JAVASCRIPT TYPECASTING

Typecasting is changing a variable from one datatype to another datatype. For example, you might want to change a number to a string, a string to a number, a number to a Boolean, ect.

1. converting a string to a number:

This can involve changing to an integer or changing to a floating point value.

In JavaScript, you can convert a string to a number using several methods, with the most common ones being the Number() function, the unary plus (+) operator, parseInt(), and parseFloat().

Method	Description	Example	Result
Number()	Converts the entire string to a number. Returns NaN if any part is non-numeric (except for whitespace).	Number("123.45")	`123.45`
Unary +	A concise operator that performs the same conversion as Number().	+"123.45"	`123.45`
parseInt()	Parses a string and returns an integer. It stops parsing when it encounters the first non-numeric character and ignores characters that follow.	parseInt("123.45xyz")	`123`
parseFloat()	Parses a string and returns a floating-point number. Similar to parseInt(), it stops at the first invalid character.	parseFloat("123.45xyz")	`123.45`

Key Differences and Usage:

- **Strictness**: Number() and the unary + operator are stricter. They return NaN (Not a Number) if the string contains non-numeric characters (e.g., "100px" becomes NaN).
- **Partial Parsing**: parseInt() and parseFloat() are more lenient; they will parse the initial numeric portion of a string and ignore trailing non-numeric characters (e.g., parseInt("100px") becomes 100).
- **Integers vs. Floats**: Use parseInt() for whole numbers and parseFloat() or Number()/+ for numbers with decimals.
- **Radix (Base)**: When using parseInt(), it is best practice to always specify the radix (base), typically 10 for decimal numbers, to avoid potential unexpected behavior in very old browsers.

2. changing a number to an string in Javascript:

In JavaScript, you can convert a number to a string using several methods, with the most common ones being the

toString() method, the global String() function, and string concatenation.

Here are the primary methods:

1. Using the toString() method:
This is a standard and efficient method belonging to the Number.prototype object. It works directly on the number value.
javascript

```
let num = 255;
let str = num.toString();
console.log(str);          // Output: "255"
console.log(typeof str);   // Output: "string"

// You can also use it directly on number literals
console.log((123).toString()); // Output: "123"
```

Note: toString() can also accept an optional radix argument (base) to convert the number to a different base representation, such as binary (base 2) or hexadecimal (base 16).

```
let hex = num.toString(16); // "ff"
let bin = num.toString(2);  // "11111111"
```

Caution: This method will throw a TypeError if you try to call it on null or undefined values.

2. Using the global String() function:
The global String() function is safer when you are unsure of the input value's type, as it gracefully handles null and undefined.

```
let num = 123;
let str = String(num);
console.log(str);          // Output: "123"
console.log(typeof str);   // Output: "string"

console.log(String(null));      // Output: "null"
console.log(String(undefined)); // Output: "undefined"
```

3. Using String Concatenation (The empty string trick):
Concatenating an empty string ("" or '') with a number uses JavaScript's automatic type coercion to convert the number to a string.

```
let count = 7;
let text = count + '';
console.log(text);          // Output: "7"
console.log(typeof text);   // Output: "string"
```

4. Using Template Literals:
This modern ES6 feature allows you to embed expressions within a string using backticks (`).

```
let score = 42;
let message = `Your score is ${score}`;
console.log(message);      // Output: "Your score is 42"
console.log(typeof message); // Output: "string"
```

Summary of Differences:

Method	Best For	Handles null/undefined?
toString()	Direct, efficient number conversion	No, throws error
String()	Safe conversion when input type is uncertain	Yes
Concatenation	Quick, simple coercion	Yes (results in "null", "undefined")
Template Literal	Embedding numbers within larger strings	

3. changing from a number to a Boolean datatype:

In JavaScript, you can convert a number to a boolean using either the Boolean() function or the double NOT (!!) operator. The number **0** converts to false, while all non-zero numbers (including negative numbers and decimals) convert to true.

Here are the two primary ways to explicitly convert a number to a boolean:

1. Using the Boolean() Function:

The built-in global Boolean() function takes any value and converts it to its boolean equivalent. This is generally considered more explicit and readable.
javascript

```
Boolean(0);       // false
Boolean(1);       // true
Boolean(42);      // true
Boolean(-10);     // true
Boolean(3.14);    // true
Boolean(NaN);     // false
```

2. Using the Double NOT (!!) Operator:

The double NOT operator (!!) is a concise way to achieve the same result using logical negation.

- The first ! operator converts the value to a boolean and then negates it (e.g., !1 becomes false).
- The second ! negates that result again, returning the true boolean value of the original number (e.g., !false becomes true).

```
!!0;       // false
!!1;       // true
!!42;      // true
!!-10;     // true
!!3.14;    // true
!!NaN;     // false
```

Falsy and Truthy Values:
In JavaScript, all values have an inherent "truthiness" or "falsiness". The following values are considered "falsy" and will result in false when typecast to a boolean:

- 0
- NaN (Not a Number)
- null
- undefined
- "" (an empty string)
- false

All other numbers, including Infinity, are "truthy" and convert to true.

4. Boolean to number typecasting in Javascript:

In JavaScript, you can convert a boolean to a number using several methods, with true becoming 1 and false becoming 0.

The most common methods involve explicit type conversion using the Number() function or implicit conversion using arithmetic operators.

Here are the primary methods:

Explicit Conversion:
The most readable method is using the built-in Number() function.
javascript

```
let myFalseBool = false;
let myFalseInt = Number(myFalseBool); // myFalseInt is 0

let myTrueBool = true;
let myTrueInt = Number(myTrueBool);   // myTrueInt is 1
```

Implicit Conversion:
JavaScript automatically converts booleans to numbers when they are used in certain arithmetic operations.

- **Unary Plus Operator (+)**: This operator converts its operand into a number.

```
let myFalseInt = +false; // myFalseInt is 0
let myTrueInt = +true;   // myTrueInt is 1
```

- **Arithmetic Operations**: Performing any arithmetic operation like multiplication by 1 will coerce the boolean into a number.

```
let myFalseInt = false * 1; // myFalseInt is 0
let myTrueInt = true * 1;   // myTrueInt is 1

// Other operations also work:
let result = 5 - true;  // result is 4 (true is converted to 1)
```

- **Bitwise Operators**: The bitwise OR operator (|) is sometimes used as a fast way to force an integer value.

```
let myFalseInt = false | 0; // myFalseInt is 0
let myTrueInt = true | 0;   // myTrueInt is 1
```

Conditional (Ternary) Operator:
For situations where performance is critical, some benchmarks suggest the ternary operator might be the fastest method:

```
let boolValue = true;
let numberValue = boolValue ? 1 : 0; // numberValue is 1
```