JavaScript

Variables:

It’s used to store a data temporarily.I js,we can declare variable using 3 keywords : var,let,const.

- Cannot be a reserved keyword
- Cannot start with a number
- Cannot contain space or –
- They are case sensitive

```
let var1="Navya";
console.log(var1);

```

Datatypes in js:

1. Primitive datatypes
2. Reference datatypes

Primitive datatypes :

- String
- Number
- Boolean
- Undefined [default value of variables in js]
- Null

Reference datatypes :

- Objects
- Arrays
- Functions
- Date
- Math

Dynamically typed :

JavaScript is a dynamically typed language, meaning variables can change their type at runtime. We can use *typeOf* operator to check the type of a variable [typeOf of null will be an object].

Objects :

Objects are collection of key value pairs [here,keys are properties and its value can be of any type].

```
Eg ,   let person={
       name:"Navya",
       age:23
       }
```

Arrays :

Its used to store a collection of values[its used to represent a list of items]. Objects of the array and size of the array is dynamic. Array is an object.

```
Eg,    let colors=['red','blue',1]
```

Functions :

Fundamental building blocks in js.Its basically a set of statements that performs a task or calculates a value.

```
Eg,    function add(a,b){
        return a+b
        }
       console.log(add(5,2))
```

Flow of controls :

Decision Making : Conditional statements(if,if else, if elseif else, ternary)

Looping : Create loops in set of codes (for,while,do while, for…in, for…of)

Switching : Its used to control loops (break & continue)

Hoisting in js :

It refers to the process whereby interpreter appears to move the declarations to the top of the code before execution.

Eg, person(“Navya”)

```
function person(name){
console.log(name)
}
```

Js only hoists declarations, not initializations. The variable will be undefined until the line where its initialized is reached [var supports hoisting but let & const does’nt].

Function expressions are also not hoisted.

Scope in js :

Scope defines the accessibility and visibility of variables in different parts of the code.

Global scope : Variables that are declared outside a function or a block. It can be accessed from any where in the program.

Local/function scope : Variables that are declared inside the function, can only be accessed within that function.

Block scope : let and const are block scope, where as var is not block-scoped.

Error Handling :

1.try….catch syntax

```
try{
  try this code
```

}

catch(error){

error handling

}

First the code in try is executed,if there is no error, catch is ignored else catch is executed. Try…..catch works synchronously.

2.throw statement

The throw statement lets you create custom errors and throw them manually.

Events :

An event is a signal that something has happened[click,submit,load,keyup]..It triggers an action.

Event loop :

JavaScript has an event loop,call stack and a call queue + other APIs

- Sometimes js code can take a lot of time and this can block page rerender.
- Js has asynchronous callbacks for non blocking behaviour.
- Js runtime can only do one task at a time.
- The rest are queued to the task queue waiting to be executed.
- Js has a runtime model based on an event loop,which is responsible for executing the code,collecting and processing events and executing queued subtasks.
- The event loop pushes the task from task queue to call stack.

Promises in js :

A promise is a promise of code execution.The code either executes or fails,in both the cases it will be notified[it either resolves or rejects].

The consuming code can then receive the final result of promise by then & catch.

Asynchronous Programming :

Async/await

Callbacks

Promises

Closure :

A closure allows an inner function to access variables from its outer (enclosing) function, even after the outer function has returned.

Closure is a function along with its lexical environment.

Call stack :

It's used to keep track of data execution in a program. It works in LIFO (last in first out),where last function added to the stack is first one to get executed, and once it completes its execution its removed from the stack.

JavaScript Engines :

Js engine takes human readable code as input. It undergoes three major steps.

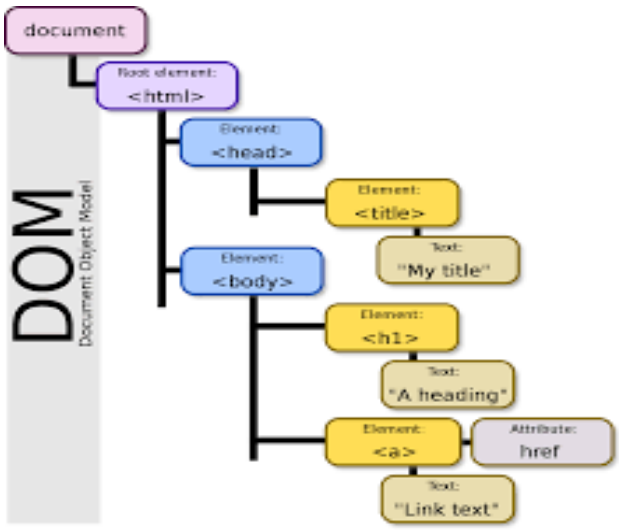
1. Parsing : It breaks the code into tokens and syntax parser converts the code into AST(Abstract syntax tree).This AST is then passed into compilation phase.
2. Compilation : Modern Js engine follows JIT Compilation(js engine can use interpreter along with compiler).It interprets code line by line (byte code), while compiler optimises code as much as it can (machine code).Then it is executed

Js engine consists of *memory heap*[it's the memory allocated for all variables and functions created in the code], *call stack* and *garbage collector*[it tries to free up memory heap whenever possible(removes that are no longer needed)It follows Mark & sweep algorithm[It starts from root,marking reachable objects,and all unmarked objects(it does'nt have any references) will be removed]].

3. Execution : The engine runs the machine code.
   Popular js engines:
   v8(chrome,node),spidermonkey(firefox),chakra(edge).

JavaScript DOM(Document Object Model) :

Web browser constructs the DOM when it loads an HTML document, and structures all elements in a tree like representation.Js can access the DOM to dynamically change the content, structure and style of a webpage.



setTimeout :

It allows us to run a function once after a given interval of time.

```
Eg,  setTimeout(function(){
        alert("Inside setTimeout")
```

},3000)

setInterval :

It runs a function every time after a given interval of time.

Eg, setInterval(function(){

alert("Inside setInterval")

},5000)

== & === :

They are comparison operators but they differ in how they compare values.

== tries to convert values to the same type before comparing.

Eg, console.log(5 == '5'); here, it converts string ‘5’ to a number 5, since values are equal it will return true.

=== does not perform type conversion and compares values as they are.

Eg, console.log(5 === '5'); here,we are comparing a number and a string, so it return false. This means that both the value and the type must be the same for the comparison to return true.

typeOf :

It’s used to determine the type of a variable or value.

Eg , console.log(typeof 42); here, it will return number

console.log(typeof 'Hello'); here, it will return string

Expression vs Statement :

Expression always returns a value. It can be assigned to a variable or used as a function argument.

Eg, let a = 5; here, ‘5’ is an expression

A statement is a line of code that performs an action but doesn't necessarily produce a value. Statements often contain expressions

Eg, if(a>3){ ,here if is a statement that controls flow.

Console.log(a)

}

IIFE – Immediately invoked function expression :

It's a js function that runs as soon as it is defined.

Syntax: (function() {

// Code inside the IIFE

})(); The func is treated as expression ,because it is wrapped in parenthesis. Variables inside an IIFE are not accessible outside the function.

Modules :

Modules in JavaScript are a way to structure and organize code into separate files and reusable pieces, which can then be imported and used in other parts of the application.

It uses export and import keywords for ES6

Export : This allows modules to be available in other modules.

Import : This allows you to bring in functionality from other modules and use it in the current module.

Uses require(importing) and module.exports(exporting) for commonJs

Message Queue :

A message queue is a form of communication between different parts of a system that allows them to exchange messages asynchronously.

1.Producers : These are the components that create and send messages to the queue.

2. Message Queue : This is the intermediary that stores the messages until they are processed.

3. Consumers : These are the components that retrieve and process the messages from the queue. They may process the messages immediately or at some later time.

Factories :

A factory function is a function that produces and returns an object(it follows camel notation).

```
Eg, function createCircle(radius){
        return {
                radius,
                draw(){
                        console.log(“Draw”)
                        }
                }
                };
                const circle=createCircle(1)
                circle.draw()
```

Constructors :

Constructors in JavaScript are used to create and initialize objects(It follows pascal notation).

```
Eg, function Circle(radius){
        this.radius=radius;
         this.draw=function(){
          console.log("Draw");
        }
        }
      const circle1=new Circle(1) ; console.log(circle1);
```

Here, **new** creates an empty object. Then,it will set the **this** value to point to that newly created object and returns the new object from the function Circle.

Classes :

Classes in JavaScript are a modern, more readable way to create objects and implement inheritance.

Syntax ; class Myclass{

constructor (){….}

method(){……}

}

this :

It refers to the object executing the current function.

Methods :

If a function is part of an object, we call it a method(if a function is defined inside an object).

Recursion :

Recursion is a programming technique where a function calls itself in order to solve a problem.

Eg, function factorial(n) {

```
  if (n === 0) {
    return 1;
  }
  return n * factorial(n - 1);
}
console.log(factorial(5)); //output = 120
```

Pure Functions :

It will always return same output for the same input and will not modify any external state or variables.

Eg, function add(a,b){

return a+b

}

console.log(add(1,2))

Impure functions :

It depends on external variables and data, also modifies external variables and data.

Eg, let count = 0;

function increment() {

count++;;

}

Side Effects :

A side effect occurs when a function interacts with or modifies something outside its scope. Pure functions does'nt have any side effects but impure function does.

Inheritance :

It allows a class to inherit properties of another class.This is done by using **extends** keyword.

Syntax ; class ParentClass {

```
constructor(property) {
    this.property = property;
}
parentMethod() {
console.log(`Parent method`);
}
}
class ChildClass extends ParentClass {
    constructor(property, childProperty) {
        super(property);
```

```
this.childProperty = childProperty;
}
childMethod() {
 console.log(`Child method`);
}
}
```

Polymorphism :

It is the ability to create a variable, a function or an object that has more than one form.

Partial Application :

Its where a function is pre-filled with some of its arguments, returning a new function that takes the remaining arguments.

```
Eg ; function multiply(a, b, c) {
   return a * b * c;
 }
 function partialMultiply(a) {
   return function(b, c) {
     return multiply(a, b, c);
   };
 }
 const double = partialMultiply(2);
 const triple = partialMultiply(3);
 console.log(double(3, 4));
 console.log(triple(3, 4));
```

Currying :

It transforms a function with multiple arguments into a nested series of functions, each taking a single argument.

Eg ; function multiply(a, b, c) {

```
  return a * b * c;
}
function curriedMultiply(a) {
  return function(b) {
    return function(c) {
      return multiply(a, b, c);
    };
  };
}
const multiplyBy2 = curriedMultiply(2);
const multiplyBy2And3 = multiplyBy2(3);
const result = multiplyBy2And3(4);
console.log(result); // Output: 24
```