contact['phone']); *// '(408)-555-9999'*
console.log(contact['email']); *// 'john.doe@example.com*