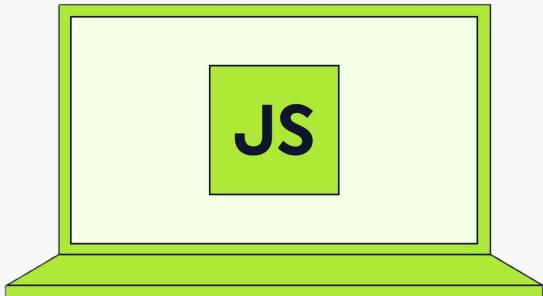




The Complete Javascript Course



@newtonschool

Lecture 2: Data Types and Operators

-Vishal Sharma

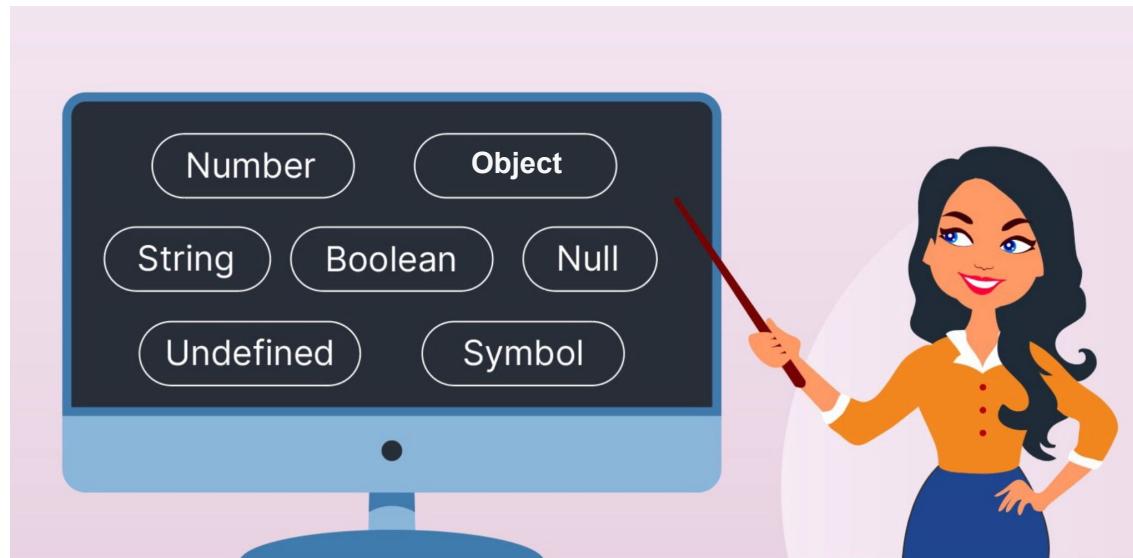


Table of Contents

- Primitive and Non-Primitive Data Types
- Types of primitive data types
- Type checking using typeof
- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Assignment Operators
- Practical Usage and Examples

What are Data Types in javascript?

In JavaScript, **data types** are the classification of data values that tell the type of data being dealt with.



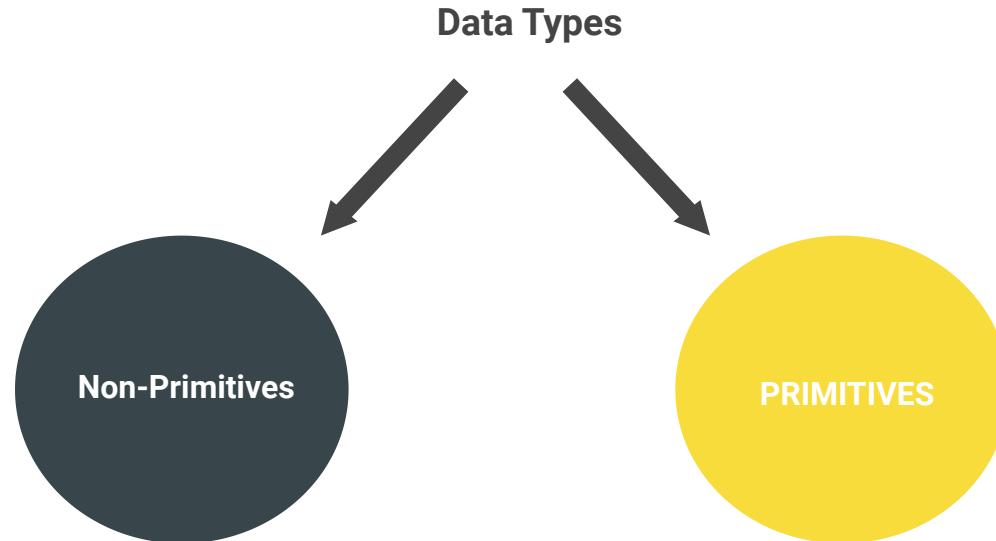
Data types indicate whether data is a number, string, boolean, etc.

Don't worry if you're unfamiliar with them yet.

Primitive and Non-Primitive Data Types

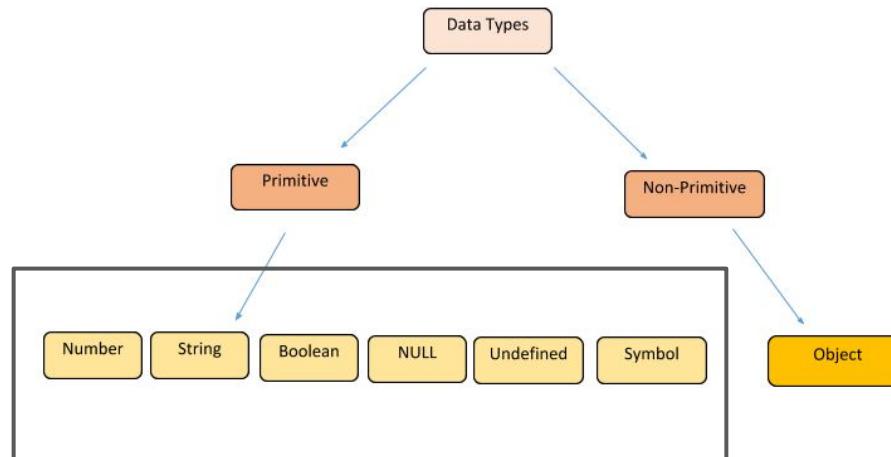
Categories of Data Types

In JavaScript, data types are primitives (simple, immutable) and non-primitives (complex, mutable, and multi-valued).



Let's start with primitives

Primitive data types in JavaScript are simple, immutable values like numbers, strings, booleans, symbols.



Declaring Primitive Data types

Primitive data types in JavaScript are simple, immutable values like numbers, strings, booleans, undefined, null etc. Let's have a look:-

```
1 // Primitive Data Types in JavaScript
2
3 // Number
4 let age = 25; // Integer
5 console.log("Age:", age); // Output: Age: 25
6
7 // String
8 let firstName = "John"; // Sequence of characters
9 console.log("First Name:", firstName);
// Output: First Name: John
10
11 // Boolean
12 let isActive = true; // Logical value
13 console.log("Is Active:", isActive);
// Output: Is Active: true
```

In JavaScript, the data type of a variable is implicitly determined by the value assigned to it, such as a **Number** for numbers, a **String** for text, and a **Boolean** for true/false values

Declaring Primitive Data types

Primitive data types in JavaScript are simple, immutable values like numbers, strings, booleans, symbols.

```
1 // Primitive Data Types in JavaScript
2
3 // Undefined
4 let userLocation;
  // Variable declared but not assigned
5 console.log("User Location:", userLocation);
  // Output: User Location: undefined
6
7 // Null
8 let car = null; // Intentional absence of a value
9 console.log("Car:", car); // Output: Car: null
```

If a variable has no value, it's automatically **undefined**. To intentionally keep it empty, assign it **null**.

Sounds overwhelming! Don't worry

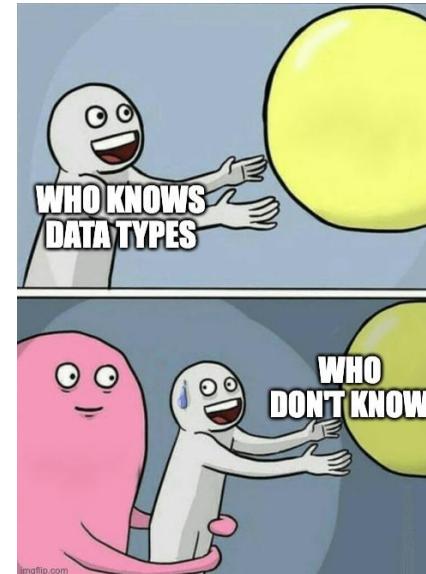
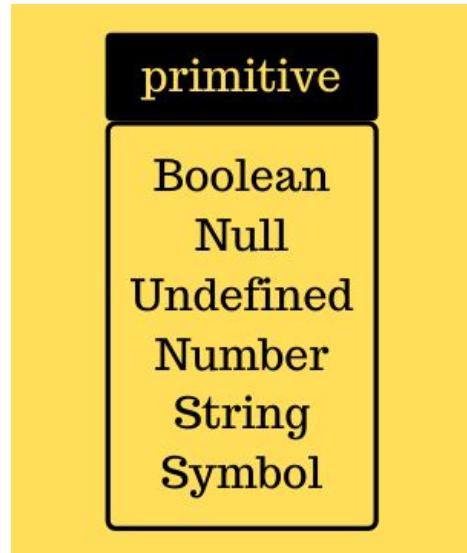
We all find it difficult when we start but it gets easier as we go along



Ask questions when you're unsure, code when things feel confusing, and remember—the more you code, the better you'll understand!

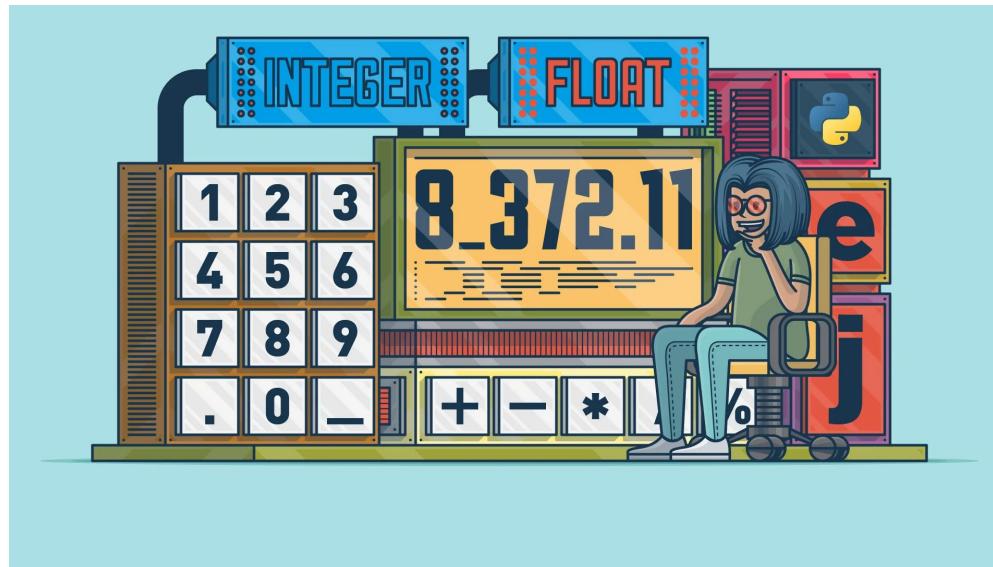
Understanding Data types

Today, we begin our journey through data types, starting with the most common and progressing to the more complex and powerful.



Meet the Numbers

They represent quantities, from counting objects to calculating complex formulas. It could be either simple integers or decimals.

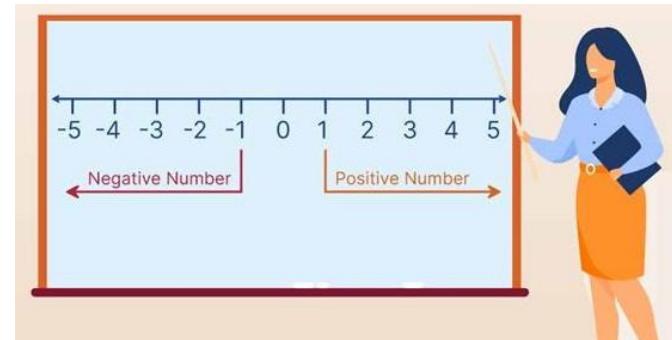


They could either
be integers or
fractions.

Declaration and Assignment

Let's declare and assign some numbers:-

```
1 let intNum = 10;      // Positive integer
2 let negInt = -5;      // Negative integer
3 let floatNum = 3.14;  // Positive float
4 let negFloat = -2.5;  // Negative float
```



Number methods

Number methods in JavaScript help manipulate numeric values efficiently. For instance, `ceil()` rounds fractional value to nearest smaller integer, while `floor()` rounds the number to nearest integer.

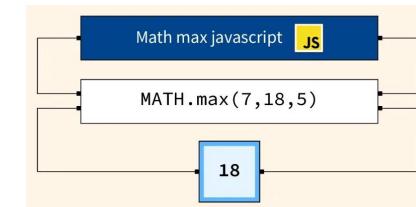
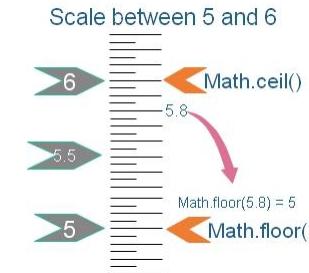


Number Methods

JavaScript provides a variety of built-in methods and operations for working with numbers. Let's start with rounding and converting:-

```

1 // Math.ceil() - Rounds a number UP to the nearest integer
2 console.log("Math.ceil(4.3) =", Math.ceil(4.3)); // 5
3
4 // Math.floor() - Rounds a number DOWN to the nearest integer
5 console.log("Math.floor(4.7) =", Math.floor(4.7)); // 4
6
7 // Math.max() - Returns the largest of the given numbers
8 console.log("Math.max(1,5,3,9,7) =", Math.max(1,5,3,9,7));
9 // 9
10
11 // Math.min() - Returns the smallest of the given numbers
12 console.log("Math.min(1,5,3,9,7) =", Math.min(1,5,3,9,7));
13 // 1
  
```



Number Methods

Let's look at few more:-



```
1 // Generate a random number between 1 and 10
2 let randomValue = Math.random();
3 // Generates a number between 0 (inclusive) and 1 (exclusive)
4
5 // Map the random value to a number between 1 and 10
6 let randomInRange = Math.floor(randomValue * 10) + 1;
7
8 console.log("Random number between 1 and 10:", randomInRange);
9 // Output: Random number between 1 and 10
```

JavaScript Math.random() Method

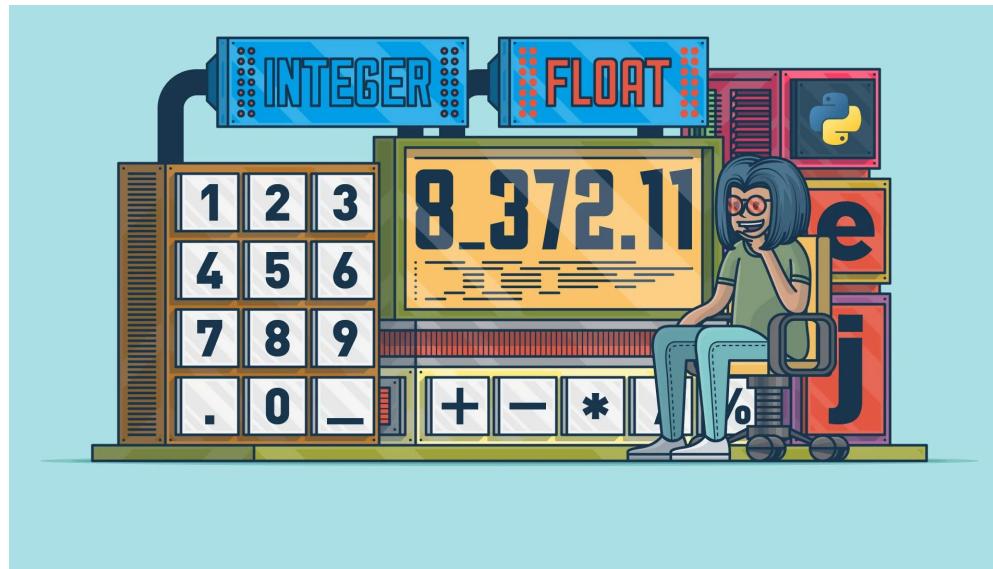


Interactive Activity

Using `Math.random()` find a number
between 2 and 8

Enter the Strings

They are the storytellers of JavaScript. They carry text, allowing us to write messages, descriptions, and even dialogue for our characters.



They could either
be integers or
fractions.

Declaration and Assignment

Let's declare and print some strings:-



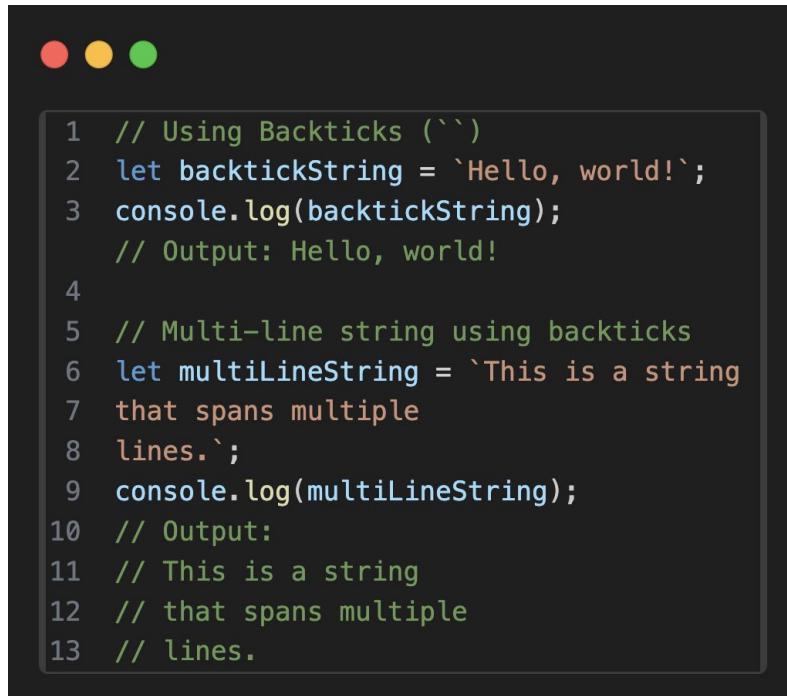
```
1 // Using Single Quotes ('')
2 let singleQuoteString = 'Hello, world!';
3 console.log(singleQuoteString);
4 // Output: Hello, world!
5
6 // Using Double Quotes ("")
7 let doubleQuoteString = "Hello, world!";
8 console.log(doubleQuoteString);
9 // Output: Hello, world!
```

Single Quotes

Double Quotes

Declaration and Assignment

Let's learn how to use backticks and how it is different from '...' and “...”



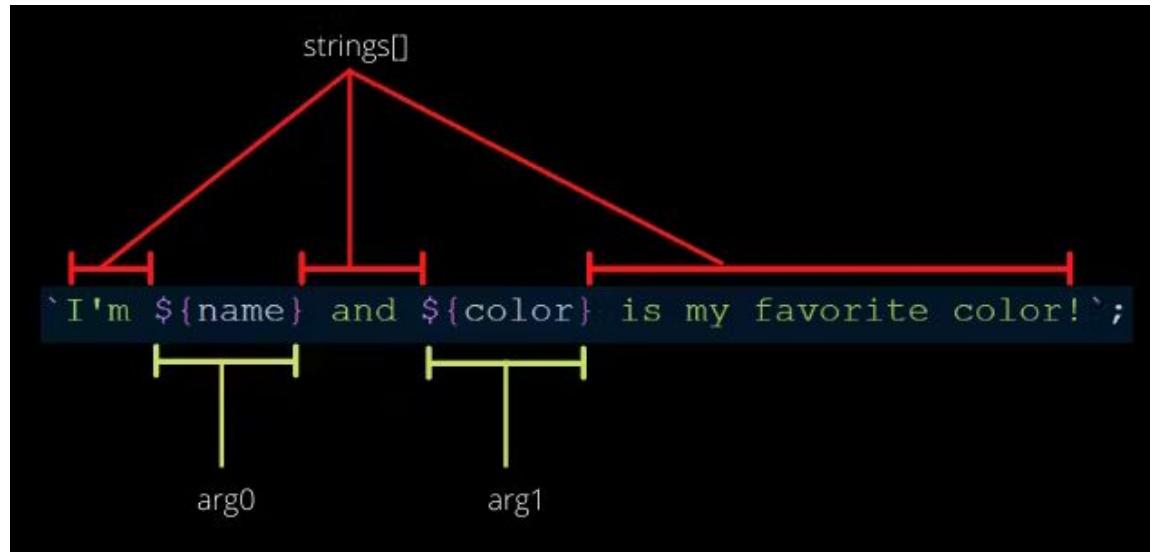
```
1 // Using Backticks (``)
2 let backtickString = `Hello, world!`;
3 console.log(backtickString);
// Output: Hello, world!
4
5 // Multi-line string using backticks
6 let multiLineString = `This is a string
7 that spans multiple
8 lines.`;
9 console.log(multiLineString);
10 // Output:
11 // This is a string
12 // that spans multiple
13 // lines.
```



Backticks
Quotes

Backticks: for template literals

Template literals use **backticks** (```) instead of single or double quotes for defining strings. With them we can embed variables inside the string. Let's see how:-



Here name and
color are
variables.

Backticks: for template literals

Template literals use **backticks** (```) instead of single or double quotes for defining strings. With them we can embed variables inside the string. Let's see how:-

```
1 let nam = 'Narendra';
2 let color = 'blue';
3
4 let str = `I'm ${nam} and ${color}
    is my favourite color`;
5
6 console.log(str);
7 // Output: I'm Narendra and blue is my favourite color!
```

With the help of template literals we embedded variables in the string.

Accessing character values in string

Each character in the string is indexed starting from 0. Consider string as row of mailboxes where each mailbox can hold a character.

For example, let's take string “**Codechef**”:

C	o	d	e	c	h	e	f
0	1	2	3	4	5	6	7

Index numbers are:



Strings are indexed

Let's try to access these character values using index.

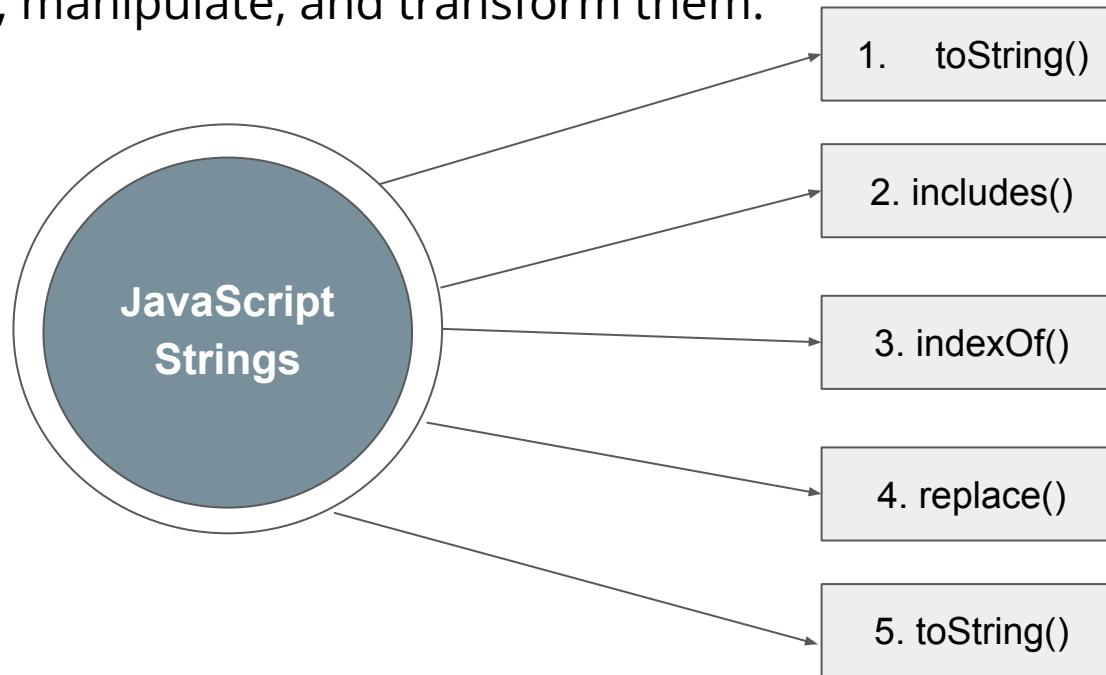
```
1 let str = "codechef";
2
3 // Accessing characters by its index (mailbox number)
4 console.log("At index 0:", str[0]); // Output: c
5 console.log("At index 1:", str[1]); // Output: o
6 console.log("At index 2:", str[2]); // Output: d
7 console.log("At index 3:", str[3]); // Output: e
8 console.log("At index 4:", str[4]); // Output: c
9 console.log("At index 5:", str[5]); // Output: h
10 console.log("At index 6:", str[6]); // Output: e
11 console.log("At index 7:", str[7]); // Output: f
```

We stored codechef in str variable and extracted each character using their respective indexes.

charAt() function achieves the same using index values.

Is that it! There is so much more!

Strings in JavaScript come with a toolbox full of powerful methods to explore, manipulate, and transform them.



String method: `toString()`

The `toString()` method in JavaScript converts a value (such as a number, array, or object) into a string representation.



```
1 let num = 123;
2 console.log(num.toString());
3 // Output: "123"
4
5 let arr = [1, 2, 3];
6 console.log(arr.toString());
7 // Output: "1,2,3"
```

Here we are converting numbers and arrays to strings. Observe how they are being converted.

Converting different data types to strings

String method: trim()

The `trim()` method in JavaScript removes whitespace from both ends of a string without affecting the spaces in between words.

```
1 let str = " Hello, world! ";
2 console.log(str.trim());
3 // Output: "Hello, world!"
```

Trimming down the white spaces

Can you observe the little white spaces and how they have gotten trimmed?

String method: replace()

The `replace()` method in JavaScript is used to replace a specified substring or pattern within a string with another substring.



```
1 let str = "Hello, world!";
2 let newStr = str.replace("world", "JavaScript");
3 console.log(newStr);
4 // Output: "Hello, JavaScript!"
```

We have successfully replaced the word. Repeat it in your code editor.

String method: `toString()`

The `toString()` method in JavaScript converts a value (such as a number, array, or object) into a string representation.

```
1 let num = 123;
2 console.log(num.toString());
3 // Output: "123"
4
5 let arr = [1, 2, 3];
6 console.log(arr.toString());
7 // Output: "1,2,3"
```

Can you observe the little white spaces and how they have gotten trimmed?

String method: includes()

The `includes()` method in JavaScript checks if a string contains a specified substring. It returns `true` if the substring is found, otherwise `false`.

```
1 let sentence = 'JavaScript is awesome!';
2 console.log(sentence.includes('is'));
// true
3 console.log(sentence.includes('awesome'));
// true
4 console.log(sentence.includes('Python'));
// false
5 console.log(sentence.includes('Java', 5));
6 // false (search starts from index 5)
```

It is checking if string includes the word and giving out boolean. Observer the last one, why is it false??

String method: indexOf()

The `indexOf()` method in JavaScript returns the index of the first occurrence of a substring in a string or `-1` if not found. It is case-sensitive.

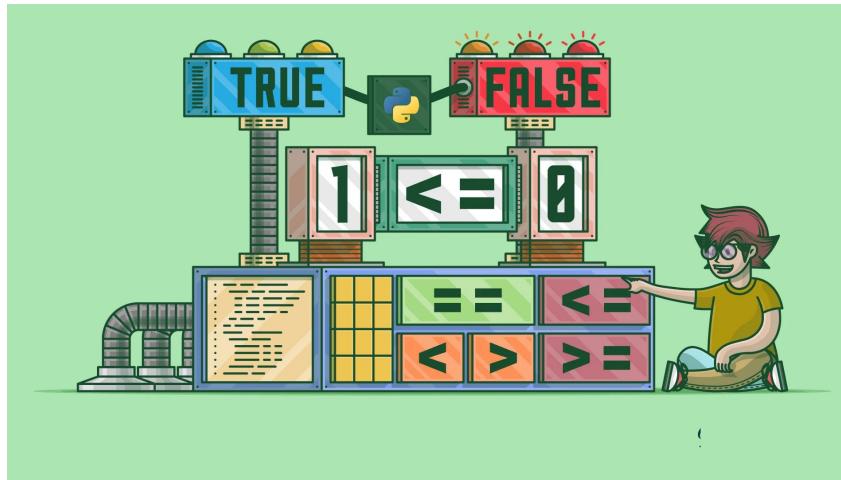
```
1 let sentence = 'JavaScript is awesome!';
2
3 console.log(sentence.indexOf('is'));
// 11 (index where 'is' starts)
4 console.log(sentence.indexOf('awesome'));
// 15 (index where 'awesome' starts)
5 console.log(sentence.indexOf('Python'));
// -1 (not found)
6 console.log(sentence.indexOf('Java', 5));
// -1 (search starts from index 5)
```

Last one returns `-1` since Java word is not available after 5th index.

Unraveling the Basics: boolean, null, and undefined

boolean

Think of it as a switch—only two states: true or false.



```
1 let isSunny = true;
2 let isRaining = false;
```

Assigning boolean values

null

This represents "nothing" or "empty". You can assign it intentionally when a variable is meant to have no value.



```
1 let car = null;  
2 // This variable is explicitly set to have no value  
3  
4 console.log(car);  
5 // Output: null
```

undefined

In JavaScript, **undefined** means a variable has been declared but has not been assigned a value yet. It also represents the default value of uninitialized variables.



undefined



defined

```
1 // Undefined example
2 let userAge;
3 console.log("userAge:", userAge);
4 // Output: undefined
5 // (variable is declared but not assigned a value)
6
7 // Defined example
8 let userName = "John Doe";
9 console.log("userName:", userName);
10 // Output: "John Doe"
11 // (variable is assigned a string value)
```

undefined vs defined

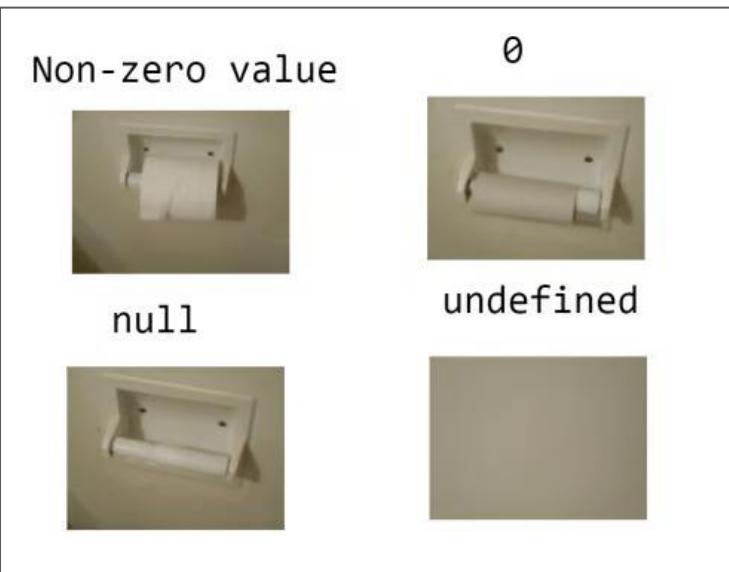
Any difference in null & undefined??

Even though they look similar but they are not same at all. You have fallen in same trap like everyone else:-



Understanding differences

Let's understand difference between zero, null and undefined



undefined	null
has not been assigned	could be assigned
typeof undefined	typeof object

Truthy and Falsy values

Any value which is not falsy is considered true.

Falsy Expressions
False
NaN
Undefined
Null
Empty String (“”/ ”)
0

Apart from these values all the values are considered true.

How to check truthy/falsy values??

We can check truthy/falsy values using Boolean function.

```
1 let value = "Hello";
2 // Example truthy value
3
4 console.log(Boolean(value));
5 // Output: true
6
7 let anotherValue = "";
8 // Example falsy value
9 console.log(Boolean(anotherValue));
10 // Output: false
```

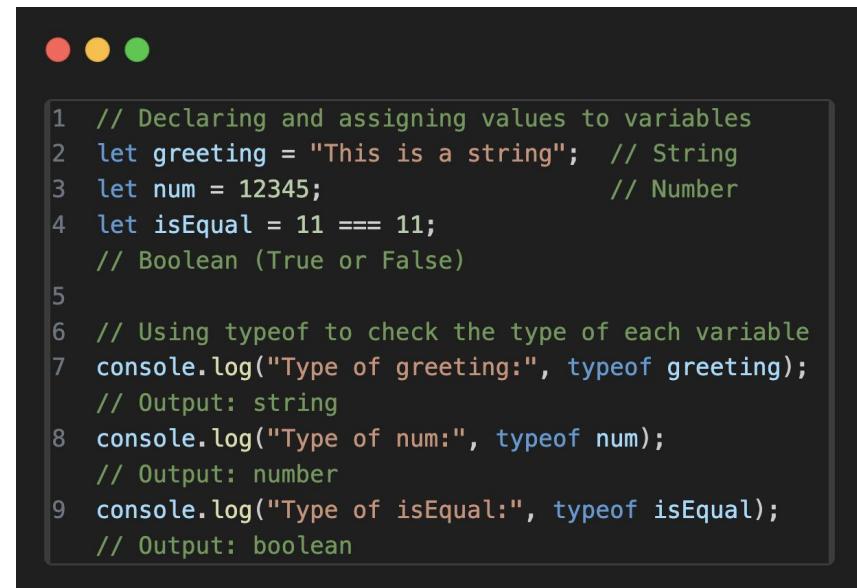
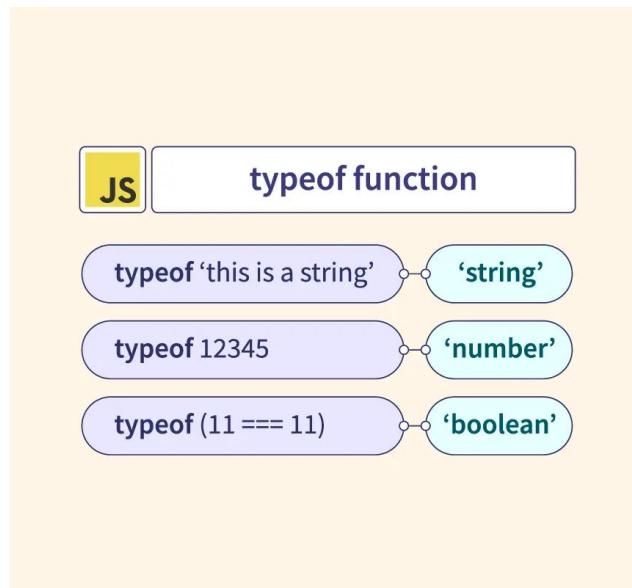
Here we used:-

1. "Hello": Evaluated as true
2. "": Evaluated as false

Try few other examples by your own.

How to check data type: `typeof`

The `typeof` operator is used to determine the type of a given variable or value. It returns a string indicating the type



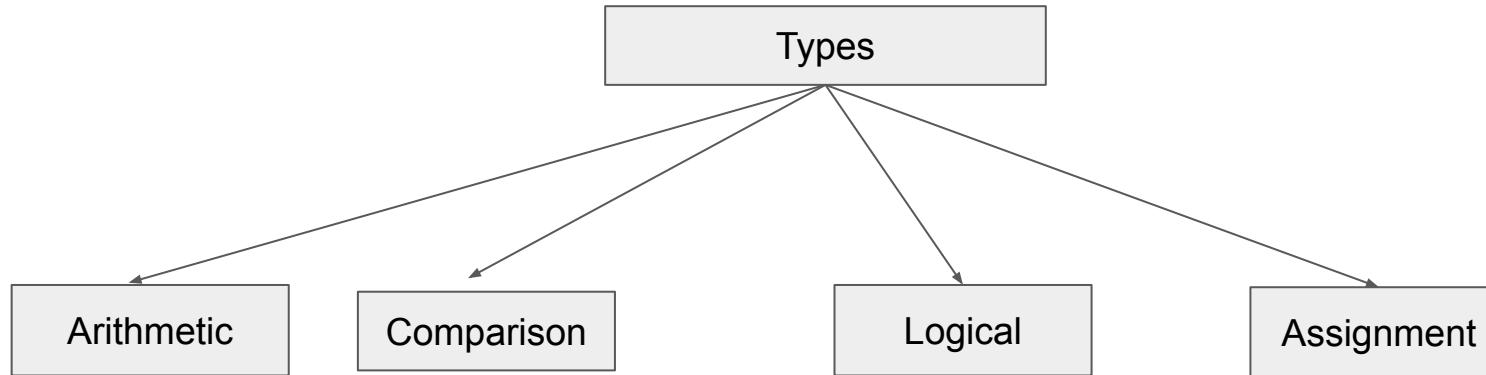
```
● ● ●

1 // Declaring and assigning values to variables
2 let greeting = "This is a string"; // String
3 let num = 12345; // Number
4 let isEqual = 11 === 11; // Boolean (True or False)
5
6 // Using typeof to check the type of each variable
7 console.log("Type of greeting:", typeof greeting);
// Output: string
8 console.log("Type of num:", typeof num);
// Output: number
9 console.log("Type of isEqual:", typeof isEqual);
// Output: boolean
```

Operators

What are operators?

In JavaScript, **operators** are special symbols used to perform operations on values



Arithmetic Operators

Arithmetic operators in JavaScript are used to perform mathematical operations on numbers.

Arithmetic Operator	Name	Example
+	Addition	$a + b$
-	Subtraction	$a - b$
*	Multiplication	$a * b$
/	Division	a / b
%	Modulus	$a \% b$
++	Increment Operator	$a++$
--	Decrement Operator	$a--$

Arithmetic Operators: Example

Here's are few examples of arithmetic operations in JavaScript:

```
● ● ●

1 // Declaring two variables
2 let num1 = 10; // First number
3 let num2 = 5; // Second number
4
5 // Addition
6 let addition = num1 + num2;
7 console.log("Addition: num1 + num2 =", addition);
8 // Output: 15
9
10 // Subtraction
11 let subtraction = num1 - num2;
12 console.log("Subtraction: num1 - num2 =", subtraction);
13 // Output: 5
14
15 // Multiplication
16 let multiplication = num1 * num2;
17 console.log("Multiplication: num1 * num2 =", multiplication);
18 // Output: 50
```

When you see your younger sibling
struggling with basic algebra



Unexpected results we get sometimes

There are few anomalies which we face due to impermissible operations. Let's understand with code:-

```
1 // 1. Division by zero = Infinity
2 console.log("1 / 0 =", 1 / 0);
3 // Output: Infinity
4
5 // 2. Number + String = string concatenation
6 console.log("1 + '2' =", 1 + '2');
7 // Output: '12'
8
9 // 3. String - String = numeric subtraction
10 console.log("'2' - '2' =", '2' - '2');
11 // Output: 0
```

Here you can observe that not all mathematical operations go as expected.

Data type of infinity

In previous slide you can observe that divide by zero provides infinity, but if infinity is a number. Want to check? Use isNaN().

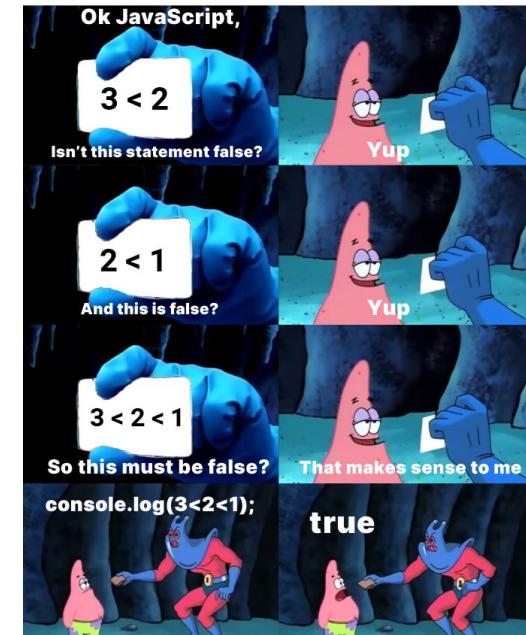
```
1 // Checking if value is a number using isNaN
2 let value1 = Infinity;
3 let value2 = -Infinity;
4 let value3 = 42;
5 let value4 = "Hello";
6
7 // Using isNaN() to check if it's a valid number
8 console.log(isNaN(value1));
// false (Infinity is a number)
9 console.log(isNaN(value2));
// false (-Infinity is a number)
10 console.log(isNaN(value3));
// false (42 is a number)
11 console.log(isNaN(value4));
// true (string is not a number)
```

You can observe infinity
is a number.

Comparison Operators

Comparison operators in JavaScript are used to compare two values and return a Boolean value (**true** or **false**), helping to evaluate relationships between

>	Greater Than
>=	Greater Than or Equal To
<	Less Than
<=	Less Than or Equal To
==	Equal To
===	Strict Equal To
!=	Not Equal To
!==	Strict Not Equal To



Comparison Operators: example

Here's a simple code example to demonstrate the use of comparison operators in JavaScript:

```
1 // Declaring two variables
2 let x = 10; // First variable
3 let y = 5; // Second variable
4
5 // Equality comparison (==)
6 console.log("x == y:", x == y);
// Output: false (x is not equal to y)
7
8 // Strict equality comparison ( === )
9 console.log("x === y:", x === y);
// Output: false (x is not strictly equal to y)
10
11 // Inequality comparison (!=)
12 console.log("x != y:", x != y);
// Output: true (x is not equal to y)
```

Pay caution while choosing '`==`' over '`===`' as they have different outcomes sometimes.

Comparison Operators: example

Continued...

```
1 // Strict inequality comparison (!==)
2 console.log("x !== y:", x !== y);
  // Output: true (x is not strictly equal to y)
3
4 // Greater than comparison (>)
5 console.log("x > y:", x > y);
  // Output: true (x is greater than y)
6
7 // Less than comparison (<)
8 console.log("x < y:", x < y);
  // Output: false (x is not less than y)
9
10 // Greater than or equal to (>=)
11 console.log("x >= y:", x >= y);
  // Output: true (x is greater than or equal to y)
12
13 // Less than or equal to (<=)
14 console.log("x <= y:", x <= y);
  // Output: false (x is not less than or equal to y)
15
```

The correct placement of `<` or `>` before `=` is important for clarity.

Incorrect:- `=<`
Correct: `<=`

Logical Operators

Logical operators in JavaScript are used to perform logical operations on values, typically boolean values (`true` or `false`).

AND, OR, and NOT

Operator	Name	Example	Result
<code>&&</code>	AND	<code>x < 10 && x !== 5</code>	<code>false</code>
<code> </code>	OR	<code>y > 9 x === 5</code>	<code>true</code>
<code>!</code>	NOT	<code>!(x === y)</code>	<code>true</code>

Now how are you feeling??

You must be having love-hate relationship now. Don't worry, you will feel at home more you practice:-



Some Real World Examples

Calculating Discounts and Special Offers

When shopping, it's crucial to calculate the best price after discounts. For example, if an item costs \$100 and has a 20% discount.

```
1 // Original price of the item
2 let originalPrice = 100;
3
4 // Discount percentage
5 let discountPercentage = 20;
6
7 // Calculate discount amount
8 let discountAmount = (originalPrice * discountPercentage) /
100;
9
10 // Calculate final price after discount
11 let finalPrice = originalPrice - discountAmount;
12
13 console.log("Original Price: $", originalPrice);
// Output: 100
14 console.log("Discount Amount: $", discountAmount);
// Output: 20
15 console.log("Final Price After Discount: $", finalPrice);
// Output: 80
```

Using arithmetic operators to determine the discount amount and the final price.

Job Offers: Making Right Choice

To choose the better job offer, you can use comparison operators to evaluate salaries.

```
1 // Salary offers from two job companies
2 let offerA = 60000; // Job offer A salary
3 let offerB = 75000; // Job offer B salary
4
5 // Compare salaries using comparison operators
6 let isOfferAGreater = offerA > offerB;
7 // Checks if offer A is greater than offer B
8 let isOfferBGreater = offerB > offerA;
9 // Checks if offer B is greater than offer A
10
11 // Display the best offer
12 if (offerA > offerB) {
13   console.log("Choose Offer A for a higher salary.");
14 } else if (offerB > offerA) {
15   console.log("Choose Offer B for a higher salary.");
16 } else {
17   console.log("Both offers are the same.");
18 }
```

Using a greater-than and smaller than to help you quickly see which offer provides a higher salary.

Practice! Practice! And become boss

More you practice more you find easy way outs and more you will become confident in writing javascript code. Practice is the key...



In Class Questions

**Thanks
for
watching!**