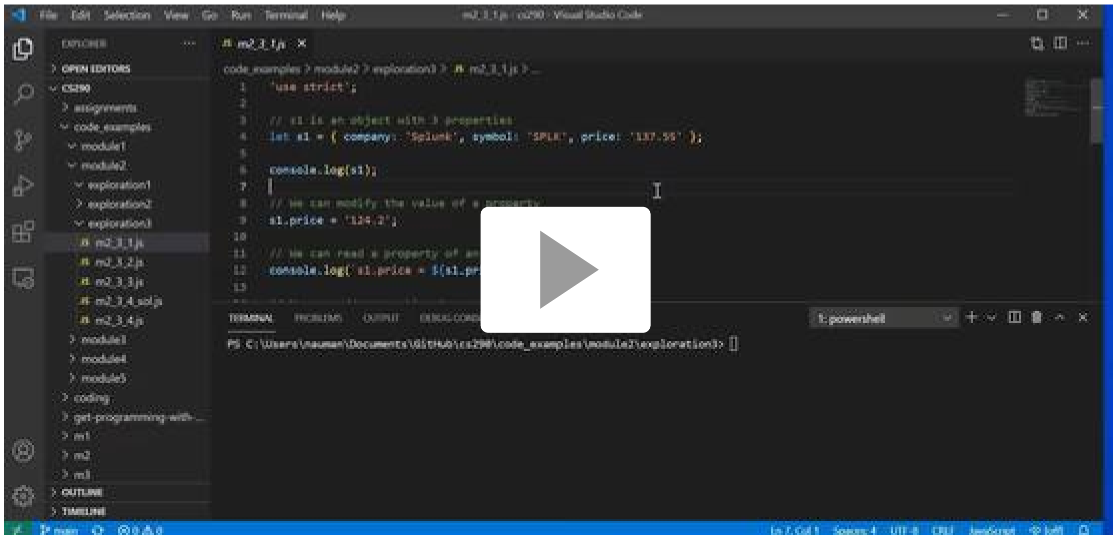
Exploration — Objects, Arrays and JSON

Introduction

In this exploration we look at objects, which are the only non-primitive type of values in JavaScript. The original object type provided by JavaScript is not very powerful. However, it is extensively used in JavaScript code and we will study it in this exploration. In a later exploration, we will study the enhanced support for object-oriented programming that is provided by modern JavaScript.



```
'use strict';

// s1 is an object with 3 properties
let s1 = { company: 'Splunk', symbol: 'SPLK', price: '137.55' };

console.log(s1);

// We can modify the value of a property
s1.price = '124.2';

// We can read a property of an
console.log(`s1.price = ${s1.pr
```

```
PS C:\Users\nauman\Documents\GitHub\cs290\code_examples\module2\exploration3>
```

0:00 / 9:48

JavaScript Objects

A JavaScript object is a set of name-value pairs. The names of these pairs are also called properties of the object. We can create (or add), read, update and delete properties of the object.

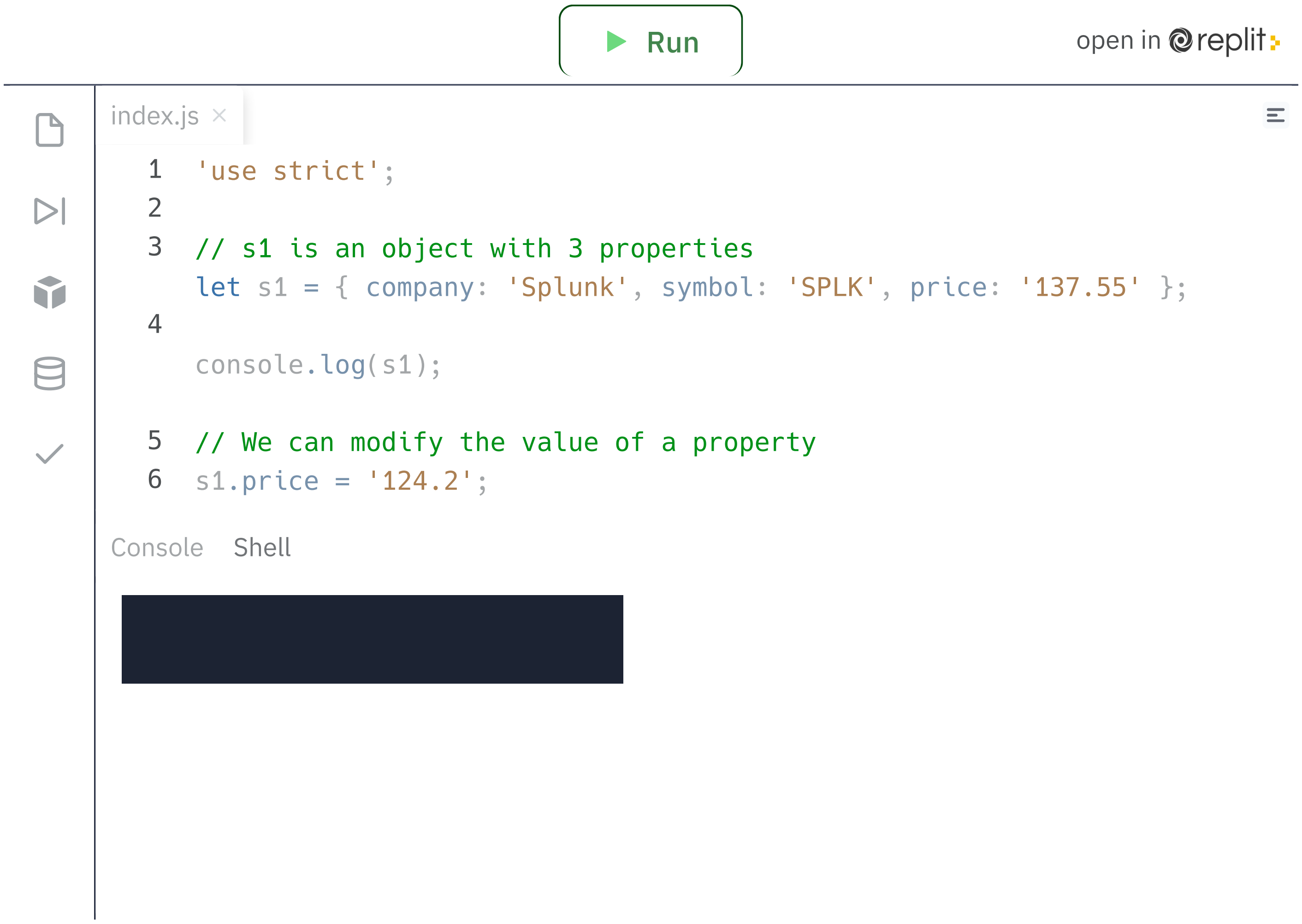
- Create and Update
  - Adding or updating a property is done using the `.` operator on the variable in an assignment statement.
    - For example, `s1.price = 10` will set the value of the `price` property of the object `s1` to 10. If the property `price` already existed, its value will be updated. If the property `price` didn't exist, it will be added.

- Read
    - A property is read using a `.` operator as well.
        - For example, `s1.price` will read the `price` property of the object `s1`
    - Another way to read a property of an object is using the bracket operator `[]`
        - For example, `s1["price"]` will read the `price` property of the object `s1`
    - Reading a non-existent property of an object returns the value `undefined`.
- Delete
    - A property is deleted using the `delete` operator.

A `const` variable of object type cannot be reassigned to another value. However, we can modify the properties of the object.

Example

In the following example, we declare two variables `s1` and `s2`, and assign object values to each of them. The variable `s2` is declared as const. This means that an attempt to assign another object to `s2` is rejected. But we can still change properties of `s2`.



```
'use strict';

// s1 is an object with 3 properties
let s1 = { company: 'Splunk', symbol: 'SPLK', price: '137.55' };

console.log(s1);

// We can modify the value of a property
s1.price = '124.2';
```

Getting Names of All Properties of an Object

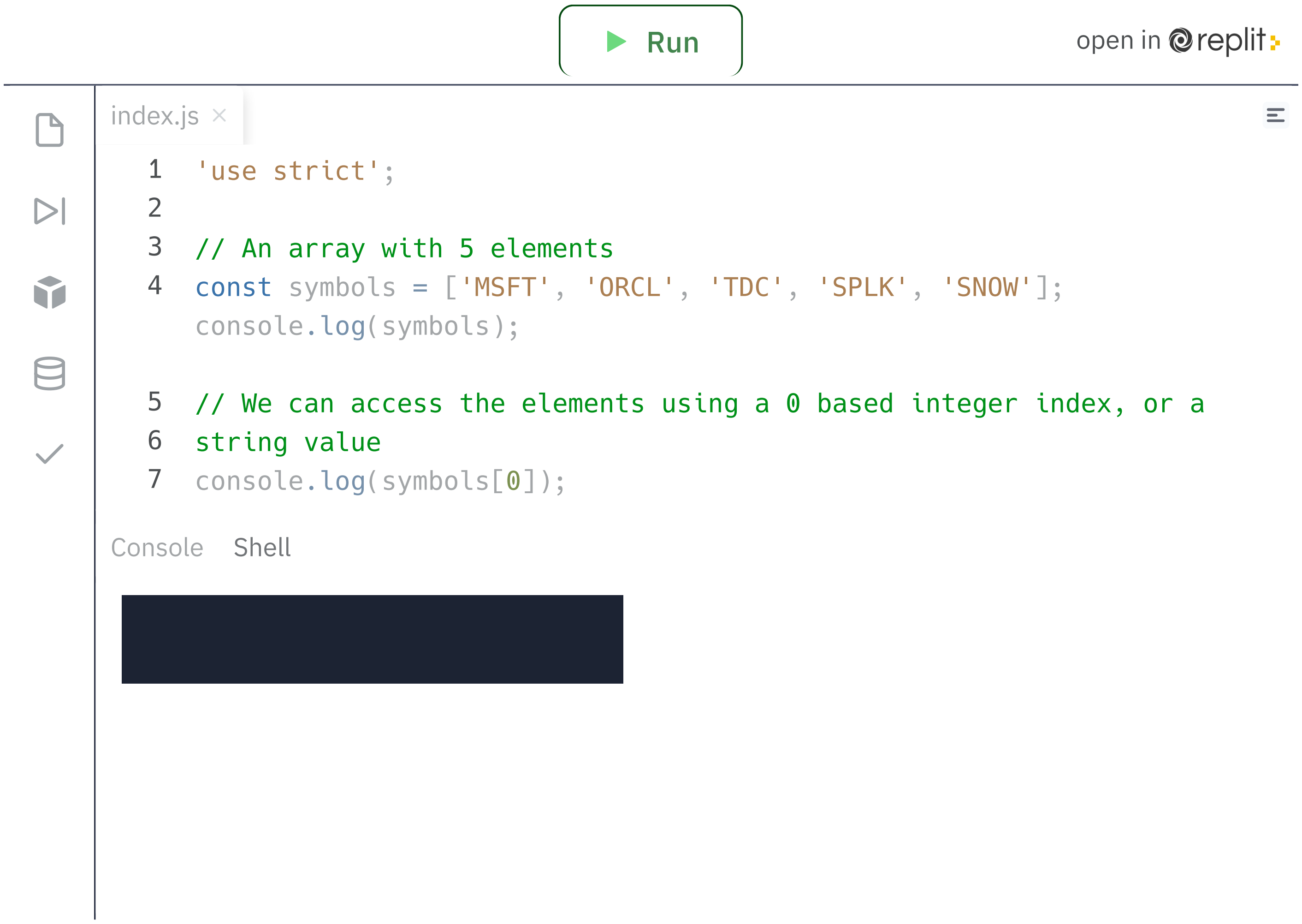
We can get the names of all properties of an object by passing it to the method Object.keys() (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys) . This method returns the names of the properties in an array, as shown in the following example:

```
> let s1 = { company: 'Splunk', symbol: 'SPLK', price: '137.55' };
undefined
> console.log(Object.keys(s1));
[ 'company', 'symbol', 'price' ]
undefined
>
```

Arrays

Arrays in JavaScript are objects whose property names are the strings '0', '1', '2', etc. JavaScript requires that the properties of an object must be of string data type. This is why the property names of an array are string values rather than the numbers 0, 1, 2, etc. Here is an example to illustrate some basic concepts about JavaScript arrays.

Example



```
'use strict';

// An array with 5 elements
const symbols = ['MSFT', 'ORCL', 'TDC', 'SPLK', 'SNOW'];
console.log(symbols);

// We can access the elements using a 0 based integer index, or a
string value
console.log(symbols[0]);
```

We highlight some aspects of arrays:

- We can access the elements of an array using a 0-based integer index in square brackets, e.g., `[0]`, in addition to using the string based property name, i.e., `['0']`.
- The values in an array can be of any JavaScript type, including an object.
- Accessing the element at an index where there is no element returns the value `undefined`.
- We can use the method **Array.isArray()** **(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/isArray)** to determine if the value of a variable is an array or not.
- A `const` variable of array type cannot be reassigned to another value. However, we can modify the elements of the array.
- The property **length** **(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/length)** of an array returns the number of elements in the array.
- An array provide the method **push()** **(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/push)** to add one or more elements at the end of the array, and the method **pop()** **(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/pop)** to remove and return the last element in the array.
- The method **includes()** **(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/includes)** can be called on an array to determine if a value exists among the elements of the array.

Exercise

Write a function `hasProperty(obj, prop)` which returns `true` if the object `obj` has the property `prop` and `false` if it doesn't have that property? For example, `hasProperty({a: 1, b: 2}, 'a')` returns `true` while `hasProperty({a: 1, b: 2}, 'c')` returns `false.`

JSON

JSON stands for **JavaScript Object Notation** and is a very widely used format for exchanging data between applications. Even though the name JSON includes the word JavaScript, the JSON format is programming language independent. Most programming languages provide libraries for JSON, including JavaScript, Python, Java, C#, etc.

The basic idea is that objects may be represented differently in different programming languages. But using JSON, we can

- Map an object in a program to a string in the JSON format, as well as
- Create an object in a program from a string in the JSON format.

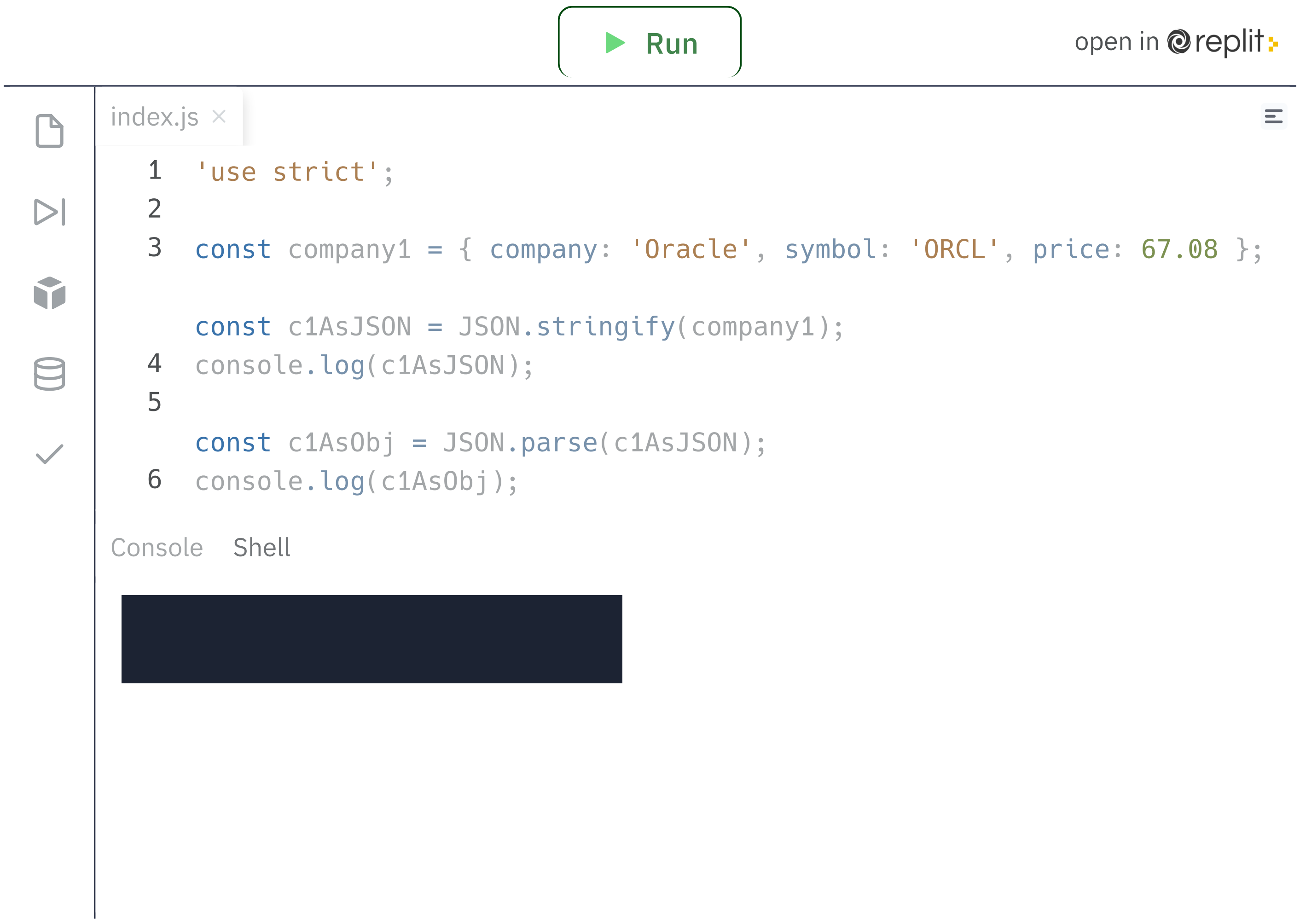
Since the JSON format is independent of any programming language, this allows programs to exchange data even when the programs exchanging the data are written in different programming languages.

The two primary methods for using JSON in JavaScript are:

- **JSON.stringify()** (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify) for creating a JSON string from a JavaScript object, and
- **JSON.parse()** (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse) for creating a JavaScript object from a JSON string.

As noted in Section 1.15 in the textbook, `JSON.stringify()` is also used for logging objects on the console.

Example



```
'use strict';

const company1 = { company: 'Oracle', symbol: 'ORCL', price: 67.08 };

const c1AsJSON = JSON.stringify(company1);
console.log(c1AsJSON);

const c1AsObj = JSON.parse(c1AsJSON);
console.log(c1AsObj);
```

Examining the value returned by `JSON.stringify()` in the above example, we note that in a string in JSON format:

- The names of the properties of the object are delimited by double quotes.
- All string values are delimited by double quotes.
- Array elements are separated by commas.

typeof operator

JavaScript provides the operator **typeof (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof)** which we can use to find the type of a value.

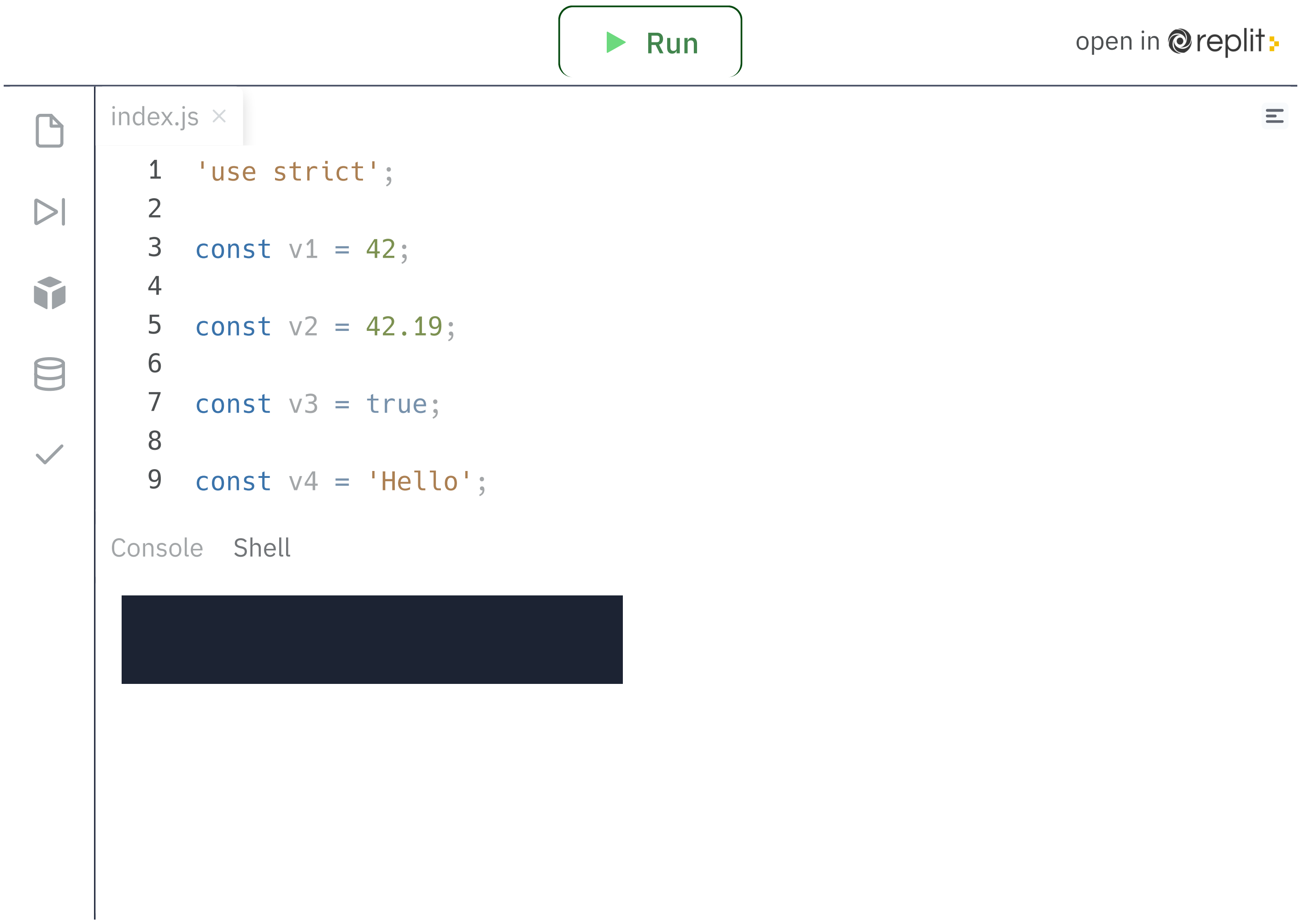
Example

Here is what `typeof` prints on the console when called on the values `42` and `'42'`.

```
> typeof 42
'number'
> typeof '42'
'string'
```

Exercise

Find the type of the values for each of the variables v1 through v11.



```javascript
'use strict';

const v1 = 42;

const v2 = 42.19;

const v3 = true;

const v4 = 'Hello';
```

Golden Rule 1 and Scope

Let us briefly look at the motivation for Golden Rule #1

Golden Rule 1. Declare variables with `let` or `const`, not `var`.

This rule is related to the scope of declaration. The scope of a variable or a function is the part of the program where that variable can be accessed.

Block scope

A block in JavaScript is delimited by curly braces, i.e., `{}`. A whole file is also at the block scope.

Global Scope

A variable or a function declared at the global scope is accessible everywhere in the program. A global variable of function becomes a property of the **global object (https://developer.mozilla.org/en-US/docs/Glossary/Global_object)**.

- For a program running in Node, the global object is the variable named `global.`
- For a program running in the web browser, the global object is the variable named `window`.

Function Scope

A variable or a function defined within a function is said to have function scope.

Using let vs. var

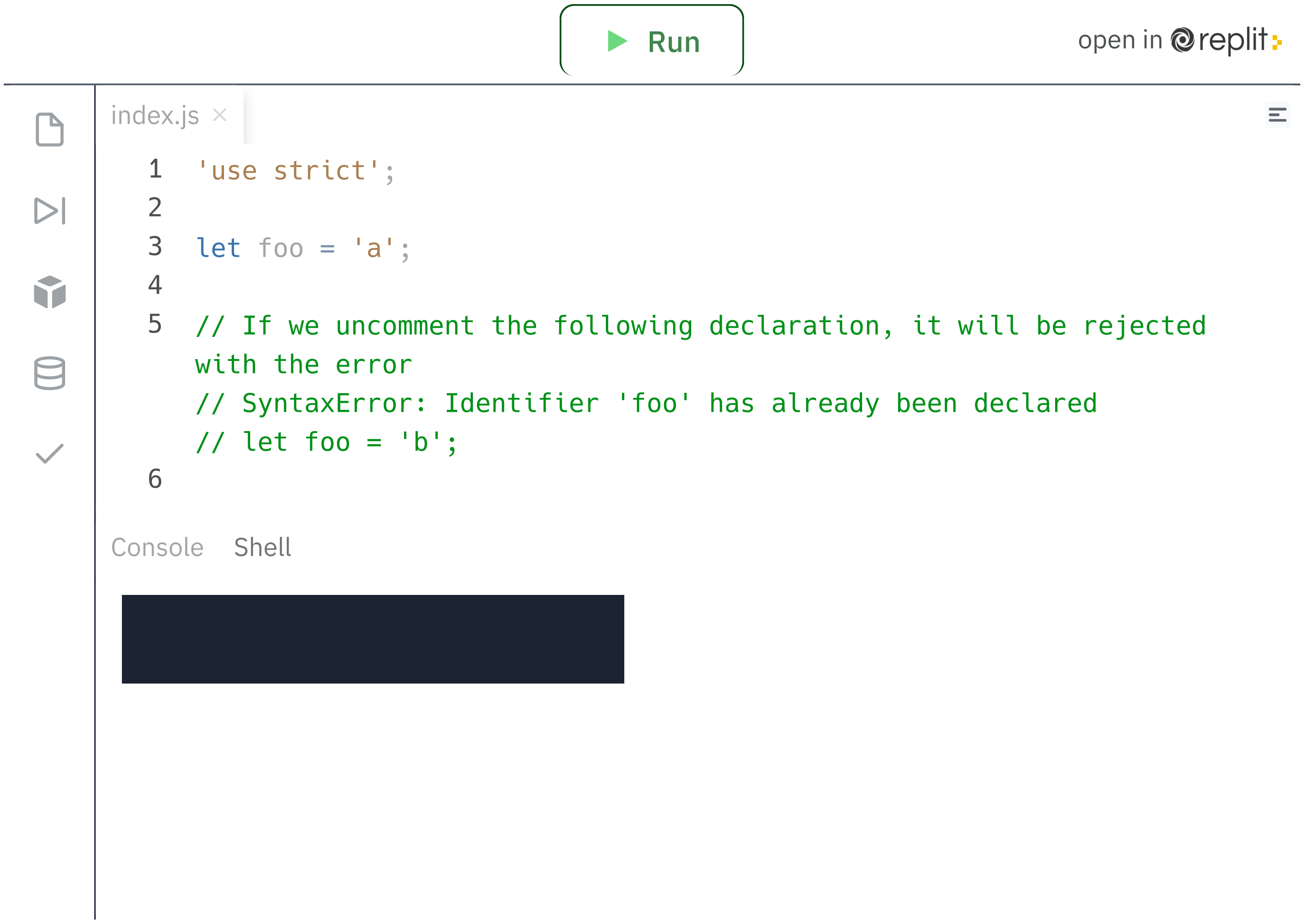
A variable or function declared with `let` or `const` is at the block scope. It can only be accessed within this block. This means that if we have two declarations of a variable with the same name using `let` in two different blocks, these are two different variables and modifying the value of one variable will not interfere with the value of the other variable. Furthermore, when we use strict mode, JavaScript will not allow multiple declarations of a variable with the same name in a block if we are using `let`.

As opposed to this, when we declare a variable with `var`, its scope is either global, or if such a variable is declared within a function then its scope is within the function. This means that two declarations of a variable with the same name using `var` in the same scope are considered the same variable. Let us say, we declare a variable using `var` outside a function in one file, and then have another declaration with the same name using `var` later in the same file again outside a function. This declaration will attach the variable with the global object and both the declarations refer to the same variable.

Example

In the following example, we declare `foo` using `let`. If we try to declare it again in the same block, JavaScript will raise an error. We can declare another variable with the name `foo` in a different block. We see that changing the value of this variable doesn't change the value of the variable `foo` declared in the different block.

As opposed to this, when we have two declaration of a variable named `baz` using `var`, these are considered the same variable. Changing the value of `baz` in one block changes the value of `baz` in the other block.



```javascript
'use strict';

let foo = 'a';

// If we uncomment the following declaration, it will be rejected with the error
// SyntaxError: Identifier 'foo' has already been declared
// let foo = 'b';

```

Summary

In this exploration, we studied JavaScript objects and arrays. We also learned about JSON which is a very important and widely used format for exchanging data in web applications.

Additional Resources

Here are some references to learn more about the topics we discussed in this exploration.

- **Working with Objects** **(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)** on MDN.
- Discussion of **arrays** **(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)** on MDN.
- The formal JSON specification is available at **here** **(https://www.json.org/json-en.html)**.
- MDN's **page on JSON** **(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON)** includes examples and details of the

methods `JSON.stringify()` and `JSON.parse()`.