Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.

Actually, that’s not exactly the maximum value possible. It’s actually, it’s closer to 9 quadrillion when you factor in the extra bit to denote negative or positive, and the bits needed to store the location of the decimal point.

This is called a signed number, specifically, if we’re not talking about decimals, but integers (whole numbers). Then the actual number is only:

Javascript has the following *obvious* operators :

* Addition ( + )
* Subtraction ( - )
* Multiplication ( \* )
* Division ( / )

In addition, it has a few others, specifically the **modulus**operator, which returns a remainder:

console.log(101 % 100);

Another value type in Javascript is the string value. There are a few different ways to write them (one is especially powerful, you’ll see).

"I love dumplings" -- double quotes

'I love dumplings' -- single quotes

`I love dumplings` -- backticks

**The Escape Character \**

Javascript has something called an *escape character* that tells the program you’d like to do things like start a new line, or add a tab.

* \' single quote
* \" double quote
* \\ backslash
* \n new line
* \r carriage return
* \t tab
* \b backspace
* \f form feed

**The Backtick ``**

Creating string values with backticks `hi!` come with more than a few benefits. For one, you don’t need to use the escape character to complete the challenge about. You can just write this with backticks formatted as you’d like:

**STRING INTERPOLATION + CONCATENATION**

You can also perform arithmetic on string values. well, technically not arithmetic, but you can add strings together:

*console.log('my favorite' + 'cat' + 'is' + 'concat');  
console.log('my favorite' + ' cat' + ' is' + ' con' + 'cat' );*

//my favoritecatisconcat  
//my favorite cat is concat

Strings also have methods that you can call on to perform operations on them.

***CHALLENGE #3:*** *write a program that outputs a NUMBER value for the length of the string 'abaracadabara'*

*start here: [https://developer.mozilla.org/en-US/](https://developer.mozilla.org/en-US/" \t "_blank)*

--solution below--

**SOLUTION:***console.log('abaracadabara'.length); //13*

Back-tick quotes are very powerful, we’ve already seen how they can be used to avoid having to use escape characters. They can also do something called *template literals*and *string interpolation*

What happens when you run this code?

console.log(`half of 200 is ${200/2}`);

**The typeof Operator**

The **typeof** operator returns a string indicating that tells us the type of value

console.log(typeof 45);   
//number

console.log(typeof ‘Bob Ross’)   
//string

The example above is because JS is extremely forgiving. It’s doing something called *type coercion*, which is a fancy way of saying JavaScript is trying to interpret what you really mean

This is why we say the two operators ==and !=are truthy and falsey respectively. In general, if you don’t want accidental type coercion, you should use === and !==, which will make sure to match on the TYPEOF the value as well.

#### The Ternary Operator

So far, we’ve been working with binary operators, they evaluate as true or false. The third, and very useful operator, is the TERNARY operator

console.log(true ? “Bears” : “Sharks”);   
//Bears

console.log(false ? “Bears” : “Sharks”);   
//Sharks

Syntax format: x ? y : z where **x** = what you want to evaluate, **y** = the value or statement if **true**, and **z** the value or statement if **false**.

for (statement 1; statement 2; statement 3) {  
 code block to be executed  
}

**Statement 1** is executed (one time) before the execution of the code block.

**Statement 2**defines the condition for executing the code block.

**Statement 3** is executed (every time) after the code block has been executed.