**Strings in Js**

**1. Includes()** // Case Sensative, returns boolean

// "aNewString".includes("w", 6));

// Accepts search word and index to search from

**2. IndexOf() || lastIndexOf()** // Returns Index or -1

**3. padEnd() || padStart()** // str3.padEnd(100, ".")

// returns string with length mentioned, by adding, space or mentioned character

// "2".padStart(2, "0") // Adds at padStart

**4. Repeat()** // repeats string for count // 'abc'.repeat(2)

**5. Replace()** // str.replace("dog", "cat")

**6. Slice()** // str.slice(10); // Returns Everything after the index

// str.slice(10, -3); // Returns Everything in between the index, skipping 1st

**7. Split()** // str.split(" ");

// Returns an array of strings populated by splitting the calling string at occurences of the substring

// str.split(" ", 5)); // Returns limited number of splits

**8. .substr** // str.substr("10"); // returns NEW STRING, Everything after the index

// str.substr("10", 10); // returns Everything after the index , with limit set

**9. .substring** // Same as substr but accepts (from, to) WHERE as substr (from, length)

**Arrays in Js**

**1. Push()** Adds one or multiple elements at end of array, method returns new length of array

**2. Pop()** Removes last element of array and returns removed elements value

**3. Unshift()** Adds one or Multiple elements at start of an array, returns new length of array

**4. Shift()** Removes first element of array and returns removed elements value

**5. IndexOf()** Finds Index of searchItem passed

**6. splice()** accepts 2 args, **(indexfrom, numberOfItems);** || Returns removed items

a.splice(2, 0, 4, 5, 6, 7) // first : from index to splice || 2nd number of iteams to splice

all others to be added at given place

**7. slice()** accepts 2 args, **(indexfrom, indexTo);** || Returns removed items

**8. concat()** concats 2 or more arrays into one myArray.concat(array2,array3,...,n)

**9. every() / some()**  every() method checks if all elements in an array pass a test (from function). Returns Boolean

**10. toString()** Converts Arrya to String with COMMA ,

**11. Filter()** array.filter(function(currentValue, index, arr), thisValue) // Returns new Array unlike map

**12. find()** returns the value of the first element in the provided array || array1.find(element => element > 10); Or returns undefined

**13. findIndex()** returns the index of the first element in the array that satisfies the provided testing function. Otherwise, it returns -1

**14. includes()** determines whether an array includes a certain value among its entries, returning true or false

**15. Join()** creates and returns a new string by concatenating all of the elements in an array || Accepts args

**16. LastIndexOf()** arr.lastIndexOf(searchElement[, fromIndex]) || Searches from backword direction

**17. map(**) creates a new array populated with the results of calling a provided function on every element in the

calling array || let new\_array = arr.map(function callback( currentValue[, index[, array]]) {})

**18. Array.reduce()** Reduces numeric array to single digit function(accum,item)

**19. reverse()** reverse is destructive -- it changes the original array.

**20. flat** converts multiple array in array to single level array arr1.flat(1)

**Objects in Js**

**1. Object.assign(target, source)** Returns modified Target || Object.assign({}, obj1);

Clone multi level object let obj3 = JSON.parse(JSON.stringify(obj1));

**2. Object.create(person);** Works same as ```= new person ```

**3. Object.defineProperty()** accepts 3 params. 1. Object name, 2. Property name, 3. Property Value

Object.defineProperty(o, 'a', {

value: 37, writable: true, enumerable: true, configurable: true

});

**4. Object.defineProperties()** Object.defineProperties(object1, {

property1: {

value: 42, writable: true

},

property2: {}

});

**5. Object.entries(obj)** const obj = { foo: 'bar', baz: 42 };

console.log(Object.entries(obj)); // [ ['foo', 'bar'], ['baz', 42] ]

// Gives out array of key,value pair || can be used with forEach

**for (const [key, value] of Object.entries(obj)) {**

**console.log(`${key} ${value}`); // "a 5", "b 7", "c 9"**

**}**

**6. Object.freeze()** A frozen object

**Object.isFrozen()**

**7. Object.fromEntries()** transforms a list of key-value pairs into an object.

const entries = new Map([

['foo', 'bar'],

['baz', 42]

]);

const obj = Object.fromEntries(entries);

Object { foo: "bar", baz: 42 }

**8. Object.getOwnPropertyDescriptor** o = { get foo() { return 17; } };

d = Object.getOwnPropertyDescriptor(o, 'foo');

// d is { configurable: true, enumerable: true,

// get: /\*the getter function\*/, // set: undefined

// }

**9. Object.getOwnPropertyNames()**  returns an array of all properties (including non-enumerable properties except

for those which use Symbol) found directly in a given object.

const object1 = {a: 1, b: 2, c: 3 };

console.log(Object.getOwnPropertyNames(object1));

// expected output: Array ["a", "b", "c"]

**10. Object.isExtensible()**

**Object.preventExtensions(object1);** // Checks if more value can be added or not, and can be prevent from extnsions

**11. Object.keys()** // returns an array of a given object's own enumerable property names, iterated

in the same order that a normal loop would.

**12. Object.values()** // Returns on values Array

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Add new** | **Remove existing** | **Change existing** |
| **Seal** | ❌ | ❌ | ✔ |
| **freeze** | ❌ | ❌ | ❌ |
| **preventExtensions** | ❌ | ✔ | ✔ |

**RegEx**

1**. /aditya/** Matches exact keyword

2. **/[abc]ditya/** Matches ans one word among abc has 1st char and must have all outside bracket

3. **/[^bcd]ditya/** Excludes everything in ^ char set

4**. /[a-z]ditya/** Matches ranges in char Set

5. **/[a-zA-Z]aditya/** Match both lower and upper case

6**. /[0-9]+/** + denotes one or unlimited times repeat

7**. /[0-9]{11}/** {11} denotes length of char set to be applied

8**. /[0-9]{5,8}/** {5, 8} denotes length of char set to be applied in range

9**. /[0-9]{5,}/** {5,} At least 5 char long

**i** = Case Insensative || By default its case sensative

**g** = Matches all occurance || By default matches fisrt occurance

**\d** = Eq of [0-9] || Matches any digits

**\w** = Eq of [a-zA-Z0-9\_] || Matches all Chars and \_ (underscores)

**\s** = Eq of Spaces , whiteSpaces || [\r\n\t\f\v]

**\t** = Eq of tabs || Matches intends

**'+'** = one or more

**'\'** = escape char

**'[]'** = Character Set

**'[^]'** = Negative symbol in char set

**'?'** = Zero or one qunatifier (makes preceding char optional) || OPTIONAL

**'.'** = Any Char whatsoever (except new line char)

**'\*'** = 0 or more quantifier (similar like +)

**'^'** = Outside char set denotes START of string

**'$'** = Denotes end of string

**'|'** = OR ex. p|tyre = will match p or tyre

**'()'** = group ex. (p|t)yre = will match pyre or tyre

// test it like this var RegExPattern = /\d{09}/;

// **RegExPattern.test("StringTotest")**

**Git**

1. **git init** = Used to convert folder to git repo

2. **git add <file name> || git add .** = Used to stage a file or all for commit

3. **git commit -m "Msg"** = Commit with message

4**. git status** = Gives status of current branch and staged files if any

5**. git config** = git config --global user.email "adityakale3@gmail.com"

git config --global user.name "Aditya Kale"

6. **git branch**

**git branch <branch\_name>** = Create a new branch

**git branch -a** = List all remote or local branches

**git branch -d <branch\_name>** = Delete a branch

7. **git checkout** = Checkout an existing branch

**git checkout -b <new\_branch>** Checkout and create a new branch with that name

8. **git merge**

**git merge <branch\_name>** = Merge changes into current branch

9. **git remote** = git remote add origin

git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git

# Adding a remote repository with the name of beanstalk

10. **git clone**

11. **git pull** = git pull git@github.com:adityakale3/Javascript.git staging

git pull origin staging

12. **git push** = git push <remote\_URL/remote\_name> <branch>

git push —all

13. **git log** = git log --pretty=oneline

git log --oneline

**JS DOM**

1**. document.getElementById('IdName');**

2. **document.getElementsByClassName('cName');** = Returns HTML Collection var elements[0]

3. **document.getElementByTagName('li');** = Returns HTML Collection

4. **document.querySelector('#wrapper')**  = document.querySelector('#books li:nth-child(2) .name')

5**. document.querySelectorAll('#wrapper')** = Returns Node List || Returns all occurance || Array methods work

*Convert HTML Collection into Array || let array =* ***Array.from(HTMLCollection);***

6. **element.parentElement** = gives parentElement

**element.children** = gives Children Element

**ParentElement.removeChild('ChildElement')** = Removes Element from DOM

7. **element.addEventListener('click', (e) => {})**

8. **document.forms['add-emp']**; = add-emp.addEventListener('submit', (e) => {

add-emp.querySelector('input[type="text"]').value; })

9. Create new Elements = const li = **document.createElement('li');**

const bookName = document.createElement('span');

const deleteBtn = document.createElement('span');

// add text content

bookName**.textContent** = value;

deleteBtn.textContent = 'delete';

// append to DOM

**li.appendChild(bookName);**

li.appendChild(deleteBtn);

list.appendChild(li);

//list.insertBefore(li, list.querySelector('li:first-child'));

10. **Element.classList** = List of class in array

**Element.classList.add('delete');**

**Element.classList.add('remove');**

**Element.classList.contains(class);**

11. **Element.getAttribute('href');**

**Element.hasAttribute('href');**

**Element.setAttribute('href', 'https://google.com');**

**Sockets**

**1.** **socket.emit**('hello', 'can you hear me?', 1, 2, 'abc'); = sending to the client

2. **socket.broadcast.emit**('broadcast', 'hello friends!'); = sending to all clients except sender

3. **socket.to**('game').emit('nice game', "let's play a game"); = sending to all clients in 'game' room except sender

4. **socket.to**('g1').to('g2').emit('nice game', "let's play"); = sending to all clients in 'g1' and/or in 'g2' room, except sender

5. **io.in**('game').emit('big', 'the game will start soon'); = sending to all clients in 'game' room, including sender

6. **io.of**('gm').emit('bigger', 'the tournament will start soon'); = sending to all clients in namespace 'gm', including sender

7. **io.of**('nSpace').to('room').emit('event', 'message'); = sending to a specific room in a specific nSpace, including sender

8. **io.to(socketId).emit**('hey', 'I just met you'); = sending to individual socketid (private message)

// WARNING: `socket.to(socket.id).emit()` will NOT work, as it will send to everyone in the room

// named `socket.id` but the sender. Please use the classic `socket.emit()` instead.

9. socket.emit('question', 'do you think so?'**, function (answer) {});** = sending with acknowledgement

10.socket.compress(false).emit('uncompressed', "that's rough"); = sending without compression

// sending a message that might be dropped if the client is not ready to receive messages

**socket.volatile.emit('maybe', 'do you really need it?');**

// specifying whether the data to send has binary data

**socket.binary(false).emit('what', 'I have no binaries!');**

// sending to all clients on this node (when using multiple nodes)

**io.local.emit**('hi', 'my lovely babies');

// sending to all connected clients

**io.emit**('an event sent to all connected clients');

**Mongo DB**

**Find :** db.documentName.find({query}, {fields})

**Count :** db.posts.find().count({key:value})

db.posts.find({ category: 'news' }).count()

**Distinct** : db.posts.find().distinct(key)

**Limit** : db.posts.find().limit(2).pretty()

**Specific Field** : db.posts.find({empcode:”20005211”}, {empcode:1, remarks:1, \_id:0})

***db.posts.findOne({ category: 'News' })***

|  |  |  |
| --- | --- | --- |
| $gt / $gte | greater than / greater than equals | db.ships.find({class:{$gt:’P'} |
| $lt / $lte | lesser than / lesser than equals | db.ships.find({class:{$lte:’P'} |
| $exists | does an attribute exist or not | db.ships.find({type:{$exists:true}}) |
| $regex | Perl-style pattern matching | db.ships.find({name:{$regex:’^USS\\sE’}}) |
| $type | search by type of an element | db.ships.find({name : {$type:2}}) |

|  |  |
| --- | --- |
| String | 2 |
| Array | 4 |
| Binary Data | 5 |
| Date | 9 |

**Insert :** db.dta.insert({key:value})

: db.dta.insertOne({key:value}) || db.dta.insertMany([ {key:value} , {key:value} ]);

**Delete** : db.dta.remove({ empcode: 20005211 });

**Update :** db.dta.update({ empcode: 20005211} , { $set : { newKey : newValue, newKey2 : newValue2} })

: db.dta.update( {serch:query}, {alldataKey : alldataValue} );

: db.dta.update( { id:val } , { $rename : { remarks : ‘comments’ } } )

**Aggregate :**

|  |  |
| --- | --- |
| $sum | Sums up the defined value |
| $avg | Calculates the average |
| $min | Gets the minimum |
| $max | Gets the maximum |
| $push | Inserts value to an array in the resulting document. |
| $addToSet | Same as push, but only unique values in array SETs |
| $first |  |
| $last |  |

|  |  |
| --- | --- |
| **$project** | Used to select some specific fields from a collection. |
| **$match** | filtering operation |
| **$group** |  |
| **$sort** |  |
| **$skip** | skip forward in the list of documents for a given amount |
| **$limit** |  |
| **$unwind** |  |

**Db.dta.aggregate([**

**{ $match : { empcode : “20004329” } },**

**{ $group : { \_id : “$tarikh” , timeduration : { $sum : “$timeduration” } } }**

**])**