**Exercise 2**

var Node = function (name) {

this.children = [];

this.name = name;

}

Node.prototype = {

add: function (child) {

this.children.push(child);

},

remove: function (child) {

var length = this.children.length;

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

if (this.children[i] === child) {

this.children.splice(i, 1);

return;

}

}

},

getChild: function (i) {

return this.children[i];

},

hasChildren: function () {

return this.children.length > 0;

}

}

function traverse(indent, node) { console.log(Array(indent++).join("--") +

node.name);

for (var i = 0, len = node.children.length; i < len; i++)

traverse(indent, node.getChild(i));

}

}

function run() {

var tree = new Node("root");

var left = new Node("left")

var right = new Node("right");

var leftleft = new Node("leftleft");

var leftright = new Node("leftright");

var rightleft = new Node("rightleft");

var rightright = new Node("rightright");

tree.add(left);

tree.add(right);

tree.remove(right);

tree.add(right);

left.add(leftleft);

left.add(leftright);

right.add(rightleft);

right.add(rightright);

traverse(1, tree);

}

Q-2

public interface TaxCalculator {

public abstract void execute();

}

public class Humanity implements TaxCalculator {

private int basic\_salary;

public Order(int basic\_salary) {

this.basic\_salary = basic\_salary;

}

@Override

public void execute() {

HRA=(10/100)\*basicsalary;

}

}

public class Logistic implements TaxCalculator {

private int basic\_salary;

public Order(int basic\_salary) {

this.basic\_salary = basic\_salary;

}

@Override

public void execute() {

HRA=(10/100)\*basicsalary;

}

}

public class Department {

public static void main(String[] args) {

basic\_salary basic\_salary = new basic\_salary();

Humanity humanity = new Humanity(basic\_salary);

Logistic logistic = new Logistic(basic\_salary);

Humanity.execute();

humanity = new humanity(basic\_salary);

logistic = new Logistic(basic\_salary);

Logistic.execute();

}

}

**Q3.**

**const** arr = [4, 6, 7, 8, 9, 10, 10];

**const** findVariance = (arr = []) => {

**if**(!arr.length){ **return** 0;

};

**const** sum = arr.reduce((acc, val) => acc + val);

**const** { length: num } = arr; **const** median = sum / num; let variance = 0; arr.forEach(num => {

variance += ((num - median) \* (num - median));

});

variance /= num;

**return** variance;

};

console.log(findVariance(arr))

Q4.

**class** productId

{

constructor( productId, ProductName,Productprice)

{

**this**.productId=productId; **this**.ProductName=ProductName; **this**.Productprice=Productprice;

}

}

let ob1=**new** productId(1111,aaaa,3345); let ob2=**new** productId(22,bbb,3456);

***Q5***

import { useState } from 'react';

export default function Form() {

// States for registration

const [name, setName] = useState('');

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

// States for checking the errors

const [submitted, setSubmitted] = useState(false);

const [error, setError] = useState(false);

// Handling the name change

const handleName = (e) => {

setName(e.target.value);

setSubmitted(false);

};

// Handling the email change

const handleEmail = (e) => {

setEmail(e.target.value);

setSubmitted(false);

};

// Handling the password change

const handlePassword = (e) => {

setPassword(e.target.value);

setSubmitted(false);

};

// Handling the form submission

const handleSubmit = (e) => {

e.preventDefault();

if (name === '' || email === '' || password === '') {

setError(true);

} else {

setSubmitted(true);

setError(false);

}

};

// Showing success message

const successMessage = () => {

return (

<div

className="success"

style={{

display: submitted ? '' : 'none',

}}>

<h1>User {name} successfully registered!!</h1>

</div>

);

};

// Showing error message if error is true

const errorMessage = () => {

return (

<div

className="error"

style={{

display: error ? '' : 'none',

}}>

<h1>Please enter all the fields</h1>

</div>

);

};

return (

<div className="form">

<div>

<h1>User Registration</h1>

</div>

{/\* Calling to the methods \*/}

<div className="messages">

{errorMessage()}

{successMessage()}

</div>

<form>

{/\* Labels and inputs for form data \*/}

<label className="label">Name</label>

<input onChange={handleName} className="input"

value={name} type="text" />

<label className="label">Email</label>

<input onChange={handleEmail} className="input"

value={email} type="email" />

<label className="label">Password</label>

<input onChange={handlePassword} className="input"

value={password} type="password" />

<button onClick={handleSubmit} className="btn" type="submit">

Submit

</button>

</form>

</div>

);

}