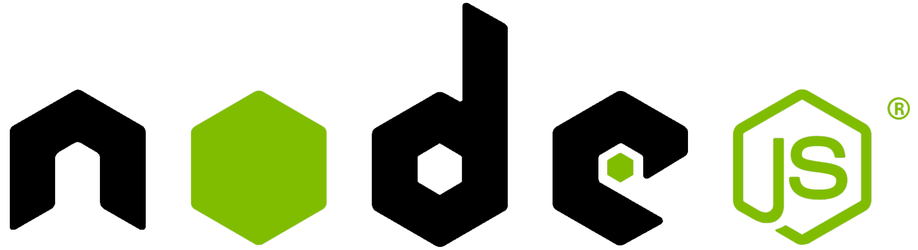
JAVASCRIPT

DATA TYPES , VARIABLE, FUNCTION AND SCOPE

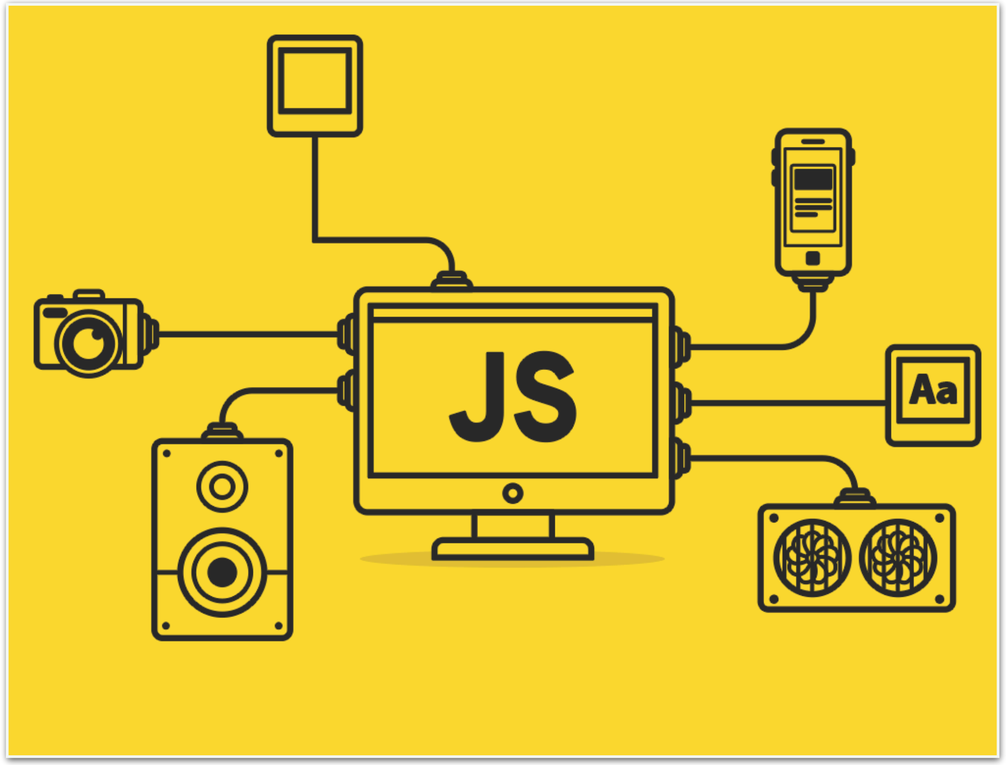
Anirach Mingkhwan

Anirach.m@fitm.kmutnb.ac.th



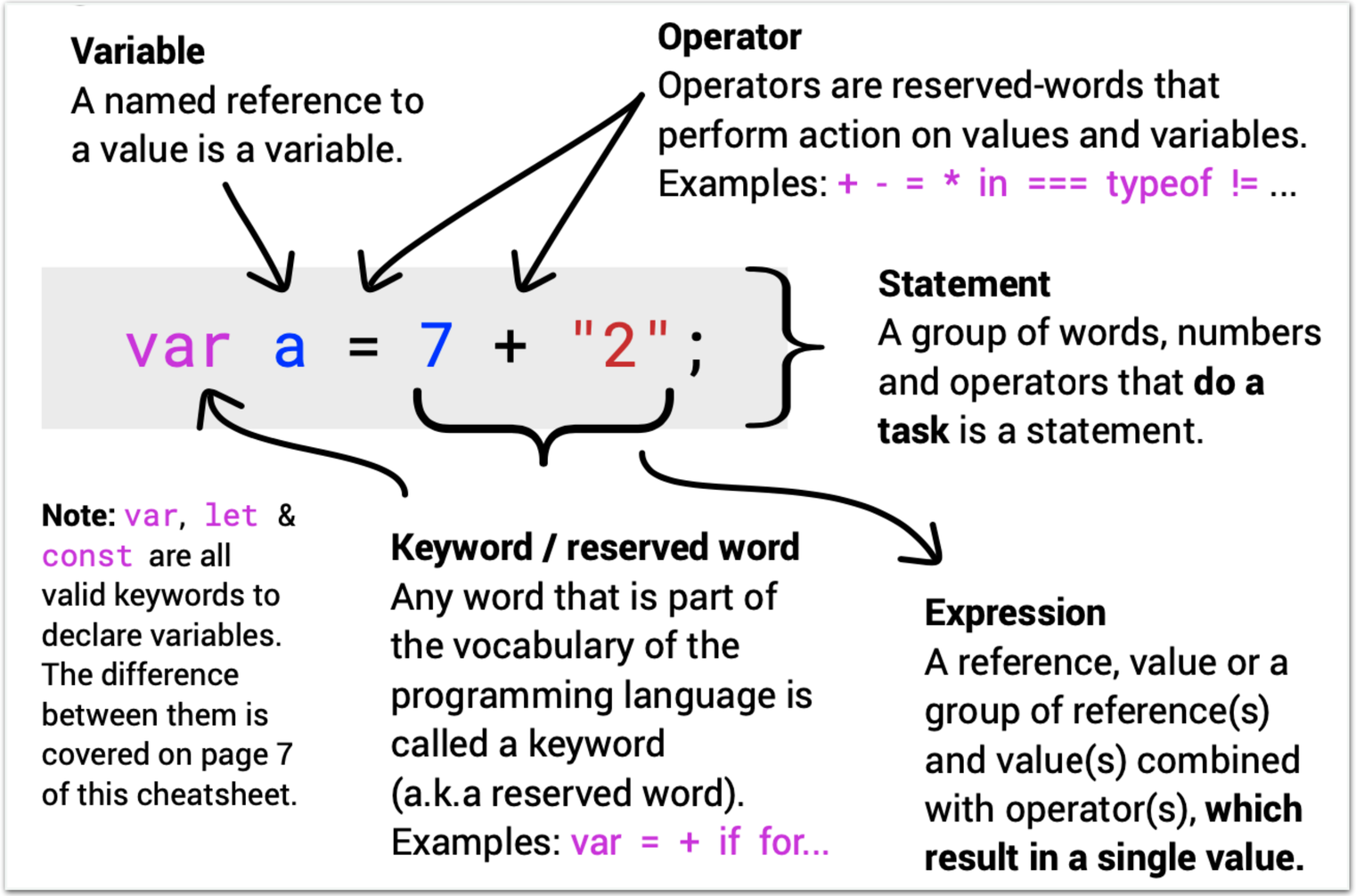
OUTLINE

- Data types & Variables
- Scope
- Condition
- Function
- Standard built-in objects



DATA TYPES & VARIABLES

BASIC VOCABULARY



Credit:https://ilovecoding.org/

RESERVED WORDS

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

Credit:https://w3schools.com/

VARIABLES

var, const, let

- var** : The most common variable. Can be reassigned but only accessed within a function. Variables defined with var move to the top when code is executed.
- const : Cannot be reassigned and not accessible before they appear within the code.
- let : Similar to const, however, let variable can be reassigned but not re-declared.

WAYS TO CREATE VARIABLE

There are 3 ways to create variables in JavaScript: `var`, `let` and `const`. Variables created with `var` are in scope of the function (or global if declared in the global scope); `let` variables are block scoped; and `const` variables are like `let` plus their values cannot be re-assigned.

```
var a = "some value";   // functional or global scoped
let b = "some value";   // block scoped
const c = "some value"; // block scoped + cannot get new value
```

Credit:https://ilovecoding.org/

USING VAR LET CONST

Credit:https://morioh.com/p/0a1cca1bf475

DATA TYPES

Six Primitive Types

1. String — `"Any text"`
2. Number — `123.45`
3. Boolean — `true` or `false`
4. Null — `null`
5. Undefined — `undefined`
6. Symbol — `Symbol('something')`

7. Object — `{ key: 'value'}`
   - Array — `[1, "text", false]`
   - Function — `function name() { }`

Credit:https://ilovecoding.org/

OBJECT

An object is a data type in JavaScript that is used to store a combination of data in a simple key-value pair. Thats it.

Key
These are the keys in user object.

```
var user = {
  name: "Aziz Ali",
  yearOfBirth: 1988,
  calculateAge: function(){
    // some code to calculate age
  }
}
```

Value
These are the values of the respective keys in user object.

Method
If a key has a function as a value, its called a method.

Credit:https://ilovecoding.org/

CODE DEMO
DATA TYPES & VARIABLES

Data Types

Lecture-02 DataTypes > JS datatypes.js > ...

```
1   const people = ["Aaron", "Mel", "John"];
2   const one = 1;
3   const str = "Hello World";
4   const b = true;
5   const employee = {
6     firstName: "Anirach",
7     lastName: "Mingkhwan",
8   };
9
10  function sayHello(person) {
11    console.log("Hello " + person.firstName);
12  }
13
14  console.log(typeof people);
15  console.log(typeof sayHello)
16  console.log(employee instanceof Array);
17  sayHello(employee);
```

```
> node datatypes.js
object
function
false
Hello Anirach
```

Display String

Lecture-02 DataTypes > JS index.js > ...

```
1   const message = "Hello";
2   const place = "World";
3
4   // logging out a string
5   console.log('Hello, World!');
6
7   // using substitutions
8   console.log('Hello, %s!', place);
9
10  // using a string literal
11  console.log(`${message}, ${place}!`);
12
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: zsh

```
> node index.js
Hello, World!
Hello, World!
Hello, World!

~/Code/JavaScript/Lecture-02 DataTypes on master !1 ?1 ..... at 13:10:50
>
```

var , let , const

Lecture-02 DataTypes > JS demovarletconst.js > ...

```
1   var hello = "Hello";
2
3   console.log(hello);
4   hello = "Hello World";
5   console.log(hello);
6
7   if (true) {
8       let world = "Hello World";
9       console.log(world);
10  }
11  // console.log(world);
12
13  const fixval = "Fix Value";
14  console.log(fixval);
15
16  // fixvar = "tes assigne value to const"
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: zsh

```
> node   demovarletconst.js
Hello
Hello World
Hello World
Fix Value

~/Code/JavaScript/Lecture-02 DataTypes on master !1 ?3 ..... at 16:36:30
>
```

Working with strings

Lecture-02 DataTypes > JS strconcat.js > ...

```
1   let str1 = "Hello";
2   let str2 = "World!";
3
4   //Using the + operator
5   console.log("********Using the + operator********\n");
6   console.log(str1 + str2);
7   console.log(str1 + "Big" + str2);
8
9   console.log("\n********Adding Spacing********\n");
10  //Adding Spacing
11  str1 = "Hello ";
12
13  console.log(str1 + str2);
14  console.log(str1 + "Big " + str2)
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
> node strconcat.js
********Using the + operator********

HelloWorld!
HelloBigWorld!

********Adding Spacing********

Hello World!
Hello Big World!

 ~/Code/JavaScript/Lecture-02 DataTypes
>
```

Be careful with numbers!

Lecture-02 DataTypes > JS strnum.js > ...

```
1   let num1 = 1;
2   let num2 = '1'
3
4   console.log(num1 + num2);
5   console.log(num1 + 1);
6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: zsh

```
> node strnum.js
11
2

 ~/Code/JavaScript/Lecture-02 DataTypes on  master ?1 ..... at 16:43:04
>
```

COERCION

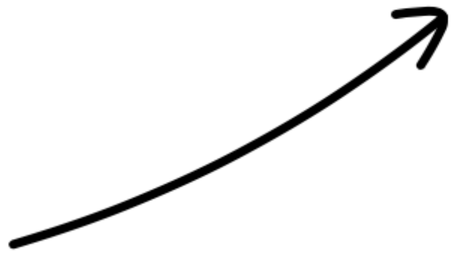
When trying to compare different "types", the JavaScript engine attempts to convert one type into another so it can compare the two values.

Type coercion priority order:

1. String
2. Number
3. Boolean

Coercion in action

Does this make sense?



```
2 + "7";   // "27"
true - 5   // -4
```

Credit:https://ilovecoding.org/

DIFERENCES IN FORMATING STRINGS

Concatenation Operator +	Template Literals
Needs single or double quotes	Needs one pair of backticks
Line breaks with newline character	Respects line breaks
Must concatenate variable values	Placeholders insert variable values and expressions

Credit:https://sec.ch9.ms

Template literals

Lecture-02 DataTypes > JS strtemplitr.js > ...

```
1   //Concatenation with template literals
2   console.log("\n********Concatenation with template literals********\n");
3
4   let str1 = "JavaScript";
5   let str2 = "fun";
6
7   console.log(`I am writing code in ${str1}`);
8   console.log(`Formatting in ${str1} is ${str2}!`);
9
10  //Expressions in template literals
11  console.log("\n********Expressions in template literals********\n");
12
13  let bool1 = true;
14  console.log(`1 + 1 is ${1 + 1}`);
15  console.log(`The opposite of true is ${!bool1}`);
16
```

```
> node strtemplitr.js

********Concatenation with template literals********

I am writing code in JavaScript
Formatting in JavaScript is fun!

********Expressions in template literals********

1 + 1 is 2
The opposite of true is false

~/Code/JavaScript/Lecture-02 DataTypes on master ?3
>
```

Null

Lecture-02-DataTypes > JS null.js > ...

```
1   function getVowels(str) {
2     const m = str.match(/[aeiou]/gi);
3     if (m === null) {
4       return 0;
5     }
6     return m.length;
7   }
8
9   console.log(getVowels("seeing"));
10
```

PROBLEMS OUTPUT TERMINAL ··· 1: zsh

```
> node null.js
3
```

The value null represents the intentional absence of any object value. It is one of JavaScript's primitive values and is treated as falsy for boolean operations.

Undefined

Lecture-02-DataTypes > JS undefined.js > ...

```
1  function test(t) {
2    if (t === undefined) {
3      return "Undefined value!";
4    }
5    return t;
6  }
7
8  let x;
9  console.log(test(x));
```

PROBLEMS OUTPUT TERMINAL ···

```
> node undefined.js
Undefined value!
```

The global undefined property represents the primitive value undefined. It is one of JavaScript's primitive types.

OPERATORS

BASIC OPERATORS

+	Addition
-	Subtraction
*	Multiplication
/	Division
(..)	Grouping operator
%	Modulus (remainder)
++	Increment numbers
--	Decrement numbers

BITWISE OPERATORS

&	AND statement
|	OR statement
~	NOT
^	XOR
<<	Left shift
>>	Right shift
>>>	Zero fill right shift

CODE DEMO
OPERATORS

Numbers and Math

Lecture-02 DataTypes > JS nummath.js > ...

```
1   let num1 = 100;
2
3   //Basic Math
4   console.log("********Basic Math********");
5   console.log(num1 + 25);
6   console.log(num1 - 100);
7   console.log(num1 * 100);
8   console.log(num1 / 1500);
9
10  console.log("********Additional Mathematical Operations********");
11  //Additional Mathematical Operations
12  console.log(num1 % 1500); // Remainder
13  console.log(++num1); //Increment
14  console.log(--num1); //Decrement
15
16  console.log("********Using the Math Object********");
17  //Using the Math Object
18  console.log(Math.PI); //Pi
19  console.log(Math.sqrt(num1)); //Square root
```

```
❯ node nummath.js
********Basic Math********
125
0
10000
0.06666666666666667
********Additional Mathematical Operations********
100
101
100
********Using the Math Object********
3.141592653589793
10
```

Numbers Conversion

Lecture-02-DataTypes > JS numconv.js > ...

```
1   let num1 = '150';
2   let flo1 = '1.50';
3
4   console.log("********Converting strings to integers********");
5   //Converting strings to integers
6   console.log(parseInt('100'));
7   console.log(parseInt(num1));
8   console.log(parseInt('ABC'));
9   console.log(parseInt('0xF')); //Hexadecimal number
10
11  console.log("********Converting strings to float********");
12  //Converting strings to float
13  console.log(parseFloat('1.25abc'));
14  console.log(parseFloat(flo1));
15  console.log(parseFloat('ABC'));
16
```

```
17  console.log("********More Conversion Examples********");
18  //More Conversion Examples
19  //Numbers after special characters are ignored
20  console.log(parseInt('1.5'));
21  console.log(parseInt('1 + 1'));
22
23  //Using Template Literals
24  console.log(parseInt(`${1 + 1}`));
25
26  console.log("********Converting numbers to strings********");
27  //Converting numbers to strings
28  console.log(num1.toString());
29  console.log(flo1.toString());
30  console.log((100).toString());
```

```
> node numconv.js
********Converting strings to integers********
100
150
NaN
15
********Converting strings to float********
1.25
1.5
NaN
```

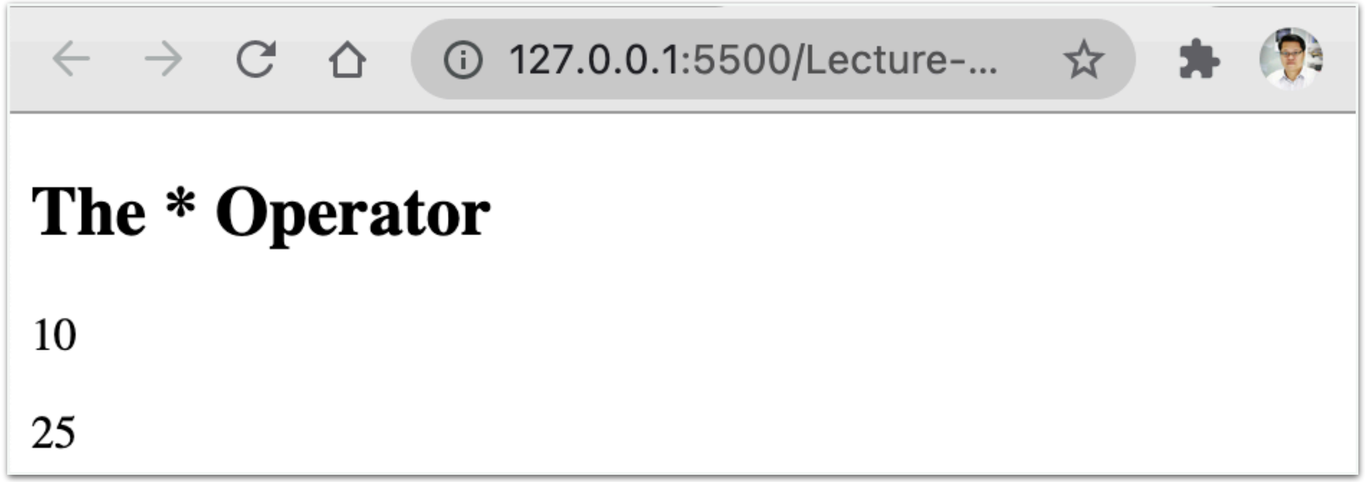
```
********More Conversion Examples********
1
1
2
********Converting numbers to strings********
150
1.50
100
```

Lecture-02-DataTypes > operator.html > ...

```html
<!DOCTYPE html>
<html>
  <body>
    <h2>The * Operator</h2>
    <p id="demo1"></p>
    <p id="demo2"></p>
    <script src="operator.js"></script>
  </body>
</html>
```

Lecture-02-DataTypes > JS operator.js > ...

```js
let x = 5;
let y = 2;
let z = x * y;
let a = x ** y;
document.getElementById("demo1").innerHTML = z;
document.getElementById("demo2").innerHTML = a;
```



CONDITIONAL

CONDITIONAL STATEMENTS

Conditional statements allow our program to run specific code only if certain conditions are met. For instance, lets say we have a shopping app. We can tell our program to hide the "checkout" button if the shopping cart is empty.

If -else Statement: Run certain code, "if" a condition is met. If the condition is not met, the code in the "else" block is run (if available.)

```
if (a > 0) {
  // run this code
} else if (a < 0) {
  // run this code
} else {
  // run this code
}
```

Ternary Operator: A ternary operator returns the first value if the expression is truthy, or else returns the second value.

```
(expression)? ifTrue: ifFalse;
```

Switch Statement: Takes a single expression, and runs the code of the "case" where the expression matches. The "break" keyword is used to end the switch statement.

```
switch (expression) {
  case choice1:
    // run this code
    break;

  case choice1:
    // run this code
    break;

  default:
    // run this code
}
```

There are certain values in JavaScript that return true when coerced into boolean. Such values are called **truthy** values. On the other hand, there are certain values that return false when coerced to boolean. These values are knows as **falsy** values.

Truthy Values

```
true
"text"
72
-72
Infinity
-Infinity
{}
[]
```

Falsy Values

```
false
""
0
-0
NaN
null
undefined
```

Credit:https://ilovecoding.org/

COMPARISON OPERATORS

==	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

Ternary Operator: A ternary operator returns the first value if the expression is truthy, or else returns the second value.

```
(expression)? ifTrue: ifFalse;
```

Equality Gotchas

```
let x = 0 == ''; // true, type coerced

let x = 0 === ''; // false, type respected
```

LOGICAL OPERATORS

&&	Logical and
||	Logical or
!	Logical not

CODE DEMO
CONDITIONAL IF AND SWITCH

If-else

Lecture-02 DataTypes > JS if-else.js > ...

```
1
2   const status = 200;
3
4   if (status === 200) {
5       console.log('OK!');
6   } else if (status === 400) {
7       console.log('Error!');
8   } else {
9       console.log('Unknown status');
10  }
11
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: zsh

```
> node if-else.js
OK!

~/Code/JavaScript/Lecture-02 DataTypes on master ?1
```

If-ternary

Lecture-02 DataTypes > JS if-ternary.js > ...

```
1   const status = 200;
2
3   const message = (status === 200) ? 'OK!' : 'Error!';
4
5   console.log(message);
6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: zsh

```
> node if-ternary.js
OK!

~/Code/JavaScript/Lecture-02 DataTypes on master ?2
>
```

OTHER BOOLEAN NOTES

Implicit false values

Strings

- Empty strings test as false

Objects

- Null or undefined objects test as false

Numbers

- 0 tests as false

```
const x = 0;
if (x) {
  console.log('x contains a value');
}
```

Use ! to reverse the result

```
const x = 0;
if (!x) {
  console.log('x contains *NO* value');
}
```

Strings are case sensitive

Convert to upper or lower case

Consider using `localeCompare`

```
const name = 'christopher';
if (name === 'Christopher') {
  console.log('The values are the same');
} else {
  console.log('The values NOT are the same');
}
```

Credit:https://sec.ch9.ms

Implicit-false

Lecture-02 DataTypes > JS implecit-false.js > ...

```
1   const name = '';
2
3   if (name) {
4       console.log('We have a name!');
5   } else {
6       console.log('No name provided');
7   }
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: zsh

```
> node implecit-false.js
No name provided

~/Code/JavaScript/Lecture-02 DataTypes on master ?3
```

Case-sensitive

Lecture-02 DataTypes > JS case-sensitive.js > ...

```
1    const status = 'error';
2
3    if (status.toUpperCase() === 'ERROR') {
4        console.log('Something went wrong!');
5    } else {
6        console.log('Looks great!!');
7    }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: zsh

```
› node case-sensitive.js
Something went wrong!
```

Using OR with if-else

Lecture-02-DataTypes > JS or.js > ...

```
1   const status = 500;
2
3   if (status === 200) {
4       console.log('OK!');
5   } else if (status === 400 || status === 500) {
6       console.log('Error!');
7   } else {
8       console.log('Unknown status');
9   }
```

PROBLEMS OUTPUT TERMINAL 1: zsh

```
› node or.js
Error!
```

Switch

Lecture-02 DataTypes > JS switch.js > status

```
1   const status = 700;
2
3   switch (status) {
4       case 200:
5           console.log('OK!');
6           break;
7       case 400: // or
8       case 500: // or
9           console.log('Error!');
10          break;
11      default: // else
12          console.log('Unknown status');
13  }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
> node switch.js
Error!
```

FUNCTION

A function is simply a bunch of code bundled in a section. This bunch of code ONLY runs when the function is called. Functions allow for organizing code into sections and code reusability.

Using a function has ONLY two parts. (1) Declaring/defining a function, and (2) using/running a function.

Parameters / Arguments (optional)
A function can optionally take parameters (a.k.a arguments). The function can then use this information within the code it has.

Name of function
Thats it, its just a name you give to your function.
Tip: Make your function names descriptive to what the function does.

```
// Function declaration / Function statement
function someName(param1, param2){

    // bunch of code as needed...
    var a = param1 + "love" +  param2;
    return a;
}

// Invoke (run / call) a function
someName("Me", "You")
```

Code block
Any code within the curly braces { ... } is called a "block of code", "code block" or simply "block". This concept is not just limited to functions. "if statements", "for loops" and other statements use code blocks as well.

Return (optional)
A function can optionally spit-out or "return" a value once its invoked. Once a function returns, no further lines of code within the function run.

Invoke a function
Invoking, calling or running a function all mean the same thing. When we write the function name, in this case someName, followed by the brackets symbol () like this someName(), the code inside the function gets executed.

Passing parameter(s) to a function (optional)
At the time of invoking a function, parameter(s) may be passed to the function code.

Credit:https://ilovecoding.org/

CODE DEMO
FUNCTION

First function

Lecture-02 DataTypes > JS 1-function.js > ...

```
1   // 1. Function Definition
2   function printHello(name){
3       console.log("Hello "+name);
4       return name +" hello!";
5   }
6
7   console.log(printHello.name)
8
9   // 2. Function Invocation
10  let result = printHello("Anirach !");
11  console.log(result);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
> node 1-function.js
printHello
Hello Anirach !
Anirach ! hello!
```

Functions Assigned to Variables

Lecture-02-DataTypes > function > JS variable-func.js > ...

```
1   let plusFive = (number) => {
2     return number + 5;
3   };
4   // f is assigned the value of plusFive
5   let f = plusFive;
6
7   console.log(plusFive(3)); // 8
8   // Since f has a function value, it can be invoked.
9   console.log(f(9)); // 14
10
```

```
> node variable-func.js
8
14

~/C/JavaScript/L
>
```

CODE DEMO
ARROW FUNCTION

Arrow function

Lecture-02 DataTypes > JS arrowfunctoin.js > ...

```
1   const add = (a, b) => a + b;
2   console.log(add(1, 2));
3
4   const subtract = (a, b) => {
5     const result = a - b;
6     return result;
7   };
8   console.log(subtract(4, 2));
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
> node arrowfunctoin.js
3
2
```

Arrow function example

Lecture-02-DataTypes > function > JS arrowf-sample.js > ...

```
1   // Arrow function with two arguments
2   const sum = (firstParam, secondParam) => {
3     return firstParam + secondParam;
4   };
5   console.log(sum(2, 5)); // Prints: 7
6
7   // Arrow function with no arguments
8   const printHello = () => {
9     console.log("hello");
10  };
11  printHello(); // Prints: hello
12
13  // Arrow functions with a single argument
14  const checkWeight = (weight) => {
15    console.log(`Baggage weight : ${weight} kilograms.`);
16  };
17  checkWeight(25); // Prints: Baggage weight : 25 kilograms.
18
19  // Concise arrow functions
20  const multiply = (a, b) => a * b;
21  console.log(multiply(2, 30)); // Prints: 60
22
```

```
> node arrowf-sample.js
7
hello
Baggage weight : 25 kilograms.
60

~/C/JavaScript/Lecture-02
>
```

CODE DEMO
CALLBACK FUNCTION

Callback function

Lecture-02-DataTypes > JS callbackfunction.js > ...

```
1   function createQuote(quote, callback) {
2     let myQuote = "Like I always say, " + quote;
3     callback(myQuote); // 2
4   }
5
6   function logQuote(quote) {
7     console.log(quote + ' Yes..');
8   }
9
10  createQuote(" you will getting better!", logQuote); // 1
11
```

PROBLEMS OUTPUT TERMINAL ··· 1: zsh

```
> node callbackfunction.js
Like I always say,  you will getting better! Yes..
```

SCOPE

SCOPE

Scope is a concept that refers to where values and functions can be accessed.

Various scopes include:

- *Global* scope (a value/function in the global scope can be used anywhere in the entire program)
- *File* or *module* scope (the value/function can only be accessed from within the file)
- *Function* scope (only visible within the function),
- *Code block* scope (only visible within a `{ ... }` codeblock)

```
function myFunction() {

  var pizzaName = "Volvo";
  // Code here can use pizzaName

}

// Code here can't use pizzaName
```

Credit:https://www.codecademy.com/learn/introduction-to-javascript/

BLOCK SCOPE

Block Scoped Variables

`const` and `let` are *block scoped* variables, meaning they are only accessible in their block or nested blocks. In the given code block, trying to print the `statusMessage` using the `console.log()` method will result in a `ReferenceError`. It is accessible only inside that `if` block.

```
const isLoggedIn = true;

if (isLoggedIn == true) {
  const statusMessage = 'User is logged in.';
}

console.log(statusMessage);

// Uncaught ReferenceError: statusMessage is not
defined
```

Credit:https://www.codecademy.com/learn/introduction-to-javascript/

GLOBAL VARIABLE

Global Variables

JavaScript variables that are declared outside of blocks or functions can exist in the *global scope*, which means they are accessible throughout a program. Variables declared outside of smaller block or function scopes are accessible inside those smaller scopes.

Note: It is best practice to keep global variables to a minimum.

```
// Variable declared globally
const color = 'blue';

function printColor() {
  console.log(color);
}

printColor(); // Prints: blue
```

Credit:https://www.codecademy.com/learn/introduction-to-javascript/

STANDARD BUILT-IN OBJECTS

INHERITED OBJECTS

String

Google 'Mozilla String' to find the docs

`.concat()`
`.charAt()`
`.indexOf()`
`.startsWith()`
`.endsWith()`
`.split()`
`.slice()`

Number

Google 'Mozilla Number' to find the docs

`.toFixed()`
`.toPrecision()`
`.toString()`

Boolean

Google 'Mozilla Boolean' to find the docs

`.toString()`

BUILT-IN OBJECTS

Math

Google 'Mozilla Math' to find the docs

```
Math.pow(2, 3)       // 8
Math.sqrt(16)        // 4
Math.min(7, 8, 6)    // 6
Math.max(7, 8, 6)    // 8
Math.floor(123.45)   // 123
Math.ceil(123.45)    // 124
Math.round(123.45)   // 123
Math.random()        // 0.45..
```

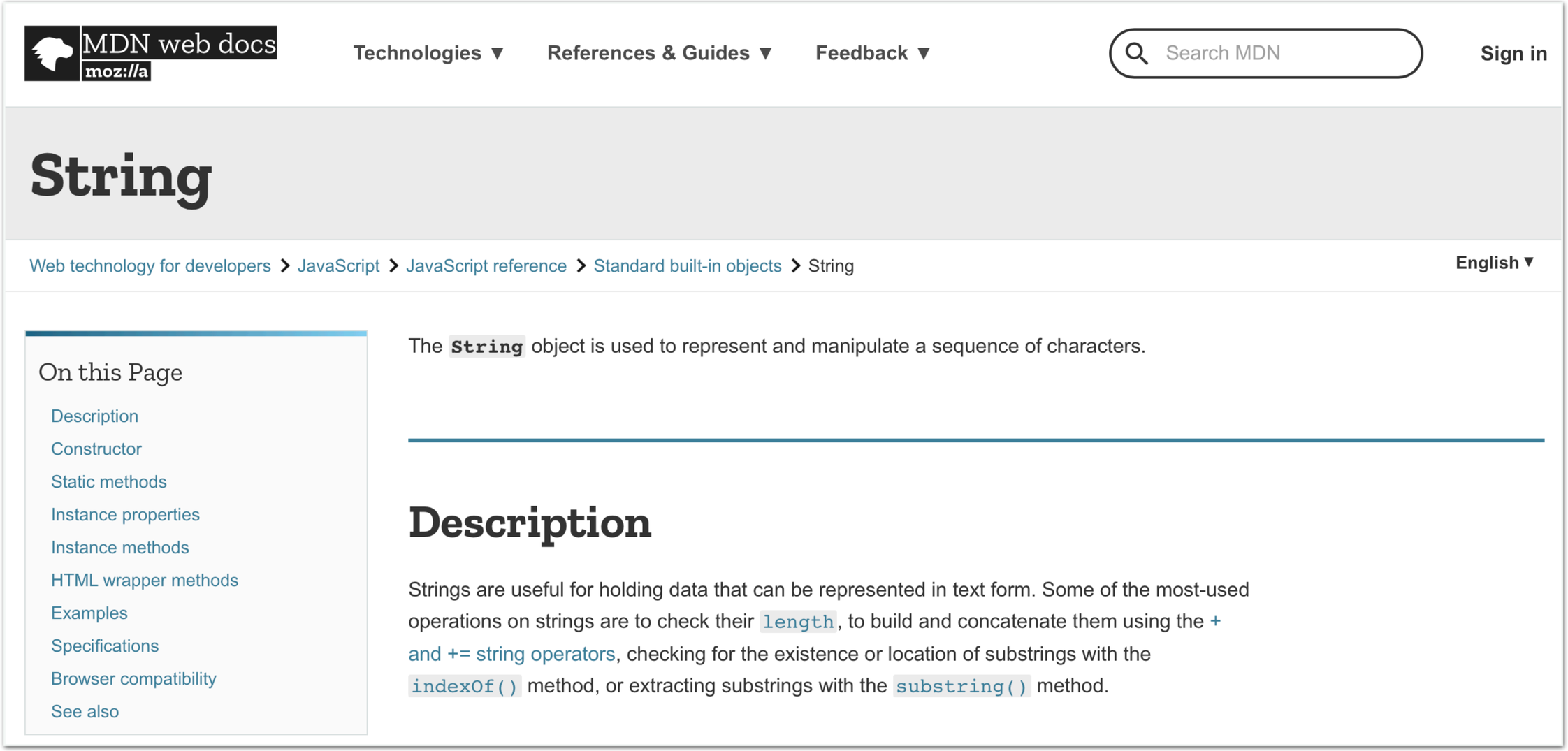
Date

Google 'Mozilla Date' to find the docs

```
const d = new Date('9/17/1988');
d.getDay()
d.getFullYear()
d.getMonth()

Date.now()
Milliseconds since Jan 1, 1970
```

STRING OBJECT



Credit: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

String Method

Lecture-02-DataTypes > JS strsplit.js > ...

```
1   const str = 'The quick brown fox jumps over the lazy dog.';
2
3   const words = str.split(' ');
4   console.log(words[3]);
5   // expected output: "fox"
6
7   const chars = str.split('');
8   console.log(chars[8]);
9   // expected output: "k"
```

Credit: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Arrow function

Lecture-02 DataTypes > JS reverse.js > ...

```
1   function reverseString(value) {
2       let reversedValue = "";
3
4       value.split("").forEach((char) => {
5           reversedValue = char + reversedValue;
6       });
7
8       return reversedValue;
9   }
10
11  console.log(reverseString("Reverse Me"));
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: zsh

```
> node reverse.js
eM esreveR

~/Code/JavaScript/Lecture-02 DataTypes on master !1 ?2 ..... at 13:46:56
>
```

String Object

Lecture-02-DataTypes > JS reverse2.js > ...

```
1   function reverseString(value) {
2
3       const reversedValue = value.split('').reverse().join('')
4       return reversedValue;
5   }
6
7   console.log(reverseString("Hello JavaScript"));
8
```

PROBLEMS OUTPUT TERMINAL ··· 1: zsh

```
> node reverse2.js
tpircSavaJ olleH

~/C/JavaScript/Lecture-02-DataTypes on master !1 ?6
```

NUMBER OBJECT

INHERITED OBJECTS

String

Google 'Mozilla String' to find the docs

```
.concat()
.charAt()
.indexOf()
.startsWith()
.endsWith()
.split()
.slice()
```

Number

Google 'Mozilla Number' to find the docs

```
.toFixed()
.toPrecision()
.toString()
```

Boolean

Google 'Mozilla Boolean' to find the docs

```
.toString()
```

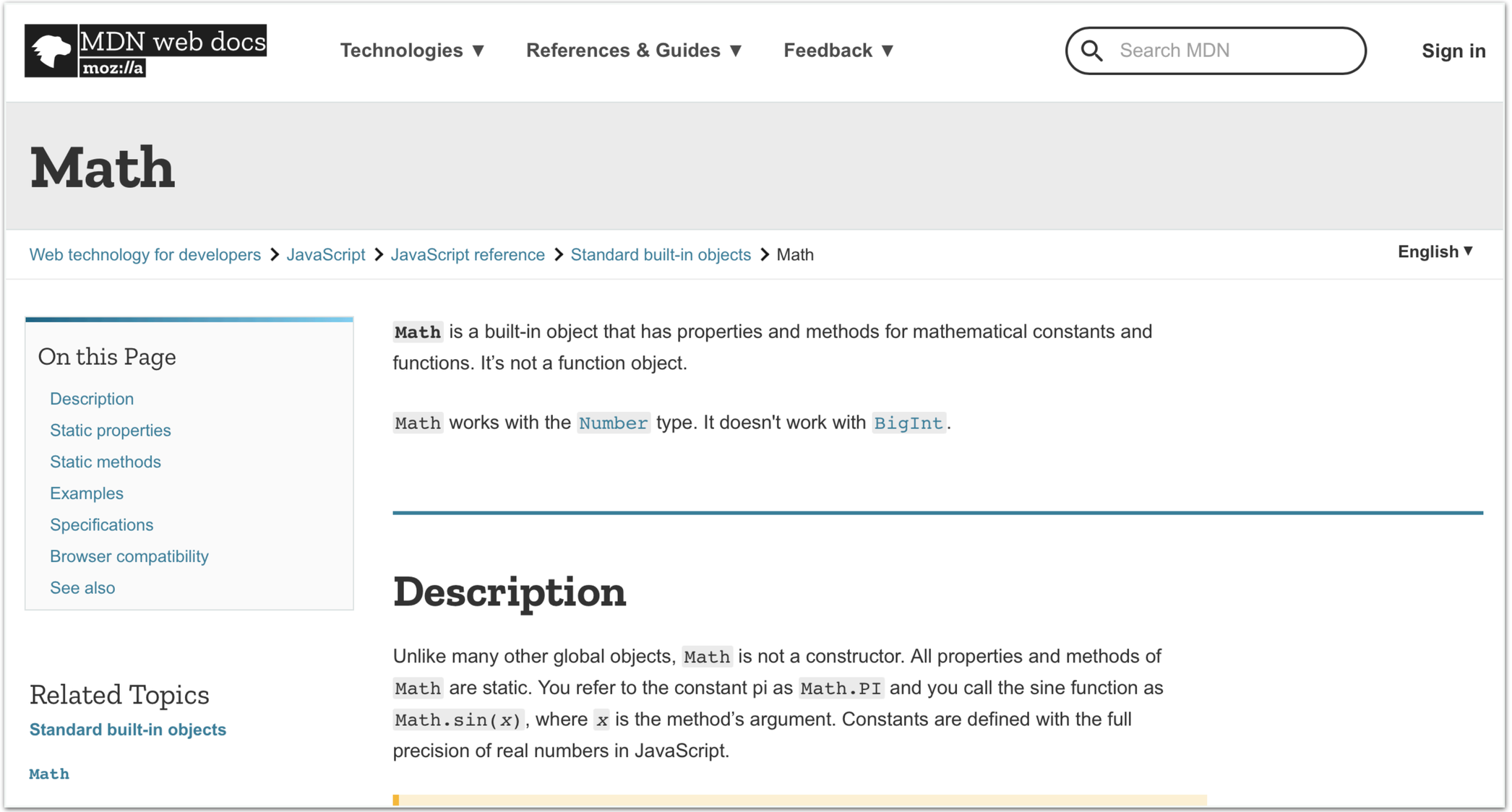
Number Object

Lecture-02-DataTypes > JS parsefloat.js > ...

```
1   function circumference(r) {
2     if (Number.isNaN(Number.parseFloat(r))) {
3       return 0;
4     }
5     return parseFloat(r) * 2.0 * Math.PI;
6   }
7
8   console.log(circumference("4.567abcdefgh"));
9   // expected output: 28.695307297889173
10
11  console.log(circumference("abcdefgh"));
12  // expected output: 0
```

Credit: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number

MATH OBJECT



Credit: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

Math Object

Lecture-02-DataTypes > JS mathobj.js

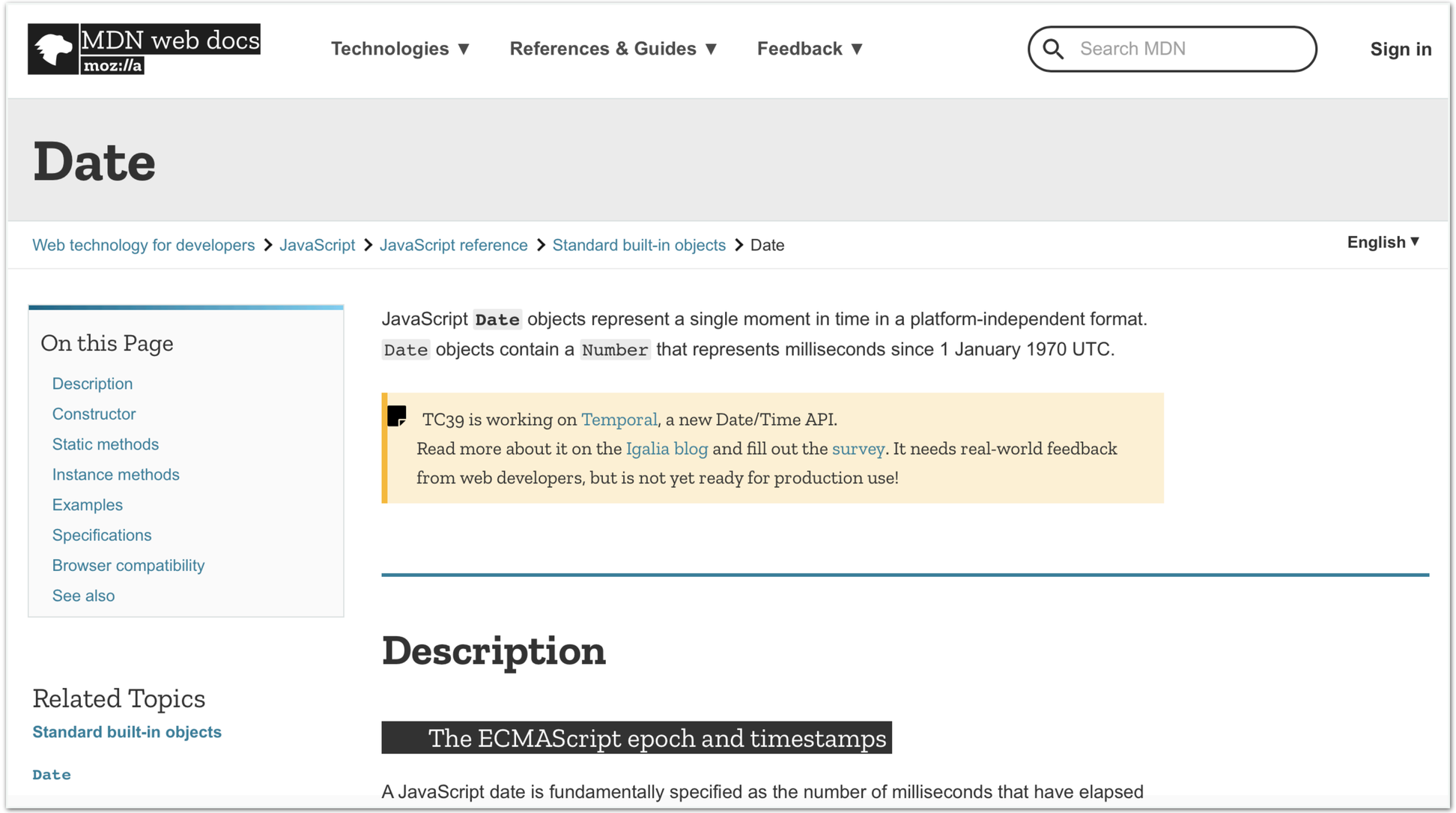
```
1   console.log(Math.floor(5.05));
2   // expected output: 5
3
4   console.log(Math.ceil(7.004));
5   // expected output: 8
6
7   console.log(Math.random());
8   // expected output: a number from 0 to <1
9
10  console.log(Math.min(-2, -3, -1));
11  // expected output: -3
12
```

PROBLEMS OUTPUT TERMINAL ··· 1: zsh

```
> node mathobj.js
5
8
0.9703198535857198
-3
```

Credit: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

DATE OBJECT



Credit: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

Date Object

Lecture-02-DataTypes > JS dateobj.js > ...

```
1   //Date function return milliseconds that have elapsed
2   //since midnight on January 1, 1970, UTC
3   // this example takes 2 seconds to run
4   const start = Date.now();
5
6   console.log('starting timer...');
7   // expected output: starting timer...
8
9   setTimeout(() => {
10    const millis = Date.now() - start;
11
12    console.log(`seconds elapsed = ${Math.floor(millis / 1000)}`);
13    // expected output: seconds elapsed = 2
14  }, 2000);
15
```

PROBLEMS OUTPUT TERMINAL ··· 1: zsh

```
> node dateobj.js
starting timer...
seconds elapsed = 2
```

Credit: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

EXCERCISE

Ex-01

Create a function that takes `length` and `width` and finds the perimeter of a rectangle.

Examples

```
findPerimeter(6, 7) → 26

findPerimeter(20, 10) → 60

findPerimeter(2, 9) → 22
```

Ex-02

Create a function that takes an equation (e.g. `"1+1"`), and returns the answer.

Examples

```
equation("1+1") → 2

equation("7*4-2") → 26

equation("1+1+1+1+1") → 5
```

THANK YOU

- Data type, Variable
- Operators
- Conditional
- Standard built-in objects