Conversion of different data types to number in JS -

```
let string = "33abc"
let stringc = Number (string)
console.log (type of (stringc))  → number
console.log (stringc)  → NaN
```

```
let score = null
let valueInNumber = Number (score)
console.log (type of (valueInNumber))  → number
console.log (valueInNumber)  → 0
```

```
let score = undefined
let valueInNumber = Number (score)
console.log (type of (valueInNumber))  → number
console.log (valueInNumber)  → NaN
```

```
let score = true.
let valueInNumber = Number (score)
console.log (type of (valueInNumber))  → number
console.log (valueInNumber)  → 1
```

"" ⇒ 0
"33" ⇒ 33
"33abc" ⇒ NaN
true ⇒ 1
false ⇒ 0
null ⇒ 0
undefined ⇒ NaN

Type conversion of different data types into boolean.

let isLoggedIn = 1

let booleanIsLoggedIn = Boolean(isLoggedIn)
console.log(booleanIsLoggedIn) → true

1 → true
0 → false

let isLoggedIn = ""
let booleanIsLoggedIn = Boolean(isLoggedIn)
console.log(booleanIsLoggedIn) → false

"" → false
"abc" → true

null → false
undefined → false

Type conversion of different data types to string -

Integer

1 → "1"
0 → "0"

null

null → "null"

Booleans

true → "true"
false → "false"

undefined

undefined → "undefined"

Operations in JS -

```
let value = 3
let negValue = -value

console.log(negValue) → -3
```

```
console.log(2*2) → 4
console.log(2**3) → 2^3 = 8
```

```
console.log(2/3) → 0.66666...
console.log(2%3) → 2
```

```
let str1 = "Hello"
let str2 = " World"
let str3 = str1 + str2
console.log(str3) → Hello World
```

```
console.log("1" + 2) → "12" → Typecasted to string
```



```
console.log("1" + 2 + 2) → 122
console.log(1 + 2 + "2") → 32
```

console.log(3+4*5%3) → 5

console.log(true) → true
console.log(+true) → 1 → The number/datatype after + is converted to an integer and then is logged in the console.
Therefore, true to int is 1 and hence 1 is printed.

console.log(+"") → 0

Comparison of different data types in JS -

console.log(null > 0) → false
console.log(null == 0) → false
console.log(null >= 0) → true

The reason is that an equality check == and comparisons >, <, >=, <= work differently.

Comparisons convert null to a number, treating it as 0.
Therefore,
null > 0 → false.
null >= 0 → true

```
console.log(undefined == 0)
console.log(undefined > 0)
console.log(undefined < 0)
```
} false.

This is because even if we convert undefined to integer, it is always converted to NaN hence any comparison will always give value as false

(===) Strict check in JS -

console.log("2" === 2); → false As both the data types are different.

console.log("2" === "02") → false As both the strings are different.

console.log("2" === "2") → true As both the strings are equal/same.

Primitive vs Non-primitive data type -

The way in which we can store and retrieve the data that is there in the memory made this primitive and non-primitive type categorization.

Primitive (call by value) -

7 types

String, Number, Boolean, null, undefined, Symbol, BigInt

Symbol → to make something unique

BigInt → very big integer values

Non-primitive (Reference type) -

3 types

Arrays, Objects, Functions.

JS does not have float, double, all are numbers

How to declare symbols -

```
const id = Symbol('123');
const anotherId = Symbol('123');
```

→ return type is also Symbol

```
console.log(id === anotherId)
```

↓

false is outputed

Here values stored in both id and anotherId will be different even if same Symbol ('123') is initialized.

If we do console.log(id) and console.log(anotherId), we can't retrieve the value of symbol() stored as symbols are internally designed to be unique and immutable without exposing their internal details. These are primarily used as ~~primary~~ unique keys

Memory in JS (Stack vs Heap)

Stack (Primitive)
Heap (Non-primitive)

Stack memory example -

let myName = "Akshat"

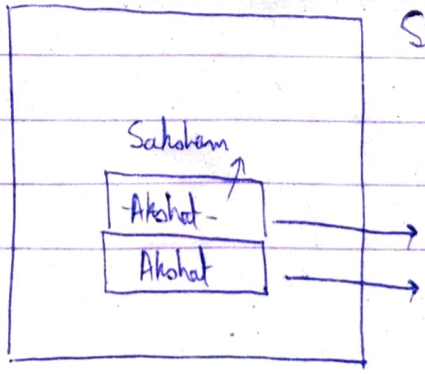
let anotherName = myName

anotherName = "Sakshaam"

console.log(myName) → Akshat
console.log(anotherName) → Sakshaam

↓

Behind the scenes



Here, two memory spaces are there for each variable individually. and hence if change is done on one of the memory other is not affected

Heap memory example –

object ({ })

let userOne = {

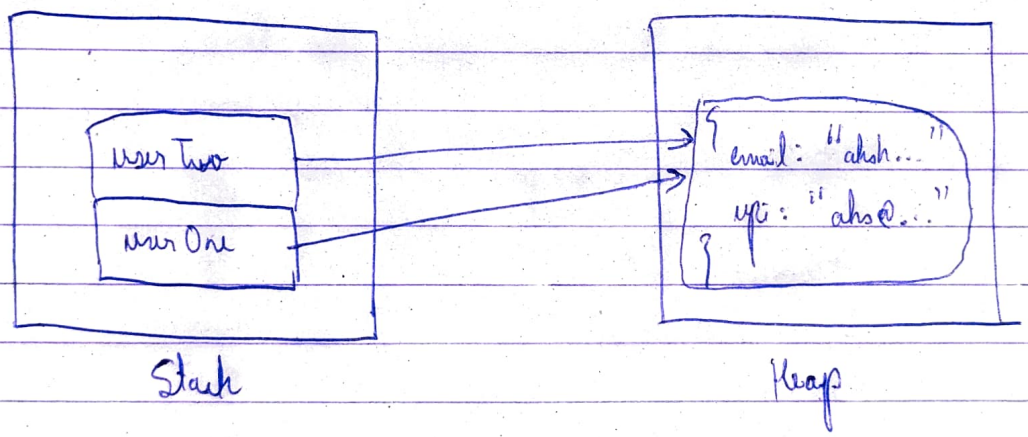
email : "akshatjain@gmail.com",

upi : "aks@upi"

}

object is stored in a variable

let userTwo = userOne



Here, both userOne and userTwo is referring to same piece of code in Heap memory.

userTwo.email = "sakshamn@gmail.com"

console.log(userOne.email) ⟶ sakshamn@gmail.com

```
const name = "hitesh"
const repoCount = 50

console.log(`Hello my name is ${name} and my repo count is ${repoCount}`);
```

```
const getName = new String('akshat')
```

We are using JS objects

Internally, this is how the string is stored. (Object form)

0: "a"
1: "k"
2: "s"
3: "h"
4: "a"
5: "t"

→ In form of key value pairs and not in array.

To access the values,

```
console.log(getName[0]);
```

→ The value stored in key 0 gets printed.

Ways in which strings can be declared.

String Literals -

```
let a = "ABC"
let b = 'ABC'
let c = `ABC`
```

String Objects

```
let d = new String("ABC")
```

Obj → Primitives

```
let strLiteral = strObj.toString()
```

console.log(gameName.__proto__); → { } → Object

↓

gives the prototype of

console.log(gameName.length) → 6

console.log(gameName.toUpperCase()) → AKSHAT → Original string is not changed

console.log(gameName.charAt(2)); → gives the character that has key value of 2 (s)

console.log(gameName.indexOf('a')) → gives the index at which a is present first (i.e., 0)

For slicing the substring from a string --

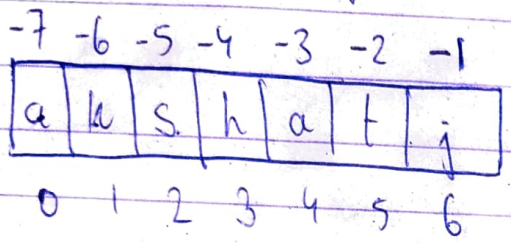
const newString = gameName.substring(0, 4) → aksh

↳ [0, 4]

↳ 4th index is excluded

console.log(newString) → aksh

Slice function internal working -



New Name. slice (2)

↓

Shatj

New Name. slice (-2)

↓

tj

New Name. slice (-3, -1)

↓

at

New Name. slice (-7, -4)

↓

ahs

New Name. slice (-2, 1)

↓

No output as we cannot print string from backwards to forwards.

New Name. slice (2, -1)

↓

shat

trim() in JS -

If in a string we have some spaces included, these spaces can be removed easily by using trim() function.

eg- const newStringOne = " akshat "

const newtrimStringOne = newStringOne.trim();

console.log(newtrimStringOne) → "akshat" → All the spaces are being removed

Internal working

" akshat "

↓

"akshat"

" ak sh at "

↓

"ak sh at"

trim() only removes spaces preceding the string and after the string and not the spaces between the string.

const url = "https://hitesh.com/hitesh%20choudhary"

↳ normal url

const url = "https://hitesh.com/hitesh choudhary"

space gets converted to %20 as browser does not understand spaces in url.

"https://hitesh.com/hitesh%20choudhary"

replace() in JS -

```
url.replace('%20', '-')
```

Value to be searched

value to be replaced with.

o/p - https://hitesh.com/hitesh-choudhary.

includes() in JS

console.log(url.includes('hitesh')) → returns true if 'hitesh' is found, else returns false in url string

console.log(url.includes('akshat')) → false

Splitting a string based upon a particular pattern

```
const getName = "akshat is learning js and reactjs"

const newgetName = getName.split(" ")

console.log(newgetName) ⟶ ['akshat', 'is', 'learning', 'js', 'and', 'reactjs']

console.log(newgetName[2]) → 'learning'

const newgetName = getName.split("")

console.log(newgetName) → ['a', 'k', 's', 'h', 'a', 't', ' ', 'i', 's', ' ', 'l', 'e', 'a', 'r', 'n', 'i', 'n', 'g', ' ', 'j', 's', ' ', 'a', 'n', 'd', ' ', 'r', 'e', 'a', 'c', 't', 'j', 's']

console.log(newgetName[2]) → 's'
```

Math and Numbers in JS -

The main motto of using this is that we can use the inbuilt functions provided by this class.

```
const score = 400
console.log(score)  → 400
```

```
const balance = new Number(100)
console.log(balance)  → [Number: 100]
```

object

Creates an object that can store integer value.

```
console.log(balance.toString().length);  → 3  (As 100 is converted to "100")
console.log(balance.toFixed(2))  → 100.00
```

```
const hundreds = 1000000
console.log(hundreds.toLocaleString())  → 1,000,000
```

represents the given integer value is comma seperated. (US format)

```
console.log(hundreds.toLocaleString('en-IN'))  → 1,00,000
```

represents the given integer value in comma seperated (Indian format)

→ Original value is preserved as integer only.

let myDate = new Date() ← Creates a Date Object

console.log(myDate.toString())

→ Returns Wed Mar 01 2023 format

To create a custom date -

month (0 - Jan)

year ↑ ↑ ↑ day

let myCreatedDate = new Date(2023, 0, 23)

console.log(myCreatedDate.toDateString()); → Mon Jan 23 2023

let myCreatedDate = new Date("2023-01-14")

let myTimeStamp = Date.now() → gives time in milliseconds

console.log(myCreatedDate.getTime()) → gives time in milliseconds

console.log(Date.now()/1000) → gives time in seconds (in decimal values)

↓

console.log(Math.floor(Date.now()/1000));

↓

gives time in seconds (in inte-

let newDate = new Date()

console.log(newDate) → gives the date

console.log(newDate.getMonth()) → gives the month value i.e., (0, 1, 2, 3)

Jan Feb Mar Apr

Arrays in JS -

- Shallow copy is applied in JS
- Size of array is dynamic

Syntax

```
const myArr = [0,1,2,3,4,5]

const myArr2 = new Array(1,2,3,4)
```

Array methods -

myArr.push(6); → Adds an element at the end in the array.

console.log(myArr); → 0,1,2,3,4,5,6

myArr.pop() → removes an element from the end of the array.

myArr.unshift(9) → [9,0,1,2,3,4,5]

myArr.unshift(99) → [99,9,0,1,2,3,4,5]

Adds element in the 0th index and shifts other elements by 1 index.

```
const arr = [1,2,3,4]

arr.shift()

console.log(arr) → 2,3,4
```

Join in JS

```
let arr = [1,2,3,4]
let newarr = arr.join();

console.log(newarr) → "1,2,3,4"

console.log(newarr[1]) → 1

console.log(newarr[0]) → 1
```

String 1,2,3,4 is made from [1,2,3,4]

```
console.log(newarr[8]) → undefined
```

Slice vs Splice in JS

✓ Slice -

```
let arr = [1,2,3,4,5,6]
let newarr = arr.slice(1,3);

console.log(newarr) → [2,3]
console.log(arr)    → [1,2,3,4,5,6]
```

[ ) is applied

original array is not changed

✓ Splice -

```
let arr = [1,2,3,4,5,6]
let newarr = arr.splice(1,3);

console.log(newarr) → [2,3,4]
console.log(arr)    → [1,5,6]
```

both intervals are inclusive

Original array is changed

Operations on array -

```
const marvel_heroes = ["Thor", "Ironman", "Spiderman"]
const dc_heros = ["superman", "flash", "batman"]
```

marvel_heros.push(dc_heros)

console.log(marvel_heros) ⟶ ["Thor", "Ironman", "Spiderman", ["superman", "flash", "batman"]]

Both works the same way

marvel_heros.concat(dc_heros)

Concat() in string -

const allheros = marvel_heros.concat(dc_heros) → Make a new variable and then apply concat on orignal arrays

console.log(allheros) ↘

["Thor", "Ironman", "Spiderman", "superman", "flash", "batman"]

Spread operator

const all_new_heros = [...marvel_heros, ...dc_heros]

console.log(all_new_heros) ↘

["Thor", "Ironman", "Spiderman", "superman", "flash", "batman"]

Both work in similar way, only difference is that by using spread, we can merge many arrays together.

flat()

```
const arr = [1,2,3, [4,5,6], 7, [6,7 ; [4,5]]]

const real = arr.flat( )
                    ↳ depth is the parameter.
```

const real = arr.flat(1) → [1,2,3, 4,5,6,7,6,7, [4,5]]
const real = arr.flat(2) → [1,2,3, 4,5,6,7,6,7, 4,5]
const real = arr.flat(infinity) → opens all the arrays and makes a single one

i.e., [1,2,3,4,5,6,7,6,7,4,5]

Array.isArray() -

Checks whether the following data passed is array or not (returns boolean)

let arr = [1,2,3]

console.log(Array.isArray(arr)) → true
let arr2 = (20
console.log(Array.isArray(arr2)) → false

Array.from() in JS -

from() changes a datatype into array. when the datatype is passed into the function.

let str = "Akshat"

console.log(Array.from(str)) → ["A", "k", "S", "H", A, T]

console.log (Array.from ("AJ")) → ["A", "J"]

Array.of() in JS -

of() can take multiple inputs and then convert them into a single array.

let score1 = 100
let score2 = 200
let score 3 = 300

console.log(Array.of (score1, score2, score 3)) → [100, 200, 300]

Objects in JS :

Object literals -

const JsUser = { } → Object

```
const JsUser = {
    name: "Hitish",
    age: 18,
    location: "Jaipur",
    email: "hitish@google.com",
    isLoggedIn: false,
    lastLoginDays: ["Monday", "Saturday"]
}
```

internally name is "name" i.e., stored as a string.

Accessing the values of object -

① console.log(JsUser.email) → hitish@google.com

Better way of accessing values -

② console.log(JsUser["email"])

This one is better as when key value pair given in object and key is defined as "email": "",

i.e., key will be in the form of string, then we can only access the value of object in that format.

Adding symbols in objects -

```
const mySym = Symbol("key1")

const User = {
    name: "Akshat"
    age: 23
    [mySym]: "mykey1"
}
```

To access the value of symbol,

```
console.log(User[mySym])
```

Freeze in JS

If we want to lock the values of an object, we can use freeze() method so no one can access the value of that object.

```
const User = { name: "Akshat"
               age: 23
             }

Object.freeze(User)
User[name] = "Avinash" ⟶ name won't be changed as freeze() is done
```

Adding values in the object -

```
const tinderUser = {}

tinderUser.id = "123abc"
tinderUser.name = "Sammy"
tinderUser.isLoggedIn = false.

console.log(tinderUser) → {id: "123abc", name: "Sammy", isLoggedIn: false}
```

Nesting of objects -

```
const regularUser = {
                    email: "abc@email.com",
                    fullname: {
                                userfullname: {
                                                firstname: "Raja"
                                                lastname: "Jain"
                                              }
                              }
                   }
```

To access "Raja" we can -

```
console.log(regularUser.fullname.userfullname.firstname); → Raja
```

Merging of two or more objects in one -

(1) const obj1 = { 1: "a", 2: "b", 3: "c" }

const obj2 = { 4: "d", 5: "e" }

const obj3 = Object.assign({}, obj1, obj2)

Target object

Source objects

console.log(obj3) → { 1: "a", 2: "b", 3: "c", 4: "d", 5: "e" }

(2) const obj1 = { 1: "a", 2: "b", 3: "c" }

const obj2 = { 2: "d", 3: "e" }

const obj3 = Object.assign({}, obj1, obj2)

console.log(obj3) → { 1: "a", 2: "d", 3: "e" }

(3) const obj3 = Object.assign({}, obj2, obj1)

console.log(obj3) → { 1: "a", 2: "b", 3: "c" }