QUIZ

What will the result be after running the following code?

```
db.run("DROP TABLE Employee", error => {
  db.run(`CREATE TABLE Employee(
    ID INT NOT NULL,
    NAME CHAR(20) NOT NULL,
    PRIMARY KEY(ID))`, error => {
    db.run("INSERT INTO Employee [(id, name)] VALUES (1, 'Sophie')", error => {
    })
  })
});
```

The code will successfully `DROP`, `CREATE`, and `INSERT` a new row into the newly recreated database.

 Correct! Although rewriting to use `db.serialize()` will make it easier to read.

The code will raise an error when an `INSERT` gets called before the table is created.

Where would an error appear in the following methods: `db.all()`, `db.get()`, `db.run()`, and `db.each()`?

The error gets bound as a property to `this` in the callback.

The error gets printed to the web browser's console.

The first argument in the callback will be an `Error` object.

 Correct!

What should go in the blank so that the function `retrieve_name` prints the records matching the first name given to it?

```
const retrieve_name = name => {
  db.get("SELECT * FROM Employee WHERE first_name=$employeeName",
  _________________,
  (error, rows) => {
    console.log(row);
  });
}
```

```
{ $employeeName: name }
```

👏 Correct!

```
const $employeeName = name;
```

What method should be used so that each row gets added to the JavaScript object?

```
let employee = {}
db.____("SELECT * FROM Employee",
  (error, row) => {
    employee[row.id] = row.name
  }
);
```

```
.each()
```

👏 Correct!

```
.serialize()
```

```
.get()
```

Which of the following would create a new SQLite database at the path `./newdb.sqlite`?

```
const db = new sqlite3.createDB('./newdb.sqlite');
```

```
const db = new sqlite3.Database('./newdb.sqlite’);
```

👏 Correct!

```
const db = sqlite3.Database('./newdb.sqlite');
```

Which of these methods should be used with the following query in order to retrieve all books written by Mark Twain?

```
SELECT * FROM Book WHERE author='Mark Twain';
```

```
db.every()
```

```
db.get()
```

```
db.all()
```

👏 Correct!

```
db.many()
```

This code is intended to insert a new row into the `Pasta` table and then retrieve it and log it out. Why would this code not work as intended?

```
db.run('INSERT INTO Pasta (pasta_type, sauce) VALUES ("spaghetti", "carbonara")',
  (error) => {
    if (error) {
      return console.log(error);
    }
    db.get(`SELECT * FROM Pasta WHERE id = ${this.lastID}`, (error, row) => {
      if (error) {
        return console.log(error);
      }
      console.log(row);
    });
  }
);
```

The callback function for `db.run()` is an arrow function, instead of a `function` keyword function, so `this` will be bound incorrectly and not have a `lastID` property.

 You got it!

What needs to be done for the database to run multiple commands chronologically?

Pass an argument to `db.run()`, called `stepIndex`, that will indicate which step it should run as.

Use the `db` method call `.serialize()` and then write the commands in order.

 Correct!

Write the commands in order, the database will process them as it gets them.

Which command would you use to create a new table, `Employee`?

```
db.run("CREATE TABLE Employee");
```

 Correct!

```
db.get("CREATE TABLE Employee");
```

```
db.serialize("CREATE TABLE Employee");
```

Which of the following methods will return either a single row or `undefined`?

`db.serialize()`

`db.all()`

`db.get()`

👏 Correct!

`db.run()`

What should we do to guarantee the following code runs as expected?

```
db.run("DROP TABLE Employee");
db.run(`CREATE TABLE Employee(
    ID INT NOT NULL,
    NAME CHAR(20) NOT NULL,
    PRIMARY KEY(ID))`);
db.run("INSERT INTO Employee (id, name) VALUES (2, 'Quinn')");
```

Wrap the code in a call to `db.each()`

Run it, looks good to me!

Wrap the code in a call to `db.serialize()`.

👏 Correct!

Which of these methods doesn't return row data from the database?

```
db.get()
```

```
db.all()
```

```
db.run()
```

 Correct!

```
db.fetch()
```