JavaScript

String, Coercion, Objects

Today

- Strings
- Null
- Undefined
- Strings and Numbers
- Conditional
- Coercian

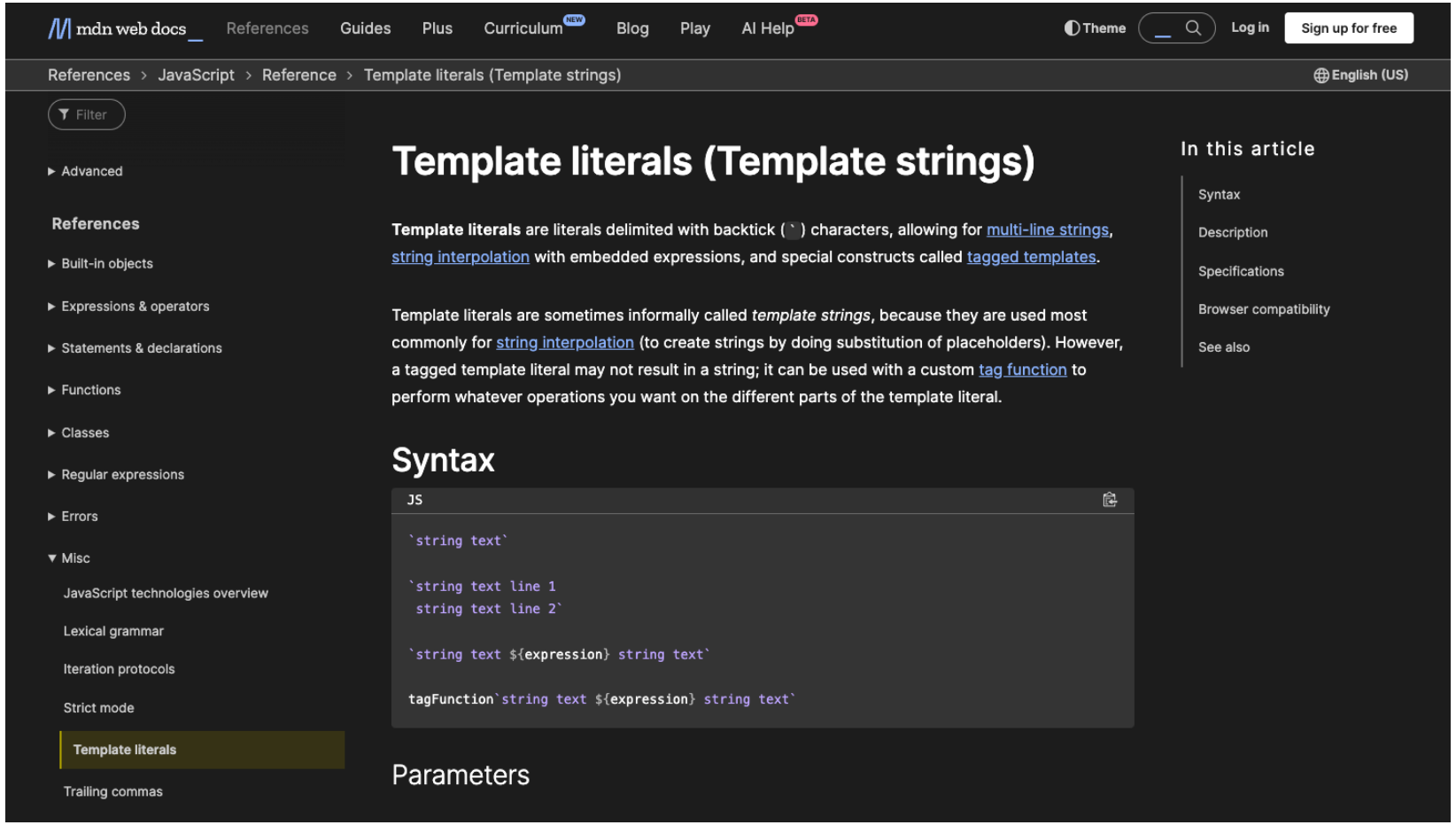
String Concatenation and Template Literals

```
const firstName = 'John';
const lastName = 'Doe';

const fullName = `${firstName} ${lastName}`;// John Doe – Template Literal
const fullName2 = firstName + ' ' + lastName;// John Doe – Concatenation


const a = 5;
const b = 10;
"Fifteen is " + (a + b) + " and not " + (2 * a + b) + "."// Fifteen is 15 and not 20.

const multiline = `This is a
multiline
string with backticks
we can use variables like ${a} and ${b} in it
and expressions like ${a + b} too`;
```

```
This is a
multiline
string with backticks
we can use variables like 5 and 10 in it
and expressions like 15 too
```

Docs

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals



Undefined

```
// Any variable that is created in JavaScript
// that is not assigned a value is undefined:
const  noValue;  // The value here will be undefined

//You can also explicitly set a variable to undefined:
const  favoriteFood = "Candy";

// Changed your mind
const  favoriteFood = undefined;
```

Null

```
// Null is not the same as undefined.
It signifies an intentional absense of data.
const  secondEmailAddress = null;
```

- It is important to remember that null and undefined are different types in JavaScript
- This can be a confusing feature of JavaScript, even for people who know other programming languages.
- The distinction can seem somewhat arbitrary when you're first learning the language, but as you get more comfortable the distinction will become clearer.

Figuring out a variable's type

- In JavaScript, we have a keyword called typeof that returns the type of the variable.

```
typeof "";        // - "string"
typeof 5;         // - "number"
typeof false;     // - "boolean"
typeof undefined; // - "undefined"
typeof null;      // this is not what we expect,
                  // it returns "object"!
```

Converting to a string: toString

- The toString method will convert any value which is not undefined or null into a string

```
const  num = 5;
const  bool = true;

num.toString(); // "5";
bool.toString(); // "true";
```

Converting to a number using *parse*

- There are several ways you can convert a value to a number.
- One way is to parse the number, using parseInt or parseFloat:
- Each function will look at a string from left to write and try to make sense of the characters it sees as numbers.

```
parseInt("2"); // 2
parseFloat("2"); // 2
parseInt("3.14"); // 3
parseFloat("3.14"); // 3.14
parseInt("2.3alkweflakwe"); // 2
parseFloat("2.3alkweflakwe"); // 2.3
parseInt("w2.3alkweflakwe"); // NaN (not a number)
parseFloat("w2.3alkweflakwe"); // NaN (not a number)
```

Converting to a number using *Number*

- This doesn't parse, it simply tries to convert the entire string directly to a number

```
Number("2"); // 2
Number("3.14"); // 3.14
Number("2.3alkweflakwe"); // NaN
Number("w2.3alkweflakwe"); // NaN
```

Converting to a number using +

- This doesn't parse, it simply tries to convert the entire string directly to a number.

```
+"2"; // 2
+"3.14"; // 3.14
+"2.3alkweflakwe"; // NaN
+"w2.3alkweflakwe"; // NaN
```

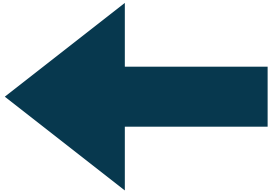
Boolean Logic

- Write conditional logic using boolean operators
- List all of the falsey values in JavaScript
- Use if/else and switch statements to include conditional logic in your JavaScript code
- Explain the difference between == and === in JavaScript
- Convert between data types explicitly in JavaScript

Conditional Logic

- An essential part of writing programs is being able to execute code that depends on certain conditions. For example:
  - You want the navigation bar on your website to look different based on whether or not someone is logged in
  - If someone enters their password incorrectly, you want to let them know; otherwise, you want to log them in
  - You're building a tic-tac-toe game, and want to know whether it's X's turn or O's turn
  - You're building a social network and want to keep person A from seeing person B's profile unless the two of them are friends

```
const  instructor = 'Brenda';

// we begin with an "if" statement
// followed by a condition in ()
// and a block of code inside of {}
if (instructor === 'Brenda') {
  console.log('Yes!');
} else {
  console.log('No');
}
```



Always true

- Notice that we used a === instead of =.
- Anytime that we use more than one equals operator (we can either use == or ===) we are doing a comparison (comparing values).
- When we use a single equals operator =, we are doing an assignment (setting a variable equal to some value).

```
const  favoriteFood = prompt('What\'s your favorite food?');

if (favoriteFood === 'pizza') {
  console.log('Woah! My favorite food is pizza too!');
} else {
  console.log('That\'s cool. My favorite food is pizza.');
}
```

- In this version, the boolean expression will be true/false depending on the value entered in ‘prompt’

Difference between “==“ and “===“

- Two different operators for comparison: the double and triple equals.
- Both operators check whether the two things being compared have the same value, but there's one important difference.
  - == allows for ***type coercion*** of the values,
  - === does not.
- To understand the difference between these operators, we first need to understand what is meant by ***type coercion***.

Type Coercion 1

- Add a number and a string.
- In a lot of programming languages, this would throw an error, but JavaScript is more accommodating
- It evaluates the expression "hi" by first coercing 5 into a string, and then interpreting the "+" operator as string concatenation.
- So it combines the string "5" with the string "hi" into the string "5hi"

```
5 + 'hi'; // '5hi'
```

Type Coercion 2

- JavaScript expects the values inside of parentheses that come after the keyword if to be booleans.

- If you pass in a value which is not a boolean, JavaScript will coerce the value to a boolean according to the rules for ***truthy/falsey*** values (more on this later)

```
if ('foo') {
  console.log('this will show up!');
}

if (null) {
  console.log('this won\'t show up!');
}
```

Type Coercion 3

- A very common way to coerce a stringified number back into a number.
- By prefacing the string with the plus sign, JavaScript will perform a coercion on the value and convert it from a string value to a number value.

```
+'304'; // 304
```

Coercion

- In REAL LIFE: Do not rely on coercion
- We typically only need to parse a number from a string on user input
- parseInt or parseFloat or actual data types as soon as possible in your program (typically as soon as input is read) is the best way to think about it
- Do not pass around numbers as strings more than you have to and remember, never use ==

“==“ Vs “===“ again

== → loose

```
5 == '5'; // true
'true' == true; // false
true == 1; // true
undefined == null; // true
```

=== → strict

```
5 === '5'; // false
'true' === true; // false
true === 1; // false
undefined === null; // false
```

- == allows for coercion while === doesn't.
- *If you don't want to have to think about coercion in your comparisons, stick to ===.*
- *IN REAL LIFE: DO NOT USE ==, forget about it*

```
const  x = 4;
if (x <= 5) {
  console.log('x is less than or equal to five!');
} else {
  console.log('x is not less than or equal to five!');
}
```

Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Falsey Values

- Some values (aside from false) are actually false as well, when they're used in a context where JavaScript expects a boolean value

- Even if they do not have a "value" of false, these values will be translated (or "coerced") to false when evaluated in a boolean expression.

6 Falsey Values in Javascript

```
0
""
null
undefined
false
NaN // (short for not a number)
```

Logical Operators

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x == 5 || y == 5) is false
!	not	!(x == y) is true

If-Else

- Sometimes you may have more than two conditions to check.
- In this case, you can chain together multiple conditions using else

```
if (number >= 1000) {
  console.log('Woah, thats a big number!');
} else if (number >= 0) {
  console.log('Thats a cool number.');
} else {
  console.log('Negative numbers?! Thats just bananas.');
}
```

Switch

- Another way to write conditional logic is to use a switch statement.
- While these are used less frequently, they can be quite useful when there are multiple conditions that can be met.
- Notice that each case clause needs to end with a break so that we exit the switch statement.

```
switch (feeling) {
  case 'happy':
    console.log("Awesome, Im feeling happy too!);
    break;
  case 'sa':
    console.log('Thats too bad, I hope you feel better soon.');
    break;
  case 'hungry':
    console.log('Me too, lets go eat some pizza!');
    break;
  default:
    console.log('I see. Thanks for sharing!');
}
```

Modulus Operator

```
5 % 3 === 2 // true (the remainder when five is divided by 3 is 2)

const  num = prompt('Please enter a whole number');
if ( num % 2 === 0 ) {
  console.log('the num variable is even!')
} else if ( num % 2 === 1) {
  console.log('the num variable is odd!')
} else {
  console.log('Hey! I asked for a whole number!');
}
```

Object Data Types

- Whereas primitive data typed variables hold individual values. e.g:
  - numbers
  - strings
  - boolean etc…
- Object types can hold *more than one value. e.g.:*
  - a number AND a string.
  - 2 numbers and a boolean and a string
  - 3 strings and 2 numbers
- Objects are central to creating interesting and powerful programs

Creating an Object

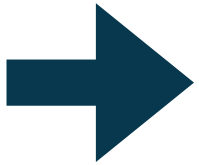
- Introduces singe variable called ‘homer’.
- This is an object with two fields
  - firstName, containing ‘homer’
  - lastName, containing ‘simpson’

```
const homer = {
  firstName: 'homer',
  lastName: 'simpson',
};
```

Objects with Strings & Numbers

```
const bart = {
   firstName: 'bart',
   lastName: 'simpson',
   age: 10,
};

console.log(bart);
```

- An object containing 2 strings and a number.



{ firstName: 'homer', lastName: 'simpson' }

Anatomy of an Object

name of the object

```
const homer = {
  firstName: 'homer',
  lastName: 'simpson',
  age: 50,
};
```

attributes (fields) of the object

attribute (field) values for the homer object

a specific attribute - called ‘age’

Objects in the Console

- We can paste code directly in the console for experimentation purposes
- Can be useful when learning or to clarify your understanding about some syntax/feature

Objects with Functions

```
const marge = {
   firstName: 'marge',
   lastName: 'simpson',
   age: 10,
   sayHello() {
      console.log('Hello from me!');
   },
};

marge.sayHello();
```

name of the object

data attributes (fields) of the object

attribute values for the object

a function attribute of the object

```
const marge = {
  firstName: 'marge',
  lastName: 'simpson',
  age: 45,
  sayHello() {
    console.log('Hello from me!');
  },
};

console.log(marge);
console.log(marge.firstName);
console.log(marge.age);

marge.sayHello();
```

accessing marge's fields

calling the function within the marge object.

this refers to the 'current' object. Ned in this case

```
const ned = {
  firstName: 'ned',
  lastName: 'flanders',
  age: 45,
  speak() {
    console.log('How diddley do? says ' + this.firstName);
  },
};

ned.speak();
```

How diddley do? says ned