# Node

```
const http = require('http');     ■ File is a CommonJS module; i
const PORT = 3000;
const HOST = '127.0.0.1';

const server = http.createServer(async (req,res) => {
    if(req.method === 'GET' && req.url === '/greet'){
        res.statusCode = 200;
        res.setHeader('Content-Type','text/plain');
        res.end('Hello World');
    } else if(req.method === 'POST' && req.url === '/add'){
        try{
            const body = await parseBody(req);
            const ans = body.numberA + body.numberB;
            const mul = body.numberA * body.numberB;
            const ret = { addition: ans, multiplication: mul};
            res.statusCode = 201;
            res.setHeader('Content-Type','application/json');
            res.end(JSON.stringify(ret));
        } catch (error){
            res.statusCode = 500;
            res.setHeader('Content-Type','application/json');
            res.end(JSON.stringify({message:"Error"}));
        }
    }
});

server.listen(PORT,HOST, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

# Express

```
const express = require('express');      ■ File is a CommonJS module; it may be co
const app = express();
const PORT = 3000;

app.use(express.json());

let items = [
    { id: 1, name: 'Item 1' },
    { id: 2, name: 'Item 2' }
];

app.post('/items', (req, res) => {
    const newItem = { id: items.length + 1, name: req.body.name };
    items.push(newItem);
    res.status(201).json(newItem);
});

app.get('/items', (req, res) => {      ■ 'req' is declared but its value is never
    res.json(items);
});

app.get('/items/:id', (req, res) => {
    const item = items.find(i => i.id === parseInt(req.params.id));
    if (!item) return res.status(404).json({ message: 'Item not found' });
    res.json(item);
});

app.put('/items/:id', (req, res) => {
    const item = items.find(i => i.id === parseInt(req.params.id));
    if (!item) return res.status(404).json({ message: 'Item not found' });
    item.name = req.body.name;
    res.json(item);
});

app.delete('/items/:id', (req, res) => {
    const index = items.findIndex(i => i.id === parseInt(req.params.id));
    if (index === -1) return res.status(404).json({ message: 'Item not found' });
    items.splice(index, 1);
    res.status(204).send();
});

app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

# Modules

```
class Math{
    add(a,b){
        return a+b;
    }

    subtract (a,b) {
        return a-b;
    }
}

module.exports = new Math();
```

```
module.exports = {
  add(x, y) {
    return x + y;
  },
  subtract(x, y) {
    return x - y;
  },
  multiply(x, y) {
    return x * y;
  },
  divide(x, y) {
    return x / y;
  },
};
```

```
const Math = function(){};
Math.prototype.add = (x, y) => x + y;
Math.prototype.subtract = (x, y) => x - y;
Math.prototype.multiply = (x, y) => x * y;
Math.prototype.divide = (x, y) => x / y;

const math = new Math();

module.exports = math;

//use
const math = require('./mathModule');
console.log(math.add(2, 3));
console.log(math.subtract(7, 4));
console.log(math.multiply(3, 3));
console.log(math.divide(10, 2));
```

# ES6

```
const numbers = [1,2,3,4];
numbers.forEach((num) => {
    console.log(num)
});

const doubled = numbers.map((num,index) => num*index);

const even = numbers.filter(num => num%2 === 0);

const found = numbers.find(num => num > 2);

const index = numbers.findIndex(num => num == 3);

//reduce
//array_name.reduce( callback(accumulator,curr_val), initialValue);
const sum = numbers.reduce((total,num) =>{
    return total + num;
},0);

const people = [
  { name: 'John', age: 25 },
  { name: 'Jane', age: 22 },
  { name: 'Mike', age: 25 },
  { name: 'Anna', age: 22 }
];

const groupedByAge = people.reduce((accumulator, currentValue) => {
  const age = currentValue.age;
  if (!accumulator[age]) {
    accumulator[age] = [];
  }
  accumulator[age].push(currentValue);
  return accumulator;
}, {});

//includes
const numbers = [1, 2, 3, 4];
console.log(numbers.includes(2)); // Output: true

//sort
const numbers = [4, 2, 3, 1];
numbers.sort((a, b) => a - b); // Ascending sort
```

```
//concat
const arr1 = [1, 2];
const arr2 = [3, 4];
const merged = arr1.concat(arr2);

const numbers = [1, 2, 3, 4, 5];
const sliced = numbers.slice(1, 3);
console.log(sliced); // Output: [2, 3]

const numbers = [1, 2, 3, 4];
numbers.splice(1, 2); // Removes 2 elements starting from index 1

numbers.push(5) //add 5 to the end
numbers.pop() //remove 5 from the end
```

# React

```
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<ContactList contacts={contacts} />);
```

When using list items you need a key value like the index to give the element a unique value.

```
class PersonList extends React.Component {
    render() {
        return (
            <ul>
                {this.props.people.map((person, index) => {
                    return (
                        <Person person={person} key={index}/>
                    );
                })}
            </ul>
        );
    }
}
```

Want to return a bunch of components without wrapping it inside a div:

```
class ListItems extends React.Component{
    constructor(props){
        super(props);
    }

    render(){
        return(
            <React.Fragment>
                <li>List item 1</li>
                <li>List item 2</li>
                <li>List item 3</li>
                <li>List item 4</li>
            </React.Fragment>
        );
    }
}
```

## Binding

```
export class Person extends React.Component{
    constructor(props){
        super(props);
        this.sayHey = this.sayHey.bind(this);
    }

   sayHey(e){
        e.preventDefault();
        console.log(`Hey, my name is ${this.props.person.name}`);
   }

    render(){
        const person = this.props.person;      'person' is declar
        return(
            <div>
                <button inClick={this.sayHey}>Click me</button>
            </div>
        );
    }
}
```

## State

State is initialized in the constructor.

```
class ToggleButton extends Component {
  constructor(props) {
    super(props);
    this.state = {
      isOn: false,
    };
    this.toggleButton = this.toggleButton.bind(this);
  }

  toggleButton(e) {
    e.preventDefault();
    this.setState((prevState) => ({
      isOn: !prevState.isOn,
    }));
  }

  render() {
    return (
      <div>
        <button onClick={this.toggleButton}>
          {this.state.isOn ? "On" : "Off"}
        </button>
      </div>
    );
  }
}
```

Ref

```
export class PersonForm extends React.Component{
    constructor(props){
        super(props);
        this.nameRef = React.createRef();
        this.ageRef = React.creatRef();
        this.colourRef = React.createRef();
    }

    handleSubmit = (e) => {
        e.preventDefault();
        const name = this.nameRef.current.value;
        const age = this.ageRef.current.value;
        const colour = this.colourRef.current.value;
        alert(`Name: ${name}, Age: ${age}, Favorite Colour: ${colour}`);
    };

    render(){
        return(
            <form onSubmit={this.handelSubmit}>
                <label for="name">Name</label>
                <input type="text" id="name" name="name" ref={this.nameRef}/>

                <label for="age">Age</label>
                <input type="number" id="age" name="age" ref={this.ageRef}/>

                <label for="colour">Colour</label>
                <select ref={this.colourRef}>
                    <option value="pink">Pink</option>
                    <option value="blue">Blue</option>
                </select>
                <button type="submit">Submit</button>
            </form>
        );
    }
}
```

## Person List Example

```
export class PersonList extends React.Component{
    constructor(props){
        super(props);
        this.state = {people:this.props.people};
        this.addPerson = this.addPerson.bind(this);
    }

    addPerson(name,surname){
        let people = this.state.people;
        people.push({name:name,surname:surname});
        this.setState({people:people});
    }

    render(){
        const list = this.state.people;
        return(
            <div>
                {list.map((person,index) => {
                    <Person person={person} key={index}/>
                })}

                <AddPerson onAdd={this.addPerson}/>
            </div>

        );
    }
}
```

```
export class AddPerson extends React.Component{
    constructor(props){
        super(props);
        this.nameRef = React.createRef();
        this.surnameRef = React.createRef();
        this.submitForm = this.submitForm.bind(this);
    }

    submitForm(e){
        e.preventDefault();
        this.props.onAdd(this.nameRef.current.value,this.surnameRef.current.value);
    }

    render(){
        return(
            <form onSubmit={submitForm}>
                <input type="text" ref={this.nameRef}/>
                <input type="text" ref={this.surnameRef}/>
                <button type="submit">Submit</button>
            </form>
        );
    }

}
```

```
export class Person extends React.Component{
    static defaultProps = {
        name:"",
        surname:""
    }

    constructor(props){
        super(props);
    }

    render(){
        const person = {name:this.props.name, surname:this.props.surname};
        return(
            <div>
                <p>${person.name}</p>
                <p>${person.surname}</p>
            </div>
        );
    }
}
```

# Babel

npm init

npm install express

npm install --save-dev @babel/cli @babel/core @babel/preset-env

npm install react react-dom

npm i --save-dev @babel/preset-react babel-loader webpack webpack-cli css-loader style-loader webpack-dev-server

npm i postcss postcss-loader react-router-dom path

```
db.collection.find({ <filter> }, { <projection> })
```

# Mongo Queries

Basic layout

Filter age greater than 25 and only show name and age fields:

```
db.users.find({ age: { $gt: 25 } }, { name: 1, age: 1, _id: 0 })
```

`$eq` – Equal to

`$ne` – Not equal to

`$gt` – Greater than

`$gte` – Greater than or equal to

`$lt` – Less than

`$lte` – Less than or equal to

`$in` – Matches any value in an array

`$nin` – Matches none of the values in an array

`$and` – Requires all conditions to be true

`$or` – Requires at least one condition to be true

`$not` – Inverts the effect of a query

`$nor` – Matches documents that fail both conditions

`$exists` – Checks if a field exists

`$type` – Checks the type of a field

`$all` – Matches all elements in an array

`$elemMatch` – Matches documents with an array that contains at least one element matching all criteria

`$size` – Matches documents with arrays of a specific size

Find all products between 50 and 100:

```
db.products.find({ price: { $gte: 50, $lte: 100 } })
```

Find users from NY or older than 3o:

```
db.users.find({
  $or: [{ city: "New York" }, { age: { $gt: 30 } }]
})
```

Find all where phone field exists:

```
db.contacts.find({ phone: { $exists: true } })
```

Products that contain red and blue

```
db.products.find({ colors: { $all: ["red", "blue"] } })
```

Find with at least one address in CA (Address is an array)

```
db.users.find({
  addresses: { $elemMatch: { state: "California" } }
})
```

## Sorting

1 = ascending -1 = descending

```
db.users.find({}, { name: 1, age: 1, _id: 0 })
```

Sort by age

```
db.users.find().sort({ age: -1 })
```

Sort first by name then surname

```
db.collection.find().sort({ surname: 1, name: 1 })
```

```
db.collection.aggregate([
  {
    $unwind: "$prices"  // Unwind the array to get individual price values
  },
  {
    $group: {
      _id: "$productName",   // Group by product name
      avgPrice: { $avg: "$prices" }  // Calculate the average price
    }
  },
  {
    $sort: { avgPrice: -1 }  // Sort by average price in descending order
  }
])
```

Count of each product type

```
db.collection.aggregate([
  {
    $group: {
      _id: "$productType",  // Group by productType
      count: { $sum: 1 }    // Count the number of documents for each type
    }
  }
])
```

```
db.products.aggregate([
  {
    $project: {
      productName: 1,
      price: 1,
      discountedPrice: { $multiply: ["$price", 0.9] }
    }
  }
])
```

Count products in category:

```
db.products.aggregate([
  { $match: { productType: "Electronics" } },
  { $count: "totalElectronics" }
])
```

Sum all prices

```
 db.products.aggregate([
   { $group: { _id: null, totalRevenue: { $sum: "$price" } } }
 ])
```

Average of all prices

```
db.products.aggregate([
  { $group: { _id: null, avgPrice: { $avg: "$price" } } }
])
```

```
db.products.aggregate([
  {
    $group: {
      _id: null,
      minPrice: { $min: "$price" },
      maxPrice: { $max: "$price" }
    }
  }
])
```

# Links

https://docs.google.com/document/d/1eG4hCx6ICryRWsafVUweGgjHOjZuLlvJKVtTiiSZ3G0/edit?usp=sharing

https://github.com/hayley-d/IMY220