learn JavaScript

how to create an Object

1. **Object Literal**: You can define an object using object literal notation by enclosing key-value pairs within curly braces {}:

```
let person = {
    name: 'John',
    age: 30,
    isStudent: false
};
```

2. Using the **Object Constructor**: You can also create an object using the Object constructor:

```
let person = new Object();
person.name = 'John';
person.age = 30;
person.isStudent = false;
```

3. Using a **Constructor Function**: You can define a constructor function and then create objects using the new keyword:

```
function Person(name, age, isStudent) {
    this.name = name;
    this.age = age;
    this.isStudent = isStudent;
}
let person = new Person('John', 30, false);
```

Express JS - TODO: To be comleted

- Useful resources
  - body-parser: Node.js body parsing middleware.
- **Middleware** in the context of web development, particularly in frameworks like Express.js for Node.js, refers to ***functions that have access*** to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle.

- following line of code is used to use bodyParser middleware in express.js. `app.use()` is usedto mount middleware .

```
app.use(bodyParser.urlencoded({ extended: true }));
```

Section 30: Build your own API

- useful resources
  - Rapid API: it allows people to host APIs. it is like Amazon for APIs

what makes an API to monetised?

- data collection: Covid data
- Algorithm/ service: Google map
- simplified interface: webdriverIO

Rest API

- stands for Representational State Transfer Application Programming Interface.
- what make an API restful?
  - **use HTTP Methods**: Get, post, put, patch, delete
  - it should have a **standard data format** that it responds with (Json/xml)
  - **Client-Server Architecture**: The client and server are separate from each other and can evolve independently.
  - **Statelessness**: Each request from the client to the server must contain all the information necessary to understand and fulfill that request. The server doesn't store any client state between requests.

Get a random jokes

```
app.get("/random", (req, res) => {
  const randomJoke = jokes[Math.floor(Math.random() * jokes.length)];
  res.json(randomJoke);
});
```

> Math.random() * jokes.length: This expression generates a random number between 0 (inclusive) and the length of the jokes array (exclusive). Multiplying Math.random() by jokes.length scales the random number to the size of the jokes array.

> When res.json(randomJoke); is executed, Express will automatically set the appropriate headers to indicate that the content being sent back is JSON and then serialize the randomJoke object into JSON format before sending it back to the client as the response body.

Get a specific joke

- differences between query parameters and path variables **Query Parameters:**

    1. Query parameters are key-value pairs that are appended to the end of a URL after a question mark (?).
    2. They are used to provide additional information to the server about the request.
    3. Query parameters are typically optional and can be used to filter, sort, or paginate data.Example: https://example.com/api/resource?key1=value1&key2=value2

    Path Variables (Path Parameters):

    1. Path variables are part of the URL path itself and are used to identify a specific resource or endpoint.
    2. They are included in the URL path and are preceded by a colon **:**
    3. Path variables are used to provide information about the resource being accessed or manipulated.
    4. They are often used to retrieve specific data or perform specific actions on a resource. Example: https://example.com/api/resource/{id}

```
app.get("/jokes/:id", (req, res) => {
  const id = parseInt(req.params.id);
  const foundJoke = jokes.find((joke) => joke.id === id);
  res.json(foundJoke);
});
```

- req.query vs req.params

> req.params contains route parameters (in the path portion of the URL), and req.query contains the URL query parameters (after the ? in the URL).

- Array.prototype.find()

> The find() method of Array instances returns the first element in the provided array that satisfies the provided testing function. If no values satisfy the testing function, undefined is returned. `const found = array1.find((element) => element > 10);`

- parseInt()

> The parseInt() function parses a string argument and returns an integer

---

Filter Jokes

```
app.get("/filter", (req, res) => {
  const type = req.query.type;
  const filteredActivities = jokes.filter((joke) => joke.jokeType === type);
  res.json(filteredActivities);
});
```

- Array.prototype.filter()

> The filter() method of Array instances creates a shallow copy of a portion of a given array, filtered down to just the elements from the given array that pass the test implemented by the provided function.

Post a new Joke

```
app.post("/jokes", (req, res) => {
  const newJoke = {
    id: jokes.length + 1,
    jokeText: req.body.text,
    jokeType: req.body.type,
  };
  jokes.push(newJoke);
  console.log(jokes.slice(-1));
  res.json(newJoke);
});
```

- since ou data is stored in array of objects, we create an object and push it to the array.
- we are passing data using ***body >> x-www.form-urlencoded*** in postman. to read this we will be using `req.body.text`
- **Array.prototype.slice()**: This is an array method in JavaScript that returns a shallow copy of a portion of an array into a new array object.The argument ***-1*** passed to the slice() method indicates that you want to start slicing the array from the end, counting backward from the last element.

Put a Joke

```
app.put("/jokes/:id", (req, res) => {
  const id = parseInt(req.params.id);
  const replacementJoke = {
    id: id,
    jokeText: req.body.text,
    jokeType: req.body.type,
  };

  const searchIndex = jokes.findIndex((joke) => joke.id === id);

  jokes[searchIndex] = replacementJoke;
  // console.log(jokes);
  res.json(replacementJoke);
});
```

- **PUT** requests are used to update or replace an existing resource ***entirely*** with the data provided in the request payload.

- **parseInt()**: parses a string argument and returns an integer
- **Array.findIndex()**:returns the index of the first element in an array that satisfies the provided testing function

---

Patch a Joke

```
app.patch("/jokes/:id", (req, res) => {
  const id = parseInt(req.params.id);
  const existingJoke = jokes.find((joke) => joke.id === id);
  const replacementJoke = {
    id: id,
    jokeText: req.body.text || existingJoke.jokeText,
    jokeType: req.body.type || existingJoke.jokeType,
  };
  const searchIndex = jokes.findIndex((joke) => joke.id === id);
  jokes[searchIndex] = replacementJoke;
  console.log(jokes[searchIndex]);
  res.json(replacementJoke);
});
```

- **PATCH** requests are used to apply ***partial modifications*** to an existing resource. It's used when you want to update specific fields or properties of a resource without necessarily sending the entire representation
- `jokeText: req.body.text || existingJoke.jokeText`: This line sets the jokeText property of replacementJoke. It looks for text property in the request body (req.body.text). If it's not present **(|| operator acts as a fallback)**, it uses the jokeText property from the existingJoke object, if available.
- **fallback** mechanism provides an alternative or default option when the primary choice is not available or cannot be used.

DELETE Specific joke

Section 32: SQL

- useful resouces
  - w3School
  - SQL Playground: used for practice only

CREATE TABLE Statementv

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

```
// sample
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

- **Primary keys** must contain UNIQUE values, and cannot contain NULL values.

INSERT INTO Statement

1. Specify both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

SELECT statement

```
SELECT column1, column2, ...
FROM table_name;
```

WHERE Clause

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

- The WHERE clause is used to filter records.

UPDATE Statement

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
```

```
WHERE condition;
```

- The UPDATE statement is used to modify the existing records in a table.
- Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

- **ADD Column**

```
ALTER TABLE table_name
ADD column_name datatype;
```

- **DROP COLUMN**

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

- **RENAME COLUMN**

```
ALTER TABLE table_name
RENAME COLUMN old_name to new_name;
```