Lesson 1: JavaScript for beginners

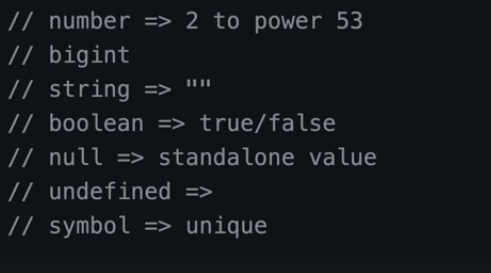
* It is better to learn CSS after JavaScript but it does not mean it’s the only path.
* Not everyone who knows JS can teach JS.
* We are going to learn all the necessary concepts in modern JS with best practices.
* The major aspect of this course is to put confidence of JS. Which is a must in Software Development.
* The best thing to get confidence is to build projects.

Lesson 4: Let, const and var

* Before proceeding make a goal like I am learning JS to build an ecommerce website using Next.js.
* So, in order make an ecommerce I need to store Users information somewhere and that’s where variables come into picture.
* We start by investigation because it is very important to investigate from official documentation in future.
* Every style like title case camel case or title case are all write just go with what you like.
* The only thing is important is variable name must be descriptive and easy to understand.
* Please avoid usage of var because of issue in block scope and functional scope.
* use **let** and **const** only.
* It is possible to create a variable without specifying var, let, or const but avoid this approach also.
* Semicolons are optional it doesn’t matter.

Lesson 5: Datatypes and ECMA standards

**Primitive Data Types**





* JS is backward compatible. So, to use newer version of JS type “use strict”.
* **“use strict”;** means use all the code as a newer version all across the file
* There is no way to say “do not use strict there”.
* By default, many places use it. But explicitly typing it is a good practice.
* **alert(“Hello world”)** is not supported in Node.js use **console.log()** or **console.table()** instead?
* Put Code readability on first priority so anybody can easily understand it.
* The official website of ECMAScript specification is <https://tc39.es/ecma262/>. But we use MDN because official documentation is for people who want to build web browsers.
* According to ECMAScript documentation there are many JavaScript objects like Objects, Date and Time but the good thing is JS don’t have too much data types.
* The range of **number** is 2**53**. We also have **bigint** to store large numbers.
* **Null** is a standalone value and its datatype is **object**. Temperature = null
* **Undefine** = variable not have a value yet. And its datatype is undefined.
* Undefine and null are two different things.
* Symbol, we use it to define uniqueness like in a lot of React components.
* **Non** primitive = objects
* To check data type, we use **typeof**. We can use it as a statement or also as a function.

Lesson 6: Datatype Conversion Confusion

* In JS datatype conversion is a nightmare.
* **Number(“33”)** to **number 33**.
* **Numbers(null**) will return **0**;
* **Number(false)** will return **0**.
* **Number(true)** will return **1**.
* **Number(“33abc”)** it will return **NaN**.
* **Number(“string”)** will return **NaN**.
* **Number(undefine)** as **NAN** as well.
* In JS **floating-point** is a type of **number**.
* **NaN** is also a type ofnumber, also **NaN** is not equal to itself.
* Boolean conversion
* **Boolean(1**) => true
* **Boolean(0)** => false
* **Boolean(“”)** => false
* **Boolean(“Hello”)** => true
* **Boolean([])** => true
* **Boolean([1,2,3])** => true
* **Boolean({})** => true
* **Boolean({a:’apple’})** => true

Lesson 7: Why string to number conversion is confusing

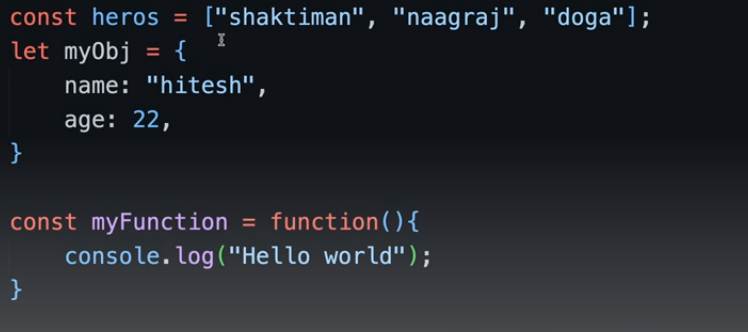
* We are going to learn about some operation that are popular in JS memes.
* There is no integer division operator in JS as I know for now.
* If you try to convert **“1” + 2 + 3** it will convert 2 and 3 to string and the output is **123**
* If you try to convert **1 + 2 + “3”** it will add 1 + 2 and then concatenate 3 output is **33**
* **+true = 1**
* **+”” = 0**
* **++2** = **prefix**, **2++** **postfix** read it about MDN

Lesson 8: Comparison of datatypes

* Everything works as accepted when comparing numbers but the problem begins when you are comparing objects of different data types.
* In JS equality check **==** and comparison **> < >= <=** works differently. Comparison converts **null** to number, treating it as zero. That’s why **null >= true, null > 0** and **null == 0** are false
* Typescript does not allow you to compare different data types.
* == convert string to number and does the conversion after converting its datatype.
* === does not convert data type so 2 === “2” is false. It’s also called strict checking
* Question so if == converting string to number so why not it doesn’t convert null to 0?
* Answer is because it only converts string nothing else.
* comparing objects of different data types is a bad practice kindly avoid it.
* Because clean code is our first priority

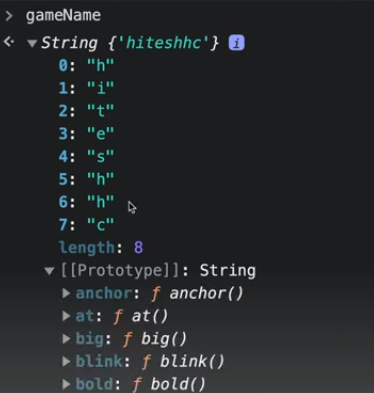
Lesson 8: Data types of JavaScript summary for interview

* According to the official documentation JS has two major types of Datatypes Primitive and Non-Primitive/Reference.
* The difference in how you store it and how you retrieve the value from it.
* Primitive are call by values.
* When you use it, you get a copy of it. So, the changes will affect only that copy.
* **Primitive datatypes are**
  + Number
  + BigInt
  + String
  + Boolean
  + null
  + undefined
  + Symbol
* **Non-primitive** or **Reference** Datatypes
  + You can get the direct access of their memory address.
  + If you want to become a master of JS just master **Objects** and **Browser-Events**
  + Arrays => object
  + Functions => function or function object
  + Objects => objects
* Stack(primitive) Heap (Relative)



Lesson 11: Strings

* 7 methods covered.
* String interpolation **`hello ${name}`** is equal to f-strings in python.
* You can create string by using JS objects, **const name = new Sting(“Pakistan”)** and it will give you access of a lot of methods and also a property of **length**.
* **Substring**, **slice()**, **indexof**, **trim**, **replace**, **includes**, **split**



Lesson 14: Array ([MDN](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array))

* Total 15 methods.
* **Push**, **unshift**, **pop**, **shift**, **includes**, **indexOf**, **at**, **slice(included, Excluded)**, **splice(included, included)**
* **concat** = use to join two array like [1,2,3] + [4] in python, it returns a new array doesn’t affect original.
* **Spread** operator **...** do the same thing but allow to join more than two array [**...**arr1, **...**arr2, **...**arr3]. Note **...**This operator known as Rest operator in function to get multiple arguments like Args or \* in Python.
* **flat**(**infinity**) allows to make nested array as a single array without any nesting by defining levels.
* **Array.isArray**(variable), tells that the object is an array or not.
* **Array.from**(variable) converts any sequence to array.
* **Array.of** same as array from.
* Read more about **isaarray**, **from** and **of** methods.

Lesson 16: Objects

* Total concepts including this and de-structure.
* Object created using a constructor **let user =** **new** **Object()** has ***singleton*** while object created using literal **{}** doesn’t have ***singleton***. There is also a concept of **this** keyword which can access the current context.
* **user.name, user[‘first name’]**
* **Object.keys(object), Object.values(object)**
* You can also define a **function** inside an object.
* **Object.freeze** will freeze the object so you cannot change the Object.
* **Object.assign({}, object1, Object2)** is me thori si confusion hoskti hai.
* **Object.entries** == **dict.items** in python it return. **[[key,value], [key, value]]**
* **tinderUSer.hasOwnProperty(‘name’**) is like **dict.get(key, error)** but it retuns Boolean.

Lesson 18: Objects de-structure

Bht ajeeb sa concept hai ye concept react me kafi use hota hai.

**let course = {**

**‘course’: ‘react for beginners’,**

**‘courseInstructorName: ‘hitesh’,**

**}**

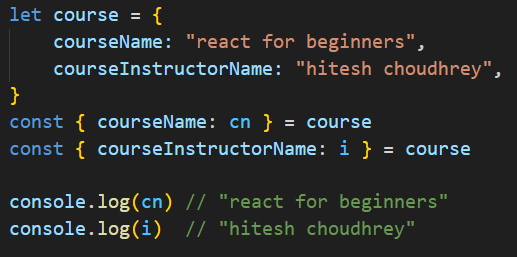
ab bajae **course.courseInstructorName** likhne k hum isko desturcture kr skty han

**const {courseInstructorName} = course** is trhan ya phir

**const {courseInstructorName: short\_name} = course** is tarhan

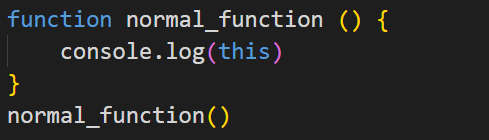
ab hum **course.courseInstructorName** ki bajae sirf **courseInstructorName** ya phir **short\_name** likh skty han

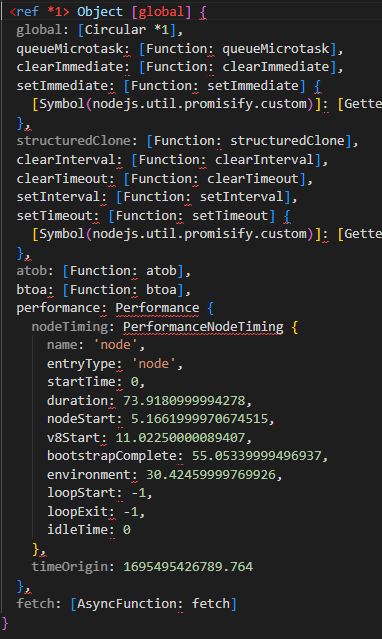
Note: Json or Objects do alg cheezain han Json me “Key” double/single quotes me aygi jab k object me Key without quotes b likh skty han.



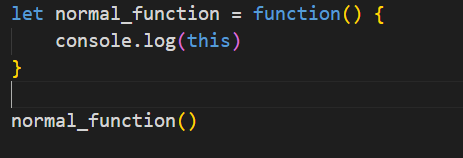
Lesson 23: Functions

1. Checking context **this** inside a regular function.

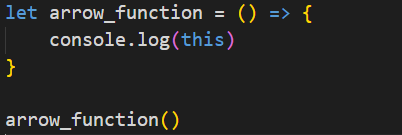




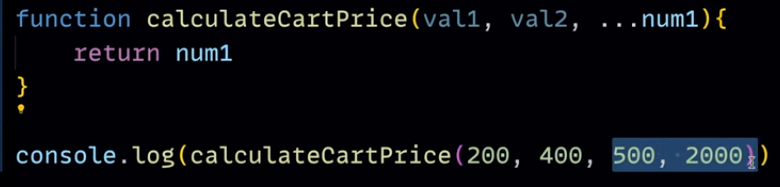
1. The output of the function blow is same as above



1. But arrow function returns an empty **this** object {}

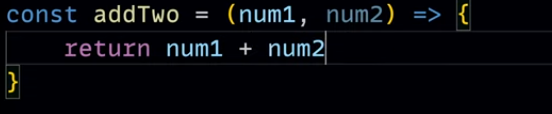


1. Regular function with Rest operator.



1. Examples of an **Arrow function**.

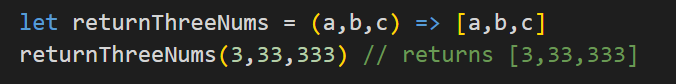
Arrow function with **curly brackets** **{}** needs an explicit return statement

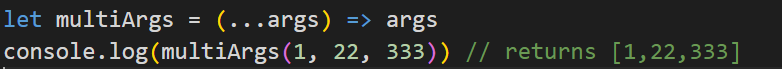
****

1. Blow are the Arrow function with **implicit return** which is used in react.

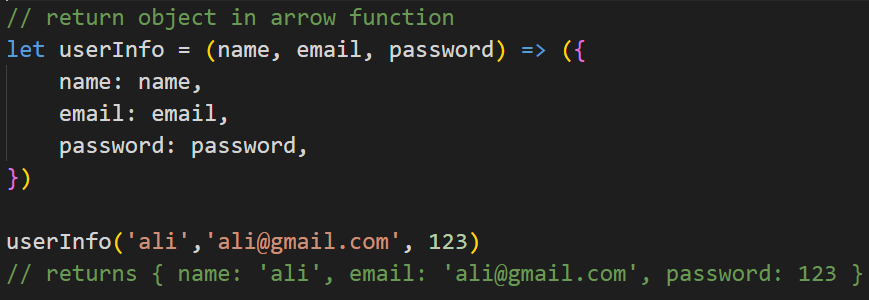








1. Arrow function with implicit return, returning an object.

­­