**Exercise 3.1**

function add(...args) {

return args.reduce((sum, currentValue) => sum + currentValue, 0);

}

const memoizeAdd = memoize(add);

memoizeAdd(100, 100);

memoizeAdd(100);

memoizeAdd(100, 200);

memoizeAdd(100, 100);

function memoize(add) {

const cache = {};

return function (...args) {

const key = JSON.stringify(args);

if (key in cache) {

return cache[key];

}

const result = add(args);

cache[key] = result;

return result;

};

}

**Exercise 3.2**

Call, Apply & Bind are types of function borrowing. (Function borrowing – when we borrow a function, which may or may not be part of an object, and pass data from another object)

e.g

let name1 = {

x:"sarthak",

y:"patnaik"

};

let printMsg = function (homeTown, State) {

console.log(this.x+" "+this.y+" is from "+homeTown+", "+State);

};

//Call

/\*The reference object along with function specific parameters are passed, separated by commas\*/

printMsg.call(name1,"BBSR","ODISHA");

//Apply

//The reference object is passed. Along with it, other parameters are passed as a list.

//This can be used when dynamic when we have dynamic arguments

printMsg.apply(name1,["LUCKNOW","UP"]);

//Bind

//bind creates and returns a copy of the function concerned and binds the object and parameters to it. The returned function can be invoked later as a separate function.

let newPrintMSg = printMsg.bind(name1,"KOLKATA","WB");

newPrintMSg();

/\*Console

sarthak patnaik is from BBSR, ODISHA

sarthak patnaik is from LUCKNOW, UP

sarthak patnaik is from KOLKATA, WB\*/

**Exercise 3.3**

function createIncrement()

{

letcount=0;

functionincrement() {

count++;

}

letmessage=`Count is${count}`;

functionlog() {

console.log(message);

}return[increment,log];

}

const[increment,log] =createIncrement();

increment();

increment();

increment();

log();// What is logged?

Here, the answer is: 0 . As 0 was logged in the lexical scope for log function when closure was returned.

**Exercise 3.4 (createStack)**

function createStack(){

let items = [];

function push(newItem){

items.push(newItem);

}

function pop() {

return items.pop();

}

function print() {

return items;

}

return {push, pop, print};

}

const stack=createStack();

stack.push(1);

stack.push(2);

stack.push(3);

stack.push(4);

stack.pop();

stack.pop();

stack.push(6);

console.log(stack.print());

Ans:- [ 1, 2, 6 ] . Here, all 3 functions refer to the same lexical scope. The same list is being utilized and referred to by the functions.