Practical

-tsbi

UniqueNumbes

```
arr=[1,....]
let arr1=[]
for(i=0;i<arr.length;i++)
  let isUnique = true
  for(j=0;j<arr.length;j++){
    if(i!=j && arr[i]==arr[j]){
      isUnique = false
    }}
    if(isUnique){
      arr1.push(arr[i]
    }
  }
c.log(arr1)
```

PrimeNumbes

```
let arr=[1,3....]
function isPrime(num){
   if(num<=1){
   } return false

   for(i=2; i*i<=num; i++){
     if(num%i==0){
       return false
   }}
   return true
}

for(let i=0; i<arr.length; i++){
   if(isPrime(arr[i])){
     c.log(arr[i])
   }
}
```

Even remove

```
arr=[....]
for(i=0;i<arr.length;i++){
   if(arr[i]%2==0){
     arr.splice(i,1)
     i--
   }
}
c.log(arr)
```

Slice

Slice(2,-1) (start,end) +ve 2, value remove 1st -ve last value remove

Splice

Splice(1,1,33) (start index, delete count, item,item (new))

Reverse

```
let str='tamil'
let rev=''
for(i=str.length-1; i>=0; i--){
    rev+=str[i]
}
```

Max/min

```
let largest=-Infinity
let smallest=Infinity
for(i=0; i<arr.length; i++){
   if(arr[i]>largest){
     largest=arr[i]
   }else if(arr[i]<smallest){
     smallest=arr[i]
   }
}
c.log(largest, smallest)
```

2nd Largest

```
let Largest=-Infinity
let SecondLargest=-Infinity
for(i=0; i<arr.length; i++){
   if(arr[i]>largest){
     largest=secondLargest
     SecondLargest=arr[i]
   }else if(arr[i]>secondLargest && arr[i]
     {
       secondLargest=arr[i]
     }
c.log(secondLargest)
```

Deep copy

```
arr1=JSON.parse(JSON.stringify(arr))
```

ARRAYMETHOD

- arr.Push(4) - add last
- .Pop() - remove last
- .Shift() - remove 1st
- .unshift(0) - add 1st
- .Concat(['3']) - join
- .join(['2']) - in given join
- .lastIndexOf('1') - return
- .IndexOf('1') - return

MATH

- Ceil(x) - round next Large
- floor(x) - round Smallest
- round(x) - nearest
- trunc(x) - remove fraction
- random() - random num
- Sqrt(x) - Square root
- abs(x) - absolute (remove -ve)
- cbrt(x) - Cube root
- E - Euler's num
- pow(x,2) - power of 2

call, apply, bind

```
function greet(country){
  c.log(`name: ${this.name}, age: ${this.age}, ${country}`)
}

const person = { name: 'joe', age: 23 }
```

call

```
greet.call(person, 'india')
```

apply

```
greet.call(person, ['india'])
```

bind

```
const bindMethod = greet.bind(person, 'India')
c.log(bindMethod)
```

Empty

```
Object.values(obj).length === 0
```

manipulation

```
obj.name          //read
obj.age = '22'    //update
obj.county = 'ind'  //create
delete obj.age    //delete
```

Methods

```
Object.keys(obj) - key in array
Object.value(obj) - value in array
Object.entries(obj) - key and value in array
Object.assign(obj, {key: val})
Object.freeze(obj) - immutable
Object.seal(obj) - x delete x add ✓ update
```

Optional chaining

```
const obj = obj?.address?.country?.name
```

this not in it

Obj double

```
obj = {a:1, b:2, c:3}
for(let x in obj){
  obj[x] *= 2
}
```

Swap key val

```
swap = {}
for(let x in obj){
  swap[obj[x]] = x
}
```

OR

```
Object.map(([key, val]) => {
  [key, val] = [val, key]
})
```

Largest age object

```
largestObject = ''
if(largestObject == '' || arr[i].age > largestObject.age)
{
  largestObject = arr[i]
}
c.log(largestObject)
```

Sum of number objects

```
obj = {a:{b:1, c:2}, d:3}
let sum = 0
for(let key in obj){
  if(typeof obj[key] == 'number'){
    sum += obj[key]
  } else {
    for(let nested in obj[key]){
      if(typeof obj[key][nestedKey] == 'number'){
        sum += obj[key][nestedKey]
```

23) color sum)

Promise

```
const myPromise = new Promise((res, rej) => {
  const success = true
  if(success){
    res('operation success')
  }else{
    rej('operation failed')
  }
})
myPromise
.then((message) => {
  c.log(message)
})
.catch((error) => {
  c.log(error)
})
```

Async-await

```
function firstPromise(){
  return new Promise((res, rej) => {
    res('sucess')
  })
}

async function display(){
  try{
    const user = await firstPromise()
    c.log(user)
  } catch(error){
    c.log(error)
  }
}
display()
```

Promise-race

```
const promise1 = new Promise((res, rej) => {
  resolve('1px success')
})
like promise

const promise = promise.race([promise1, promise2])
.then((response) => {
  c.log(response)
})
.catch((error) => {
  c.log(error)
})
```

Promise arithmetic

```
function add(a, b){
  return new Promise((res, rej) => {
    if(a||b == 0){
      rej('answer failed')
    }else{
      resolve(a+b)
    }
  })
}
add(0, 0)
```

```
.then((res) => {
  c.log(res)
})
.catch((error) => {
  c.log('error')
})
```

Currying

```
const add = a => b => c => a+b+c
c.log(add(1)(2)(3))
```

```
function add(a){
  return function(b){
    return function(c){ return a+b+c
```

call back

```
function displayMsg(){
    c.log ('How are you')
}
function greet (name, cb){
    c.log ('hello' + name)
    cb()
}
greet ('Boss', displayMsg)

function display (sum){
    c.log (sum)
}
function add (a,b, callback){
    let sum = a+b
    callback(sum)
}
add (1, 4 display)
```

while loop

```
let num = [1, 2, 3, 4, 5]
let sum = 0
let i = 0
while (i<num.length){
    sum += num[i];
    i++
}
c.log (sum
```

Do while

```
num = [1, 2, 3, 4, 5]
sum = 0
i = 0
do{
  sum += nums[i];
  i++;
}
while (i< num.length)
c.log (sum)
```

generator function

```
function* numGen(){
    yield 1,
    yield 2,
    yield 3,
}
const gen = numGen()
c.log (gen.next().value)
c.log (gen.next().value)
c.log (gen.next().value)
c.log (gen.next().value)
```

Closure

```
function display () {
  var num1 = 10
  var num 2 = 20
  function show(){
    let sum = sum1 + sum2
    c.log (sum)
  }
  return sum
}
const cal = display ()
cal()
```

Hoisting

```
c.log (a)        undefined
let a = 10
c.log (a)        10
```

ternary

```
num = 4
result = num % 2 = 0 ? 'Even' : 'odd'
clog (result)
```

factory function

```
function factory (name, place){
    return {
        name: name,
        Place: place,
        greet(){
            return `name : ${this.name}, fro
        }
    }
}
const user = factory('jane', 'london')
c.log (user.greet())
```

Set Interval

```
const interval = SetInterval(()=>{
    c.log ('hey');
},2000)
setTimeout (()=>{
    clearInterval(interval)
},7000)
```

Child Process

Parent

```
const {fork} = require('child_Process')
const child = fork('child.js')

child.on('message', (message)=>{
    c.log ('message from child : ${message}')
})
child.send ('parent say, hii bro')
```

child

```
Process.on('message', (msg)=>{
    c.log ('message from parent : ${msg}')
})
Process.send ('child say, yeh broo')
```

html server

```
require http
const server = http.createServer((req, res) => {
    res.write('hi')
    res.end()
}).listen(2000)
```

query

```
const name = req.query.name
res.send(`${name}`)
const num1 = parseInt(req.query.num1)
const num2 = parseInt(req.query.num2)
const sum = num1 + num2
if (isNaN(num1) || isNaN(num2)) {
    res.send('valid numbers')
} else {
    res.send(`sum ${sum}`)
}
res.send(req.query)
```

Params

```
app.get('/:name', (req, res) => {
    const name = req.params.name
    res.send(`name: ${name}`)
app.get('/:check', (req, res) => {
    const current = req.params.check
    const current_id = profile[current]
    res.send(current_id)
```

Event Emitter

```
const events = require('events')
const emitter = new events.EventEmitter()
emitter.on('click', () => {
    console.log('click successfully')
})
emitter.emit('click')
```

Date

```
const path = require('path')
const fs = require('fs')
const today = new Date().toString()
const filepath = path.join(__dirname, 'sample.txt')
fs.writeFile(filepath, today, (err) => {
    if (err) {
        console.log(err)
    } return
})
```

URL parse

```
const url = require('url')
const result = url.parse('http://...')
console.log(result)
```

middleware

```
req
const express
app = express()
const middleware = function (req, res, next) {
    console.log('middle ware')
    next()
}
app.use(middleware)
app.get('/', (req, res) => {
    res.send('hi')
}).listen(...)
```

Error handling

```
app.get('/', (req, res, next) => {
    next(new Error('this is error middleware'))
})
app.use((err, req, res, next) => {
    console.log(err.stack)
    res.status(404).send('page not found').listen(...)
```

Router handling

```
const router = express.Router()
router.use((req, res, next) => {
    console.log('router triggered')
})
router.get('/', (req, res) => {
    res.send('router middleware')
app.use('/router', router)
app.listen(200)
```

Morgan

```
const morgan = require('morgan')
app.use(morgan('dev'))
app.use((req, res) => {
    res.send('hello')
app.listen(...)
```

fs

```
const fs = require('fs')
```

write

```
const content = 'Hai'
fs.writeFile('my.txt', content, (err) => {
    if (err) {
        log('failed')
    } else {
        log('success')
    }
})
```

read

```
try {
const data = fs.readFileSync('new.txt', 'utf-8')
    console.log('file content :', data)
} catch (error) {
    log(error)
}
```

copy

```
fs.copyFile('new.txt', 'copy.txt', (err) => {})
```

unlink

```
fs.unlink('new.txt', () => { if (err) { ...
```

truncate

```
fs.truncate('myfile.txt', () => { if ...
```

Find letter

{name: {$regex: 'at', $options: 'i'}}

{name: {$regex: /at/i}}

$sum

([{$group: {_id: null, sum: {$sum: '$price'}}}])

$avg

([{$group: {_id: null, average: {$avg: '$price'}}}])

$count

- db.user.count({age: 25})
- db.user.countDocuments({price: {$exists: true}})
- ([{$group: {_id: null, count: {$sum: 1}}}])

$inc

{$inc: {age: 5}}

$dec

{$inc: {age: -5}}

Upsert

updateOne

({_id: 1}, {$set: {name: 'Alice', age: 30}}, {upsert: true})

$unwind

{$unwind: '$age'}

$lookup

{$lookup: {
  from: 'name of other collection',
  localField: 'field from current position',
  foreignField: 'field from other collection',
  as: 'ouput array name'}}

$capped collection

db.createCollection('name', {capped: true, size: 1024*1024, max: 1000})

$rename

- ([{}, {$rename: {'price': 'cost'}}])
- db.collectionname.renameCollection('newname')

$out

{$out: 'newcollectionname'}

$addToSet

if not add or if present not add

([{name: 'Alice'}, {$addToSet: {hobby: ...}}])

$bulkWrite

db.product.bulkWrite([
  {insertOne: {
    'document': {
      'name': 'Laptop',
      'price': 1200,
      'category': 'electronics'}}},
  {updateOne: {
    'filter': {name: 'laptop'},
    'update': {$set: {'price': ...}}},
  {updateMany: {...}},
  {deleteOne: {
    filter: {'name': 'oldLaptop'}}}
])

multiple write operation in single batch (insert, update, delete)

$expr

db.product.find({
  $expr: {
    $gt: ['$price', 'avgPrice']}}})

$elemMatch

regex name

([{gadget: {
  $elemMatch: {
    Product: 'Laptop',
    Quantity: {$gt: 2}}}}])

matched value

ARRAY

create

([{_id: 6, $set: {project: []}}])

add

([{_id: 2}, {$push: {project: {$each: ['p2', 'p3']}}}])

remove

([{}, {$pull: {project: {'p3'}}}])

key name

$facet

aggregate

{$facet: {
  totalProducts: [
    {$count: 'total'}],
  averagePrice: [
    {$group: {_id: null, avg: ...}}]}}

output

averagePrice: [{_id: null, avg: 250}]

Index

- **Single field index**

```
db.collection.createIndex({name:1})
              .find({name:joe})
```

- **Compound index**

```
db.collection.createIndex({name:1, age:-1})
```

- **Multikey index** name:'Joe', age:50

```
db.collection.createIndex({tags:1})
```

- **Text index** tags:mongodb

```
db.collection.createIndex({description:'text'})
{$text:{$search:'some text'}}
```

- **Hashed index**

```
db.collection.createIndex({user_id:'hashed'})
{_id:'hashed'}
```

- **Geospatial 2D index**

```
db.collection.createIndex({location:'2d'}),
location={$near:[50,50]}
```

- **Geospatial 2Dsphere index**

```
db.collection.createIndex({location:'2dsphere'})
location:{$geoWithin:{$centerSphere:
[[50,40],10/3954]}}
```

- **TTL index**

```
db.collection.createIndex({date:1},{expireAfterSeconds:600}),
```

Pading

aggregate

```
{$match:{address.city":"banglore",
    yoj:2015
    }
},
{$group:{_id:null, avgsalary:{$avg:'$salary'}}})
```

Delete

```
db.collection.deleteOne({name:'Joe'})
             deleteMany
db.collection.drop()
```

$exists

```
{'field':{'$exists':<boolean>}}
```

$limit

```
{$limit:<number>}
```

$sort

- `db.collection.find().sort()`
- `db.collection.aggregate({$sort:{salary:-1}},`
  `{$limit:1}`

Promise fs

```
const fs=require('fs').promise
fs.readFile('sample.txt', 'utf8')
  .then((data)=>{
    c.log(data)
  }).catch((err)=>{
    c.log(err)
  })
```

Promise greatly

```
function gt(a,b){
  return new Promise((res,rej)=>{
    if(a>b){
      res(`Promise resolved, ${a} greater than ${b}`)
    }else{
      rej(`Promise rejected, ${b} not greater than ${a}`)
    }
  })
}
```

then catch

```
gt(9,6).then((message)=>{
    c.log(message)
  }).catch((err)=>{
    c.log(err)
  })
```

async await

```
async function handlePro(a,b){
  try{
    const result=await gt(a,b)
    c.log(result)
  }catch(err){
    c.log(err)
  }
}
handlePro(9,12)
```

Map

```
MyMap=new Map()
create
myMap.set('name', 'Alice')
read
myM.get('name')
check
MyMap.has('name')
delete
MyMap.delete('name')
clear
myMap.clear()
```

rest

```
function sum(...numbers){
  return numbers.reduce((total,num)=>total+num,0)
}
c.log(sum(1,2,3,4))
```