Table of Contents

[Introduction 2](#_Toc128566189)

[Set:- 2](#_Toc128566190)

[Methods on set - 3](#_Toc128566191)

[Bitwise operator 4](#_Toc128566192)

[&(bitwise and) 4](#_Toc128566193)

[|(bitwise or) 4](#_Toc128566194)

[bitwise XOR (^) 5](#_Toc128566195)

[bitwise NOT (~)-- 5](#_Toc128566196)

[Right shift(>>) 6](#_Toc128566197)

[left shift(<<) 6](#_Toc128566198)

[Map 6](#_Toc128566199)

[Hoisting 7](#_Toc128566200)

[**OOPS(Object Oriented Programming Language)** 7](#_Toc128566201)

[Class 7](#_Toc128566202)

[Object- 8](#_Toc128566203)

[Program 8](#_Toc128566204)

[Program1 8](#_Toc128566205)

[program2 8](#_Toc128566206)

[program3 9](#_Toc128566207)

[Program 4 9](#_Toc128566208)

[Constructor 10](#_Toc128566209)

[Stacks 11](#_Toc128566210)

[program- 11](#_Toc128566211)

[Queue 12](#_Toc128566212)

[Note 12](#_Toc128566213)

[Linked List 13](#_Toc128566214)

[Single Linked List 13](#_Toc128566215)

[Creating Node(Box) 14](#_Toc128566216)

# Introduction

refer to f[older 13-02-2023](../../JS/programming/13-02-2023)

data structure are just containers to keep our data,

will also helpus to organise our data.

Array- as we already learnt abt this.

## Set:-

Collection of unique elements

ex-(2,3,4) ---> Set

(22,22,21)---> not a set as in this there was a duplicate value

(2,'20','g')---> set

create a set

const X = new Set()

x.add(2); ---> inserting data into set

x.add(3);

var arr=[2,3,5,4,5,6,1,5,4,3]

var x=new set(arr)---> allt the duplicate will be removed from var X;

x=[2,3,5,4,6,1]

x.add(15)

x.add(15) // this line will not produce error but also it should not be ADDED in the set

Methods on set - >

#### forEach method->

var arr=[2,3,5,4,1,2]

var x=new Set(arr)

x.forEach(doJob)

funtion doJob(value,index,x){

console.log(value,index,x)

}

add()---> it will add an element to the set

var arr=[2,3,5,4,5,6,1,5,4,3]

var x=new set(arr)

x=[2,3,5,4,6,1]

x.add(15)

delete()---> used to delete an element from the Set

it will return true if no is found and deleted.

else it will return false.

x.delete(2);

has()--> used to check whether the number is present in set or not.

x.has(3);

clear()---> remove all element from the set.

x.clear();

size- return the size of set.

# Bitwise operator

& - bitwise AND operator

| - bitwise OR operator

&(bitwise and)

bit1 bit2 &(bitwise and)

1 1 1

1 0 0

0 1 0

0 0 0

|(bitwise or)

bit1 bit2 |(bitwise or)

1 1 1

1 0 1

0 1 1

0 0 0

prob

2&3

2-->10

3-->11

1 0

& 1 1

1 0

2 & 8- 0000

13&6 - 0100 --4

18&15-- 00010--2

3|5 -- 111- 7

3|7 -- 111== 7

bitwise XOR (^) -- it will produce 1 if only one bit is 1 otherwise it will poduce 0

0 0 0

1 0 1

0 1 1

1 1 0

bitwise NOT (~)-- > revert the bit

~ 0==1

~ 1==0

~8=7

8^6=14

~(8^5)= 2

~8^~6 = 6

## Right shift(>>)

ex- 18>>2

it will remove 2 iteration from right and insert 0 to the left syd of bianry no

## 

## left shift(<<)

ex--

26<<3

ti wil insert 0 to the right side.

# Map

refer to [folder 14-02-2023](../../JS/programming/14-02-2023)

Map-->

i. store key value pair

x:20,

y:3

`ii. map know the order of insertion of keys.

var x= new Map()

x.set("name","sudhanshu")

x.delete(a) it will delete the key 'a'

x.clear()--> it will remove all the key value from map

size-- it will gice the size of map

has(key) –it gives the Boolean value in true or false

get(key) – it give the calue of the key

# Hoisting

In hoisting we can acess variable before initialization or declaring it

Ex

Console.log(a)

Var a=10 this give undefined

Console.log(b)

Let b=10 this give cannot access b before initialization

Console.log(d)

Const d=10 this give cannot access b before initialization

# **OOPS(Object Oriented Programming Language)**

Refer to [folder OOPS](../../JS/programming/oops(17))

## Class

* it is nothing but the blueprint of an entity
* class will contain all field of that particulaar entity

class Person{

var name;

var age;

var gender;

var email;

}

## Object-

* object is nothing but it is an instance of class
* there are different ways to create object of a class

## Program

### Program1

class Person{

    constructor(name,email,age,gender){

        this.name=name;

        this.email=email;

        this.age=age;

        this.gender=gender;

    }

print(){

    console.log("name: " +this.name,"email: " +this.email,"age : " +this.age,"gemder : "+this.gender)

}

}

var man=new Person("Ankit","vank@gmail.com",24,"male")   //create object of class

man.print();       //calling function of the class

### program2

class vehicel{

    constructor(numberoftyre,numberofMirror,isEngineavailable,isBreakavailable){

        this.numberoftyre=numberoftyre;

        this.numberofMirror=numberofMirror;

        this.isEngineavailable=isEngineavailable;

        this.isBreakavailable=isBreakavailable;

    }

    PrintDetails(){

        console.log("number of tyres is " +this.numberoftyre,this.numberofMirror,this.isEngineavailable,this.isBreakavailable)

    }

}

var car=new vehicel(41,4,true,true)

car.PrintDetails();

### program3

const person= {

    PrintValue:function PrintValue(name,gender,age,mail){

        console.log("name: "+name,"age : "+age,"gender : "+gender,"mail : "+mail)

    }

}

const person1=Object.create(person);    // create object of the variable person

person1.PrintValue("ANkit","Male",24,"vani@gmail.com");

### Program 4

class MathematicalOperations {

    constructor(num1,num2){

        this.num1=num1;

        this.num2=num2

    }

    add(num1,num2){

        console.log("addition is : "+(this.num1+this.num2));

    }

    subtract(num1,num2){

        console.log("subtraction is : "+(this.num1-this.num2))

    }

    multiply(num1,num2){

        console.log("multiplication is : "+(this.num1\*this.num2))

    }

    divide(num1,num2){

        console.log("division  is : "+(this.num1/this.num2))

    }

}

var operaition= new MathematicalOperations(10,5)

operaition.add();                   // in this we did the all thee operations with the same value

operaition.subtract();

operaition.multiply();

operaition.divide();

#### 2nd method

class MathematicalOperations {

    add(num1,num2){

        console.log("additiomn is : "+(num1+num2))

    }

    subtract(num1,num2){

        console.log("subtraction is : "+(num1-num2))

    }

    multiply(num1,num2){

        console.log("multiplication is : "+(num1\*num2))

    }

    divide(num1,num2){

        console.log("division  is : "+(num1/num2))

    }

    MOD1(num1,num2){

        console.log("MOD is : " +(num1%num2))

    }

}

var operaition=new MathematicalOperations()

operaition.add(5,2);

operaition.subtract(15,2);

operaition.multiply(15,9);

operaition.divide(525,5)

operaition.MOD1(85,9)

## Constructor

it will executed evry time when we create an object of the class

class test {

constructor(){

this.a=10

this.b=20

this.c=30

}

dojob(){

console.log("object has been create")

}

}

var t= new test()

console.log(t.a,t.b,t.c)

t.dojob();

# Stacks

it is a data structure in which it work with **filo (first in last out) or lifo (last in first out)** system

push - insert value to the stack

pop - extract a value from stack

peek - tell me the value that is on top.t will not remove any element

size - tell me the size of stack

## program-

var stack=[]

function push(val){

stack.push(val)

}

function print(){

for(var i=0;i<stack.length;i++){

process.stdout.write(" "+stack[i])

}

}

function size(){

console.log(" "+stack.length)

}

function peek(){

if(stack.length==0){

console.log("stakck empty")

}

else{

console.log(stack[stack.length-1])

}

}

push(20);

push(40);

push(60);

push(80);

push(45);

print();

size();

console.log(" "+stack.pop());

peek();

# Queue

queue is a data structure that works on First in and First out perinciple

same method as stack

push- used to insert element in the queue

pop- pop used to extract element by shift and unshift method

peek- used only to show the element of the index this does not change in queue

size give the length of queue

implementation is same as stack

Note-

* function inside the class called as method.
* By var u can access variable before initialization
* In let and const we can not access variable before initializing
* That is called hoisting- ability to use variable before initialisation

# Linked List

Rfer to folder [..\..\JS\programming\pracrise(27)](../../JS/programming/pracrise(27))

linear data structure occupy memory in sequention manner.

Ex-array,set,queue,stack

non linear data structure occupy memory in sequentionial manner

whereas in non sequentionial manner also

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1000 | 1004 | 1008 | 1012 | 1014 | 1018 |

Inserting linked list

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1000 {  Value of this block} | nextAddress of 1008 | 1008 {  Value of this block} | nextAddress of 1014 | 1014 {  Value of this block} | nextAddress of 1018 |

## Single Linked List

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1000  Value of this block | nextAddress of next block 1004 | 1004  value | Nextaddress  1008 | 1008  Value | Address to null |

It should start with **start** variable and end with **NULL**

### Creating Node(Box)

Creating NOde

class Node{

constructor(value,next){

this.value=value

this.next=nextaddress

}

}

var obj=new Node(10,2000)

console.log(obj.value)

console.log(obj.next)