Variables

This lesson teaches the properties of a variable and how do we declare it in JavaScript.

We'll cover the following

Role of a Variable

A computer program stores data using variables. A *variable* is an information storage area. We can imagine it as a box in which you can put and store things!

Variable Properties

A variable has three main properties:

- Its *name*, which identifies it. A variable name may contain upper and lower case letters, numbers (not in the first position) and characters like the dollar sign (`$`) or underscore (`_`)
- Its *value*, which is the data stored in the variable
- Its *type*, which determines the role and actions available to the variable

> You don't have to define a variable type explicitly in JavaScript. Its type is deduced from the value stored in the variable and may change while

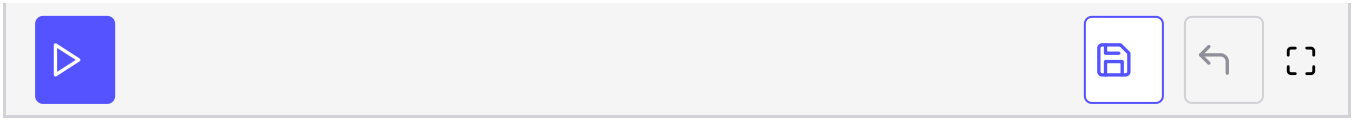
the program runs. That's why we say that JavaScript is a `dynamically typed` language. Other languages, like C or Java, require variable types to always be defined. This is called `static typing`.

Declaring a Variable

Before you can store information in a variable, you have to create it! This is called declaring a variable. `Declaring` a variable means the computer reserves memory in which to store the variable. The program can then read or write data in this memory area by manipulating the variable.

Here's a code example that declares a variable and shows its contents:

```
let a;
console.log(a);
```



In JavaScript, you declare a variable with the `let` keyword followed by the variable name. In this example, the variable created is called `a`.

> In previous versions of the language, variables were declared using the `var` keyword.

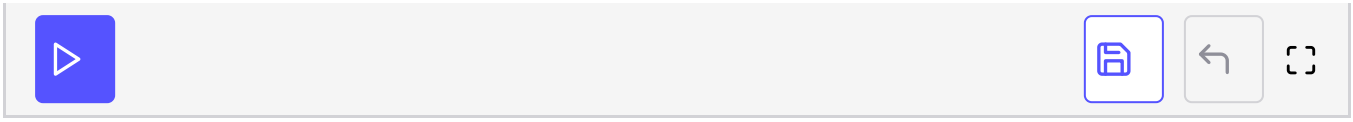
Note that the result is `undefined`. This is a special JavaScript type indicating no value. I declared the variable, calling it `a`, but didn't give it a value!

Assign Values to Variables

While a program is running, the value stored in a variable can change. To give a new value to a variable, use the `=` operator called the *assignment operator*. Check out the example below:

```
let a;
```

```
a = 3.14;
console.log(a);
```

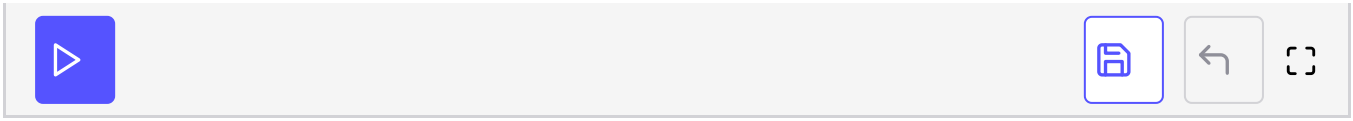


We modified the variable by assigning it a value. `a = 3.14` reads as “a receives the value 3.14”.

> Be careful not to confuse the assignment operator = with mathematical equality! You’ll soon see how to express equality in JavaScript.

You can also combine declaring a variable and assigning it a value in one line. Just know that, within this line, you’re doing two different things at once:

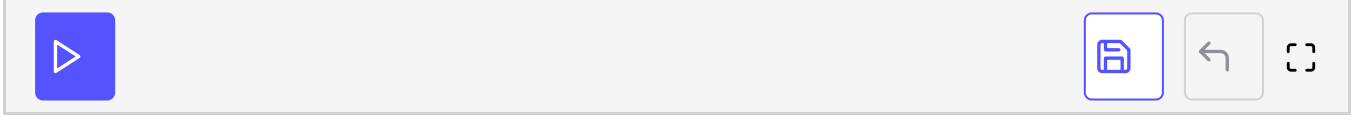
```
let a = 3.14;
console.log(a);
```



Declaring a Constant Variable

If the initial value of a variable won’t ever change during the rest of program execution, this variable is called a *constant*. This constantness can be enforced by using the keyword `const` instead of `let` to declare it. Thus, the program is more expressive and further attempts to modify the variable can be detected as errors.

```
const a = 3.14; // The value of a cannot be modified
a = 6.28;       // Impossible!
```

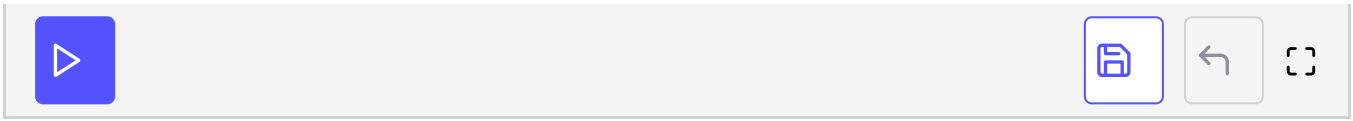


Increment a Number Variable

You can also increase the value of a number with `+=` and `++`. The latter is called the *increment operator*, as it allows incrementation (increase by 1) of a variable’s value.

In the following example, lines 2 and 3 each increase the value of variable b by 1.

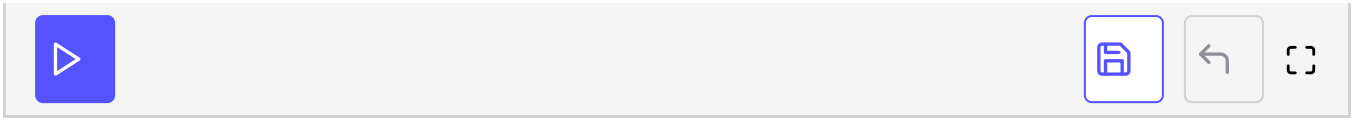
```
let b = 0;      // b contains 0
b += 1;         // b contains 1
b++;            // b contains 2
console.log(b); // Shows 2
```



Variable Scope

The *scope* of a variable is the part of the program where the variable is visible and usable. Variables declared with `let` or `const` are *block-scoped*: their visibility is limited to the block where they are declared (and every sub-block, if any). In JavaScript and many other programming languages, a *code block* is a portion of a program delimited by a pair of opening and closing braces. By default, a JavaScript program forms one block of code.

```
let num1 = 0;
{
  num1 = 1; // OK : num1 is declared in the parent block
  const num2 = 0;
}
console.log(num1); // OK : num1 is declared in the current block
console.log(num2); // Error! num2 is not visible here
```



This code will produce an error due to the scope defined