# Variables

console.log("test ")

const accountId = 23355;

let accountEmail = "siddharthnaudiyal@gmail.com";

var accountPassword = "Sidhu#daf";

accountCity = "Jaipur";

console.table([accountEmail,accountId,accountPassword,accountCity])

**const** means it cant be changed

**let** and **var difference =** we use let instead of var because in modern day javascript it solves a problem of scope (“{ }”) var was like if u change var it also changes var in whole code irrespective of scope but let only assign value at that scope.

**Don’t use var because of issue in block scope and functional scope.**

* You can assign value in variable without let var const too. It is possible in javascript
* And if you **let**  a variable without any value inside it. Node js would show undefined value

# Datatypes

**Number** 2 to power 53

**Bigint**

**String**

**Boolean** true false

**Null** stand alone value like you wont return zero if you temperature app aint working youll use null cuz zero is value

**Undefined**

**Symbol** unique

**Object**

**Typeof**  it tells you type of data types but type of null is **object**

# Data Conversion

let score = 33

let score2 = "45"

console.log(typeof(score))

let valueInNumber = Number(score2)

console.log(valueInNumber)

console.log(typeof valueInNumber)

**Typeof()** to know the type and **Number()** to convert it into number

* Syntax would be N capital or starts with capital
* When changing number if its not a number in string it will return **NaN**(not a number)
* **Null**  would be converted to **0**
* **Undefined** would be converted to **NaN**
* But they all would be typeof **number**

let isLoggedIn = 1

let isBoolean = Boolean(isLoggedIn)

console.log(isBoolean)

* 1 in Boolean is **true**
* **0** in Boolean is **false**
* **“ “**  is **false**
* **“Siddharth”** is **true**

# Conversion

console.log(null > 0)*//false*

console.log(null == 0)*//false*

console.log(null>=0) *//it converts null to zero (true)*

console.log("2"==2) *//true because it converts string to number auto*

console.log("2"===2)*// falase as it checks its data type too*

# Data types Summary

**Primitive Data types –** strings, number, bigInt, Boolean, null, undefined ,symbol.

They all call by value – that means when make changes they are copy not the actual thing in memory

**Non Primitive (Refrence) -**  Array, Object and Functions

JavaScript is a **dynamically typed language** ✅

That means:

* You **don’t need to declare the type** of a variable when you create it.
* The **type is determined at runtime** (when the code runs), not at compile time.
* A variable can hold different types of values at different times.

const score = 100               *// Number datatype*

const scoreValue = 100.3        *// num*

const isLoggedIn = false        *// boolean*

const outsideTemparture = null  *// null*

let userEmail;                  *// undefined*

const value = Symbol('124')

const value2 = Symbol('124')

console.log(value2===value)  *// false they are unique*

**Datatype of null is Object**

# Stack and heap memory

**Stack(**primitive) as a copy data

let myname = "siddharth"

let anotherName = myname

anotherName = "sid"

console.log(myname)

console.log(anotherName)

it doesn’t change the name of my name

and **Heap (** non primitive) original data

let UserOne = {

    email: "siddharth@gmail.com",

    pass: "whatver"

}

let UserTwo = UserOne;

UserTwo.email = "hellw@gmail.com"

console.log(UserOne.email)  *//hellw@gmail*

console.log(UserTwo.email)  *//hellw#gmail*

it has changed cuz its pointing towards that memory not creating a copy

# Strings

Concatination of strings should be done by backstrick ` ` and ${ } to add and conctatinate string

const name = "SId"

const height = 44;

*// let nameHeight = name + height;*

*// console.log(nameHeighta)*

let nameHeight2 = `my name is ${name} and my height is ${height}`

console.log(nameHeight2)

strings is also an object collection of key value pair

let gameScore = new String ("hello")

console.log(gameScore)

defining a string. Normal string also look like this in background

let gameScore = new String ("hello")

console.log(gameScore)

console.log(gameScore[0])

console.log(gameScore.\_\_proto\_\_) *// this is not empty it will show alot of property if  you see this in broweser*

**Strings Methodology**

console.log(a.charAt(2));       *// telss the location of string at index 2*

console.log(a.toUpperCase());   *// converts to uppercase*

console.log(a.indexOf("i"));    *// returns the index of i in a*

**Slicing of Strings**

let newString = a.substring(0,3) *// doesnt obey negative value it will start from start*

console.log(newString)

const anotherString = a.slice(-8,4) *// it will obey negative value as well it will start from the end too*

console.log(anotherString)

**.trim() –** used usually when you are taking a form from user and he fills spaces in front or back unknowingly or knowingly

b = "      hello kutta        "

console.log(b)                  *// output :          hello kuttaa*

console.log(b.trim())           *// output : hello kutta (this removes the spaces from front and back)*

**.replace() –** to replace something in url or something

const url = "https://www.google.com/siddharth%20naudiyal"

console.log(url.replace("%20","-"))

**.include**

console.log(url.includes("sid"))

# Numbers and Maths

let score = 30

console.log(score)

let gameScore = new Number(882) *// this will set this as number and will give prototype in console of browser*

console.log(gameScore)

**Functions for number**

**.tostring ()**

console.log(gameScore.toString())  *// this will help to get all the strings function in number by converting*

**.tofixed()**

console.log(gameScore.toFixed(2)) *// this will show 882.00 means till two decimal places*

usually used in ecommerce websites

**.toprecision()**

let nameScore = 898.93

console.log(nameScore.toPrecision(3))

it will round off till the number you have given.

**.toLocaleString()**

const hundreds = 1000000000

console.log(hundreds.toLocaleString("en-In"))*// for indian decimals en-In is used*

this will put decimals according to digits leaving localestring as it is will give you American decimal system.

# Maths

**Maths** library comes inbuilt in javascript

console.log(Math);

Object [Math] {}

It is object

Go to browser and see more property of maths

**Functions of maths**

**Math.abs()-** absolute value

console.log(Math.abs(-4));*// will turn -4 into 4 but not 4 into -4 it gives the absolute value*

**Math.round() –** math.round off

console.log(Math.round(4.343)); *// output will be 4*

**Math.ceil and Math.floor –** as name suggests ceil means ceiling and floor means ground it will round off number if ceil it will give bigger number and floor will give nearest smaller number

console.log(Math.floor(4.343)); *// output 4*

console.log(Math.ceil(4.343)); *// output 5*

**Most importantly we have :**

**Math.random() –** it will give us a random number between 0 to 1

console.log(Math.random())                      *// this will give random no between 0 to 1*

console.log((Math.random()\*10) + 1)             *// multipy by 10 gives 0 to 10 but we dont need zero as value so +1*

console.log(Math.floor((Math.random()\*10) + 1))*// floor will remove all the decimal value*

to get range we have

const max = 20;

const min = 10;

console.log(Math.floor(Math.random() \* (max - min + 1)+ min))

remember this formula

# Dates

It is object in javascript

const myCreatedDate = new Date(2023, 0 ,11)  *// Wed Jan 11 2023*

console.log(myCreatedDate.toDateString())

in js month starts from **0.**

const myCreatedDate = new Date("2004-11-23")  *// Tue Nov 23 2004,*

console.log(myCreatedDate.toDateString())

months will not start from zero if you put it inside string **yyyy-mm-dd**

let myTimeStamp = Date.now()

console.log(myTimeStamp)*//             this will return time in miliseconds 143554353426556*

console.log(myCreatedDate.getTime())    *//this will convert date in miliseconds 343215145*

**used in time buzzer** to get milliseconds who won or in booking apps you have to compare time in milliseconds.

myCreatedDate.toLocaleString("defualt",{

    weekday:"long";

})

We can customize.toLocaleString with many features.

# Array

Array is collection of multiple data stored in a single variable.

* Array in javascript is resizeable (you can add more items afterwards)
* Can contain different types of data
* 0 based indexing
* It creates shallow copy (heap memory) / same reference point
* Means if you change array you are changing the original.

let myarr = [0,2,3,4,5]

let myarr2 = new Array(1,3,"sid")

console.log(myarr)

console.log(myarr2)

creating array [ ] or new Array( )

when you see it on console you get prototype and inside prototype you get another prototype **object** one.

**Methods in array**

**Arr.push () -**  insert item at last

myarr.push(8)

console.log(myarr)

**arr.pop( ) =** removes the last element in array

**arr.unshift =** adds element to the start of the array we are adivised not to use this because shifting array could be time consuming process consuming and get a load on the memory if we have a big array.

**Arr.shift( )** = it is pop but removes element from the start of the array

**Some checking functions**

**Arr.includes( ) =** tells you that it exist in the array or not. Returns value in Boolean

let array = [1,2,3,5,5,6,"sid", "kutta"]

console.log(array.includes(1)) *// boolean true*

**arr.indexOf() =** tells you the index of the something you put inside the function

console.log(array.indexOf("sid"))

**for conversion**

**arr.join( ) =** it joins converts the array into string and return with comma separated values.

const newArr = array.join()

console.log(newArr)

**Splice and Slice**

console.log("A ",array) *//A  [ 1, 2, 3, 5, 5, 6, 'sid', 'kutta' ]*

console.log("b ",array.slice(0,7)) *//b  [ 1, 2, 3, 5, 5, 6 ]*

console.log(array)*//                [ 1, 2, 3, 5, 5, 6, 'sid', 'kutta' ]*

console.log("c ",array.splice(0,7))*//c  [1 2, 3,5, 5, 6,'sid']*

console.log(array) *//[ 'kutta' ]*

**slice(0,6) –** 0 is the start index and 6 is the 6 places it slices and into the part but doesn’t change the original array

**splice()-** does the same thing but changes the original array.

**Diffrence between push and concat and spread**

**Push =**

let marvelHeros = ["IronMan","Spiderman","Captain"]

let dcHeros = ["batman","superman","flash"]

marvelHeros.push(dcHeros)

console.log(marvelHeros)

[

'IronMan',

'Spiderman',

'Captain',

[ 'batman', 'superman', 'flash' ] **this puts array inside the array.**

]

**Concat**

let marvelHeros = ["IronMan","Spiderman","Captain"]

let dcHeros = ["batman","superman","flash"]

allHeros = marvelHeros.concat(dcHeros)

console.log(allHeros)

**Spread**

Spread means spread example glass being broken into pieces all the water get in same thing

allHeros = [...dcHeros,...marvelHeros]

console.log(allHeros)

syntax [] imagne it like we burst water bubble inside a container array all their elements get collected there **mostly used in.**

**Flat()**

let anotherArray = [1,2,3,4,[5,6,7],[2,3,44,[23434,33]]]

let realArray = anotherArray.flat(Infinity)

console.log(realArray)

this is used when you have array inside array inside array or like a lot of ghichpich

.flat(number of depth) who much you want to flat.

Use.**flat(infinity)** tips and tricks

## Is of from in array

Used in data scraping

console.log(Array.isArray("siddharth")) *// false*

console.log(Array.from("siddharth")) *// ['s', 'i', 'd','d', 'h', 'a','r', 't', 'h']*

Array.isArray() tells you that is it a a**rray or not**

And array.from **coverts** into the array

Gives empty array if it cant covert it to array []

**Of**

let score1 = 100

let score2 = 200

let score3 = 300

let total\_score = Array.of(score1,score2,score3)

console.log("Array of scores are ",total\_score)

# Objects

We can declare objects in two ways litreals and Constructors

When we make object using Constructors it makes a **singleton** ( singleton means this object is unique it is one of its kind)

Litreals don’t create objects as a singleton

const names = {

    name1: "sid",

    name2: "tannu",

    name3: "saksham",

    name4: "sejal"

}

Object

**Accessing**

We got two ways dot method . and [] method

**dot method**

console.log(names.name1)

**bracket method (prefferd)**

console.log(names["name1"])

remember that it should be on strings when you put key there

**putting symbol data type in the object**

*// objects litreals*

const mysym = Symbol("key1");

const names = {

    name1: "sid",

    [mysym]:"this is key1",

    name2: "tannu",

    name3: "saksham",

    name4: "sejal"

}

console.log(names[mysym])

this will return value as a symbol not as a string this is the only way to get it like that.

**To override current value**

names.name1 = "the\_sid\_supreme"

**object.freeze()**

names.name1 = "the\_sid\_supreme"

console.log(names.name1);

Object.freeze(names)

This will make the object freeze meaning no more changes ahead.

**Adding greetings from object**

names.greeting = function () {

    console.log(`hello ${this.name1}`)

}

console.log(names.greeting())

**so** with already existing object we added greeting

with **this.name** and you should write **greeting()** not **greeting** because it will return annonymus function as output.

# Object through constructor

Declaring object through constructor creates a singleton.

**Declaration in constructor**

1. **Singleton**

const TinderApp = new Object(); *// this creates a singleton means its unique object*

1. **Literal**

const tinder = {}

**chdcking somehtin**