# Section 2: JavaScript Fundamentals – Part 1

## JavaScript

A high-level, object-oriented, multi-paradigm programming language

(Multi-paradigm 🡪 Can be written in different styles eg. imperative and declarative)

## Role of JS

Diagram

Description automatically generated

## Data types

Value is the one that has a type, NOT the variable.

Value can be an object or 1 of 7 primitive data types:

1. Number
2. String
3. Boolean
4. Undefined
5. Null (There’s a legacy bug whereby returns “object”)
6. Symbol (ES2015) 🡪 Value that is unique and cannot be changed
7. BigInt (ES2020)

Diagram

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

## JS operator precedence

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence>

The . in is consider as an operator as well, so it has its own precedence

## Type Conversion vs Type Coercion

|  |  |
| --- | --- |
| **Type Conversion** | **Type Coercion** |
| Manually convert a data type to another | Happens when an operator (arithmetic, logical, comparison, assignment) encounters 2 values of different types  Converts 1 value’s type to match another’s type to ensure the operator’s execution |
| Converts to Number / String / Boolean  (ie. Number(“21”) / String(21) / Boolean (…)) | Eg.  STRING + NUMBER -> NUMBER converted to STRING  STRING – NUMBER -> STRING converted to NUMBER |
| Invalid conversion returns invalid value of that type  (eg. Number(“abc”) -> NaN) | Same as Type Conversion |

## Truthy and Falsy values

|  |  |
| --- | --- |
| **Falsy values** | **Truthy values** |
| Values that become false when converted into Boolean (eg. w/ logical operators or in a logical context) | Values that become true when converted into a Boolean |
| Only 5 falsy values: 0, “”, undefined, null, NaN | Any value that is not falsy |

## Strict vs Loose equality

Strict (=== or !==) 🡪 Both sides have the same data type and value

Loose (== or !=) 🡪 Perform type coercion before comparing their data types and values

## Expression vs Statement

Expression provides a value

Statement indicates an action but does not provide a value directly

## JavaScript Version & History

Text

Description automatically generatedDiagram

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

## Strict Mode in JS (Recommended)

Opting into Strict Mode allows JS to:

1. Throw errors that are otherwise silent
2. Prevent syntax usage that are likely to be defined in future version of ECMAScript

Global 🡪 Need to enter “use strict”; at the 1st line of the .js file

Module 🡪 Strict by default

<https://262.ecma-international.org/6.0/#sec-strict-mode-code>

Text

Description automatically generated

Use Strict Directive 🡪 “use strict”;

## Creating Functions in JS

|  |  |
| --- | --- |
| **Function declaration** | function concat1(str1, str2) {  return `${str1}${str2}`;  }   * Hoisted to the top of the .js file   + Function can be used before the declaration |
| **Function expression** | const concat2 = function (str1, str2) {  return `${str1}${str2}`;  };   * Storing function into a variable since **a function is a type of value** |
| **Arrow function** | const concat3 = (str1, str2) => `${str1}${str2}`;   * Short form of function expression * Cannot use keyword |

## Accessing an Object’s member

|  |  |
| --- | --- |
| **Dot notation** | Party.tanker  Use when the member’s name is known at compile time |
| **Bracket notation** | Party[“tanker]  Use when the member’s name is only known at run time |

## Arrays

A special type of Object in JS, its methods like are just an Object’s method