Lecture 2

Data Types & Variables

Topics

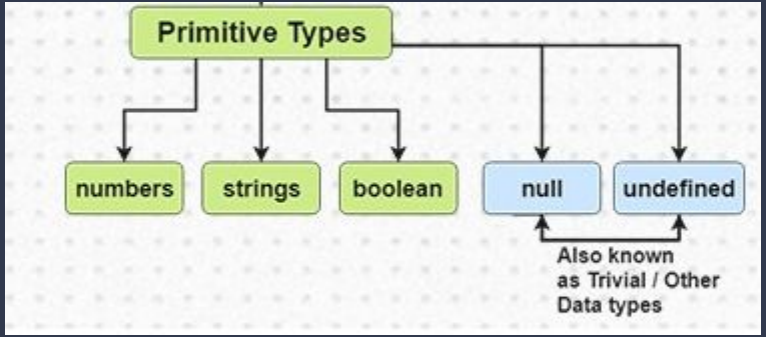
- Intro to Data Types
- Numbers
- Strings
  - Comments
- String Concatenation
- Variables

Data Types

Intro to Data Types

- data and data types are the building blocks to any programming language
- help us organize our information and determine how our programs are run
- it is important to be aware of types of data and when to use them
- We are going to learn how to define and manipulate the primitive data types of

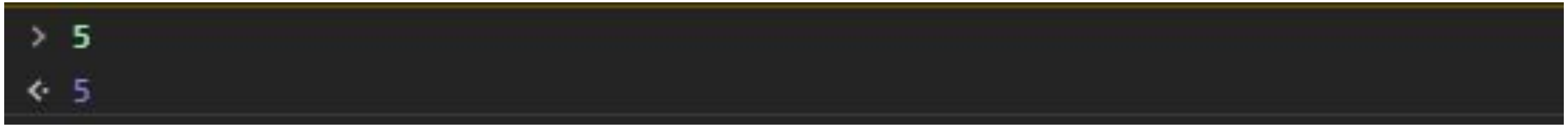
Primitive Data Types



- Primitive Data Types of JavaScript
  - numbers,
  - booleans,
  - Strings,
  - undefined
  - null types
- Once familiar with these data types, we can store data and into variables
- Data Types allow us to reuse and manipulate data throughout our code

Numbers

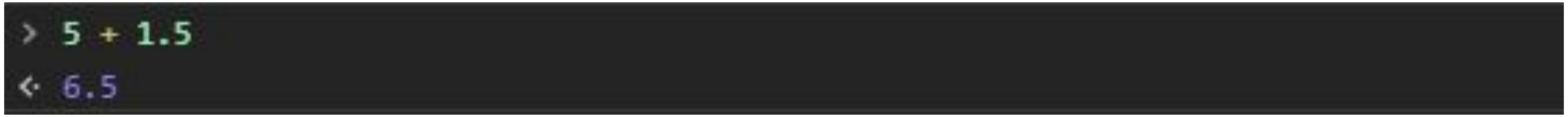
- Defining a number in JavaScript is actually pretty simple. The **Number** data type includes any positive or negative integer, as well as decimals. Entering a number into the console will return it right back to you.



```
> 5
<· 5
```

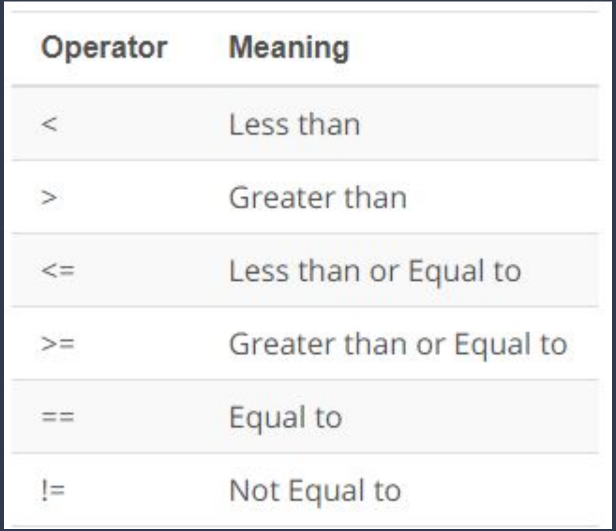
Arithmetic operations

- You can also perform calculations with numbers pretty easily. Basically type out an expression the way you would type it in a calculator.



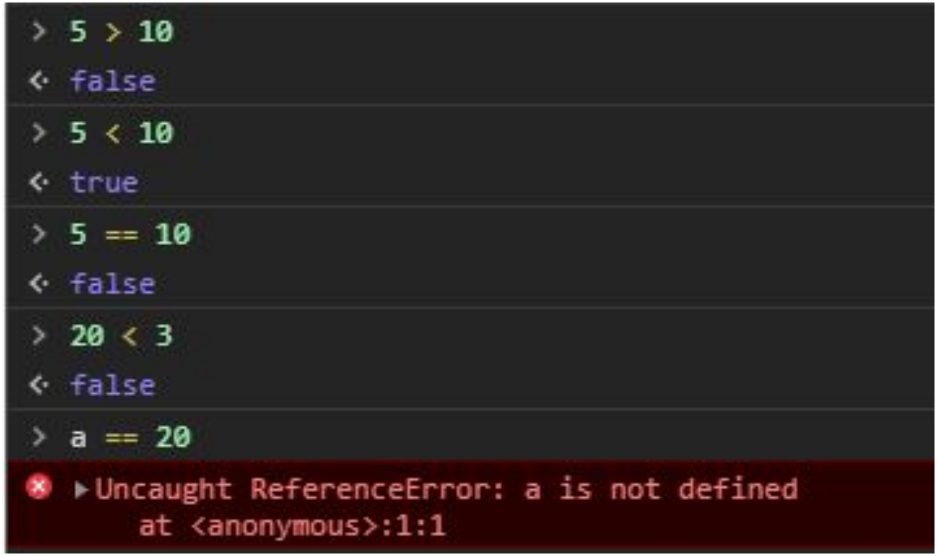
```
> 5 + 1.5
<· 6.5
```

Comparing Numbers



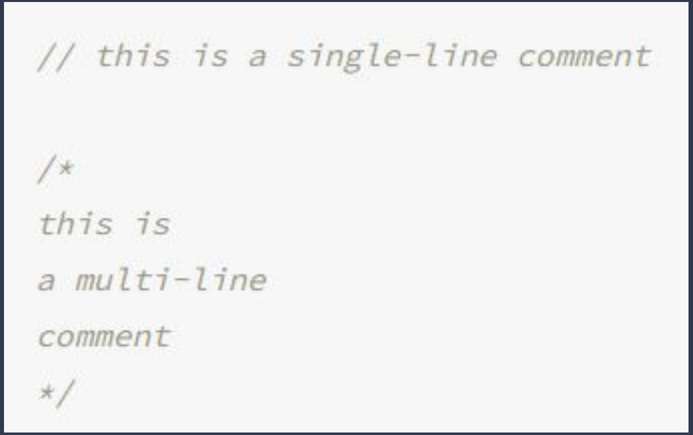
Operator	Meaning
<	Less than
>	Greater than
<=	Less than or Equal to
>=	Greater than or Equal to
==	Equal to
!=	Not Equal to

- Just like in mathematics, you can compare two numbers to see if one's greater than, less than, or equal to the other.



```
> 5 > 10
<· false
> 5 < 10
<· true
> 5 == 10
<· false
> 20 < 3
<· false
> a == 20
✖ ▶Uncaught ReferenceError: a is not defined
     at <anonymous>:1:1
```

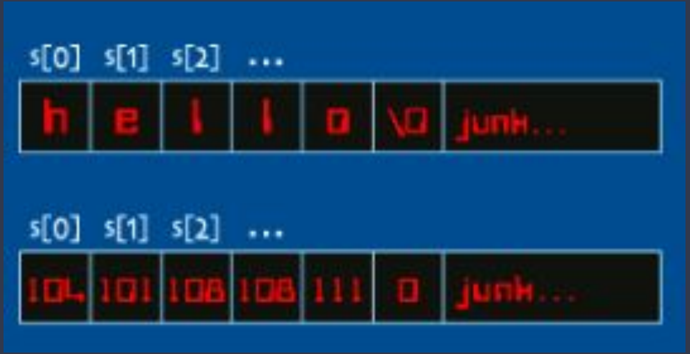
Comments



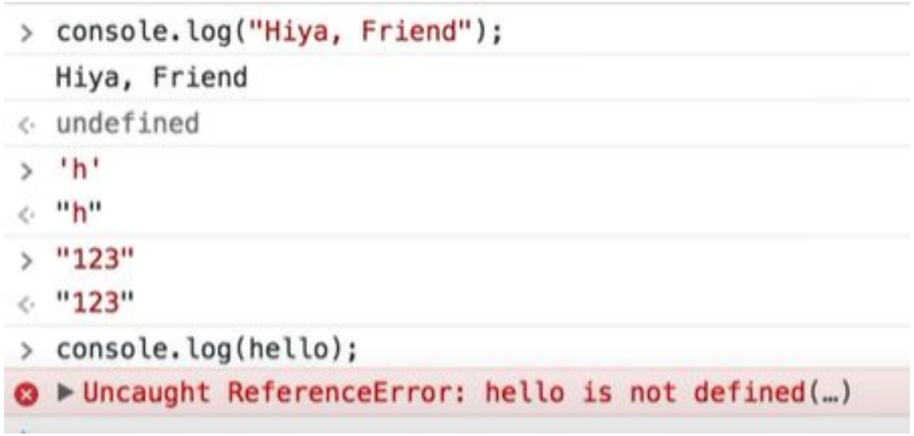
```
// this is a single-line comment

/*
this is
a multi-line
comment
*/
```

- You can use **comments** to help explain your code and make things clearer.
- In JavaScript, comments are marked with a double forward-slash `//`.
- Anything written on the same line after the `//` will not be executed or displayed.
- To have the comment span multiple lines, mark the start of your comment with a forward-slash and star, and then enclose your comment inside a star and forward-slash `/* … */`

Strings

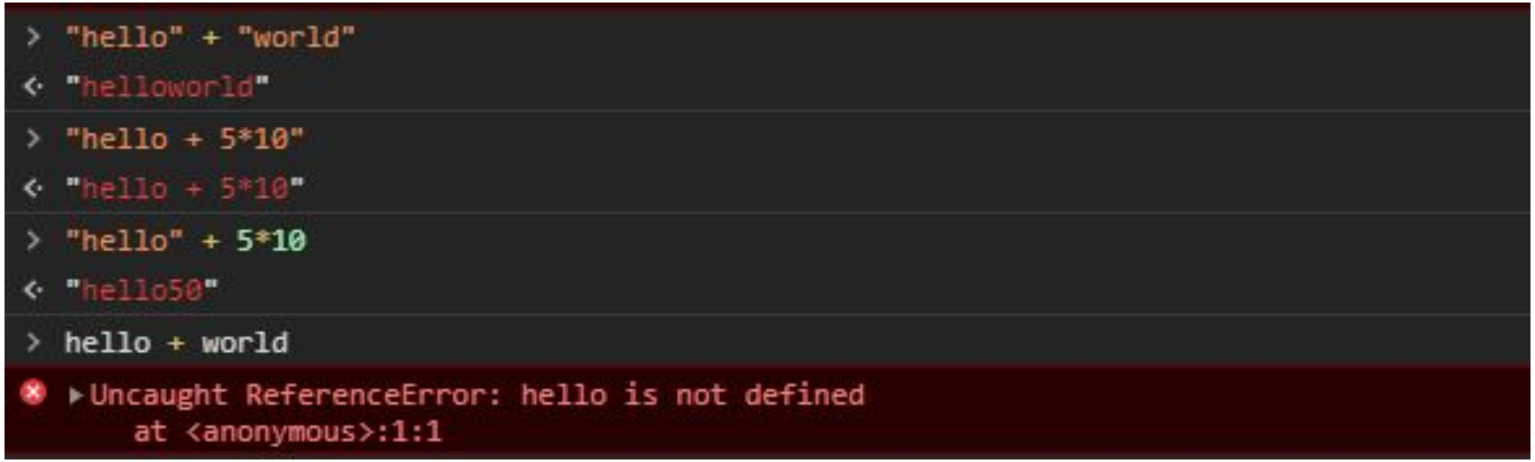


- A **string** (or a text string) is a series of characters like "John Doe"..
- You may have already used strings in conole.log("hello world") message
- *It is correct to either use double " or single ' quotes with strings, as long as you're consistent.*



```
> console.log("Hiya, Friend");
  Hiya, Friend
<· undefined
> 'h'
<· "h"
> "123"
<· "123"
> console.log(hello);
⊗ ▶Uncaught ReferenceError: hello is not defined(…)
```

- **Strings** are a collection of characters enclosed inside double or single quotes.
- You can use strings to represent data like sentences, names, addresses, and more.
- Concatenating two strings together is actually pretty simple!



```
> "hello" + "world"
<· "helloworld"
> "hello + 5*10"
<· "hello + 5*10"
> "hello" + 5*10
<· "hello50"
> hello + world
✖ ▸Uncaught ReferenceError: hello is not defined
      at <anonymous>:1:1
```

String Concatenation

Variables

Following are some basic definitions of variables in a typical programming language context.

- A variable provides us with a **named storage** that our programs can manipulate. It is the **basic unit of storage** in a program.
- Variables are used to store information to be referenced & manipulated in a computer program.
- In programming, a variable is a value that can be changed depending on the conditions or information being passed to the program.

Variable Naming Conventions

Recommended JavaScript Style Guide - Variable Naming.

- Write the name of the variable using **camelCase** (the first word is lowercase, and all following words are uppercase).
- Try to use a variable name that accurately describes what the data is about.

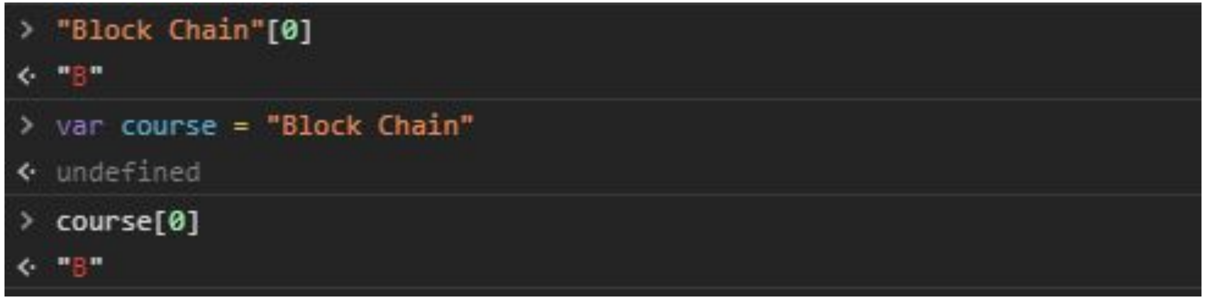
  Not using **camelCase**, is not going to break JavaScript, but it recommended to follow the style guides for clean, consistent and easy to read code.

```
var finalSubTotal = 53.03; // uses camelCase if the variable name is multiple words
var tax = 0.13; // uses lowercase if the variable name is one word
```

String Index

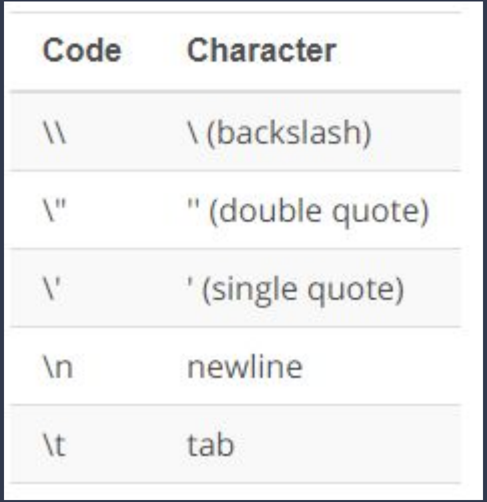
Indexing:

- To access an individual character, you can use the character's location in the string, called its **index**.
- Just put the index of the character inside square brackets (starting with `[0]` as the first character) immediately after the string. For example:



```
> "Block Chain"[0]
<· "B"
> var course = "Block Chain"
<· undefined
> course[0]
<· "B"
```

Escaping Strings

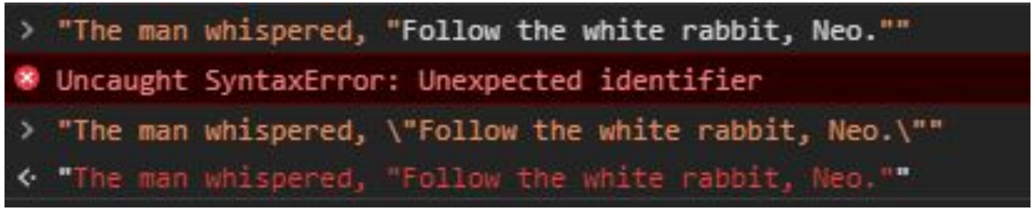


Code	Character
\\	\ (backslash)
\"	" (double quote)
\'	' (single quote)
\n	newline
\t	tab

- We might need to use quotes within a string. To prevent a SyntaxError will have to denote the beginning and end of the strings.

- To escape quotes, will need to use the backslash character (\) to escape the quotes.

- Escaping a character tells JavaScript to ignore the character's special meaning and just use the literal value of the character.



```
> "The man whispered, "Follow the white rabbit, Neo.""
✖ Uncaught SyntaxError: Unexpected identifier
> "The man whispered, \"Follow the white rabbit, Neo.\""
<· "The man whispered, "Follow the white rabbit, Neo.""
```

Comparing Strings

- Another way to work with strings is by comparing them. You've seen the comparison operators `==` and `!=` when you compared numbers for equality. You can also use them with strings! For example, let’s compare the string `"Hello"` to `"hello"`. This will return **false**.
- **Case-sensitive**
    - When you compare strings, case matters. While both string use the same letters (and those letters appear in the same order), the first letter in the first string is a capital `H` while the first letter in the second string is a lowercase `h`. `“H” != “h”` will return **true**
- **Internal Working**
    - In Javascript, strings are compared character-by-character in alphabetical order. Each character has a specific ***numeric*** value, coming from ASCII value of Printable characters. For example, the character 'A' has a value 65, and 'a' has a value 97.

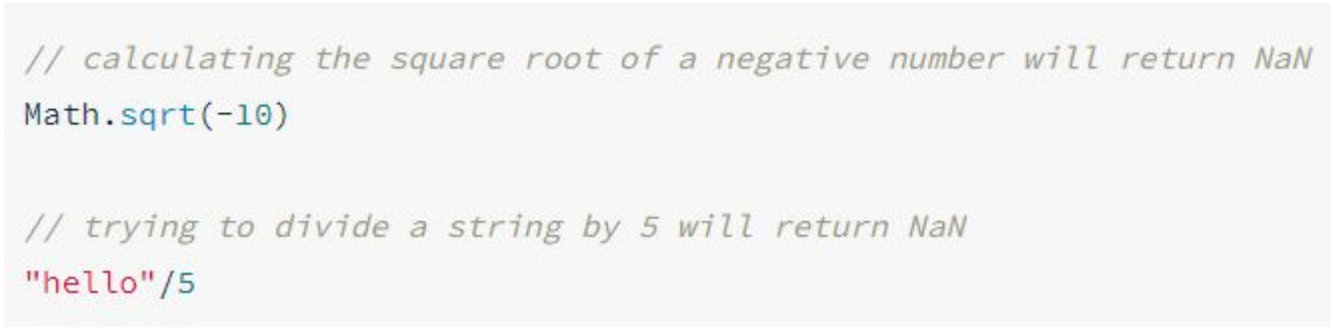
Booleans

- A boolean variable can take either of two values - `true` or `false`.
- A boolean variable is mainly essential in evaluating the outcome of conditionals (comparisons). ***The result of a comparison is always a boolean variable***.

```
var a = 10;
var b = 20;
// a comparison - we will study this in detail in upcoming lesson
if (a>b) // The outcome of a>b will be a boolean
    console.log("Variable `a` has higher value"); // if a>b is true
else
    console.log("Variable `b` has higher value"); // if a>b is false
```

Null, Undefined and NAN

- **null** refers to the “value of nothing”
- **Undefined** refers to the “absence of a value”
- **NaN** stand for “Not-a-Number”
  - it's often returned indicating an error with number operations. For instance, if you wrote some code that performed a math calculation, and the calculation failed to produce a valid number, NaN might be returned.

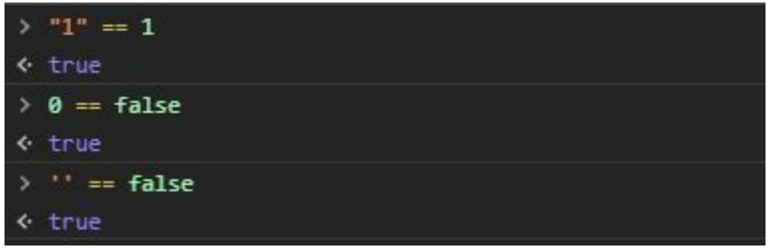


```
// calculating the square root of a negative number will return NaN
Math.sqrt(-10)

// trying to divide a string by 5 will return NaN
"hello"/5
```

Equality

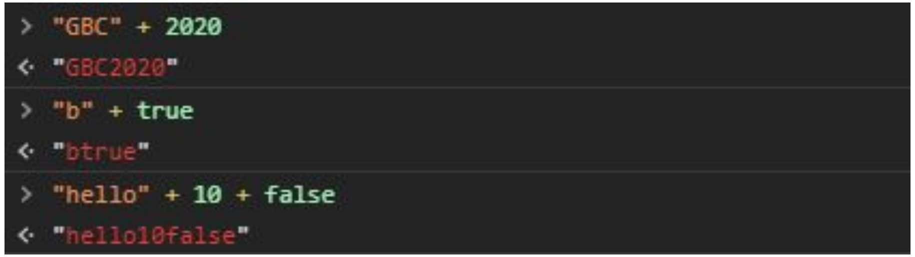
- We can use `==` and `!=` with **number equality**. Also, we can test for equal values when comparing different data-types, it can lead to some interesting results

  Type Conversion
- In the case of regular comparison, the operands on either side of the `==` operator are first converted to numbers, before comparison.
- Therefore, a `' '`, `false`, and `0` are all considered equal. Similarly, a `'1'` and `1` are also considered equal. If we don't want to convert the operands, before comparison, we have to use a **strict comparison** `===`



```
> "1" == 1
< true
> 0 == false
< true
> '' == false
< true
```

- JavaScript is known as a ***loosely typed language***.
- When you're writing JavaScript code, you do not need to specify data types. Instead, when your code is interpreted by the JavaScript engine it will automatically be converted into the "appropriate" data type.
- This is called ***implicit type coercion*** and you've already seen examples like this before when you tried to concatenate strings with numbers.
- It's behavior like this which makes JavaScript unique from other programming languages, but it can lead to some quirky behavior when doing operations and comparisons on mixed data types.



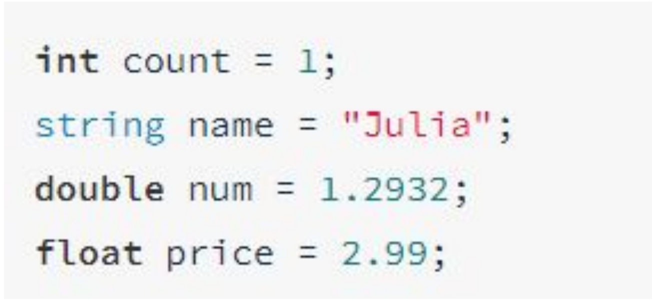
```
> "GBC" + 2020
< "GBC2020"
> "b" + true
< "btrue"
> "hello" + 10 + false
< "hello10false"
```

Implicit type conversion

Strongly Typed vs Loosely Typed

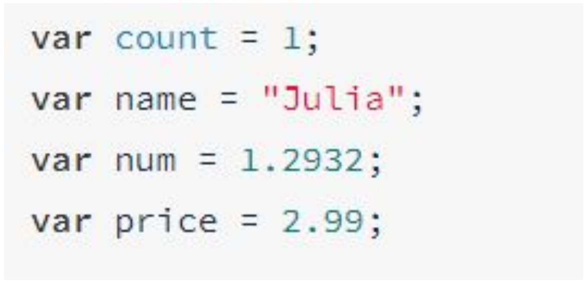
- *A **strongly typed language** is a programming language that is more likely to generate errors if data does not closely match an expected type. Because JavaScript is loosely typed, you don't need to specify data types; however, this can lead to errors that are hard to diagnose due to implicit type coercion.*

Example of Strongly Typed Language



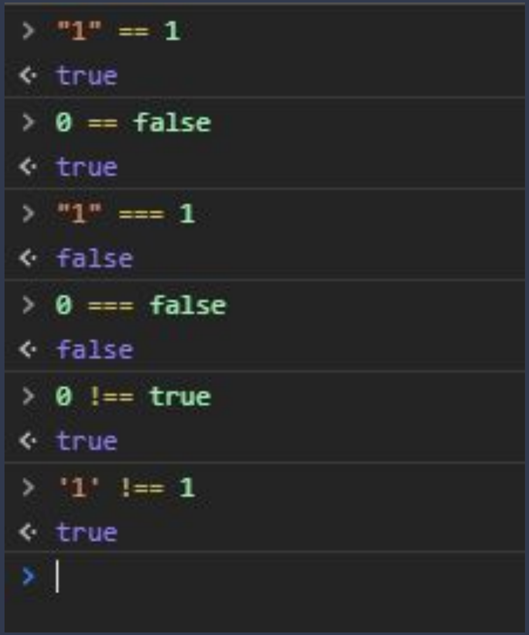
```
int count = 1;
string name = "Julia";
double num = 1.2932;
float price = 2.99;
```

JavaScript Equivalent



```
var count = 1;
var name = "Julia";
var num = 1.2932;
var price = 2.99;
```

Strict Equality



```
> "1" == 1
< true
> 0 == false
< true
> "1" === 1
< false
> 0 === false
< false
> 0 !== true
< true
> '1' !== 1
< true
> |
```

- When you use the `==` or `!=` operators, JavaScript first converts each value to the same type (if they're not already the same type); this is why it's called "type coercion"! This is often not the behavior you want, and **it's actually considered bad practice to use the `==` and `!=` operators when comparing values for equality.**

- Instead, in JavaScript it's better to use **strict equality** to see if numbers, strings, or booleans, etc. are identical in *type* and *value* without doing the type conversion first. To perform a strict comparison, simply add an additional equals sign `=` to the end of the `==` and `!=` operators.