ECMAScript 2015+

FROM 'CAPABILITY' TO 'EXPERTISE'

Installation

- Install Node Latest LTS version - https://nodejs.org/en/
- Install / Update Visual Studio Code to Latest Release - https://code.visualstudio.com/
- Extensions of Visual Studio Code
  - AutoFileName
  - vscode-icons
  - JavaScript (ES6) code snippets
- File Menu -> Preferences -> Theme -> File Icon Theme -> Select VScode Icons

Babel

- Babel is a JavaScript compiler.
- Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.
- Here are the main things Babel can do for you:
  - Transform syntax
  - Polyfill features that are missing in your target environment (through a third-party polyfill such as core-js)
  - Source code transformations (codemods)

Polyfill

- A polyfill is a piece of code (usually JavaScript on the Web) used to provide modern functionality on older browsers that do not natively support it.
- core-js
  - Modular standard library for JavaScript.
  - Includes polyfills for ECMAScript up to 2021: promises, symbols, collections, iterators, typed arrays, many other features, ECMAScript proposals, some cross-platform WHATWG / W3C features and proposals like URL.

Create and Compile

- Create a new project folder
- Open the project folder path in terminal window & run the following commands
  - npm init -y
  - npm i -D @babel/core @babel/preset-env @babel/cli
  - npm i core-js
- Create babel.config.json file with configuration inside the project folder
- To Compile
  - npx babel --version
  - npx babel src --out-dir dist

Steps to run the application

- Download and Extract the zip file.
- Open the extracted folder in Visual Studio Code
- Open the terminal window on the folder path and run
  - npm install
- Compile the code
  - npx babel src --out-dir dist
- Once the compilation completes, run from terminal window
  - node dist/main.js

Modular JavaScript

- **Module loaders** and **Module bundlers** both make it more actionable to write modular JavaScript applications.
- Module loaders
  - A module loader is typically some library that can load, interpret and execute JavaScript modules you defined using a certain module format/syntax, such as AMD or CommonJS.
- Module bundlers
  - Module bundlers are an alternative to module loaders.
  - They do the same thing (manage and load interdependent modules) but do it as part of the application build rather than at runtime.
  - The primary advantage of a bundler is that it leaves you with far fewer files that the browser must download.
  - This can give your application a performance advantage, as it *may* decrease the amount of time it takes to load.

Build Tools

- Build Tools are utilities that help automate the transformation and bundling of your code into a single file. Most JavaScript projects use these build tools to automate the build process.
- There are several common build tools available that can be integrated with Babel.
- Some of the tools are:
  - Browserify
  - Grunt
  - Gulp
  - **Webpack**

Webpack

- We need a module bundler like Webpack to resolve dependencies.
- Webpack can understand the relationships between multiple files and "resolve" them in a process called dependency resolution.
- Dependency resolution doesn't just which files are needed but the order they are required in since any dependency can rely on another.
- Webpack Dev Server:
  - A NodeJS server and its purpose is to propagate changes from your webpack bundle to your browser during the development of your app
  - Hot Module Reloading:
    - This feature allows the app to continue running in its current state whilst the parts you are changing can be swapped out live.

Client Side Build

Manual Configuration
- Learn Node JS
- Learn Webpack
- Learn Babel & Config.

npm start

Build & Deploy your Project on Local Server

WEBPACK-DEV-SERVER

npm run build

Build your project

WEBPACK CLI

WEBPACK

ES 2015 + (Multiple Files)

Babel + Preset (Env) + Polyfill (core-js)

Browser Compatible Files (Multiple Files)

Bundling

Single File / Set of Files

Inject Bundles

HTML PAGE

Deploy

Local HTTP Server

BROWSER

Webpack Configuration

- Entry Point
- Extensions
- Loaders (Used to pre-process files)
- Plugins
- Optimization
- Output

Steps to run the application

- Download and Extract the zip file.
- Open the extracted folder in Visual Studio Code
- Open the terminal window on the folder path and run
  - npm install
- Once the installation completes, run
  - npm start – Starts the Development Server on localhost:3000
  - npm run build – To create a production build in dist folder
  - npm run build-serve – To create a production build in dist folder and serve it locally

Variables

- Variables are containers for storing data (storing data values).
- There are 4 Ways to Declare a JavaScript Variable:
  - Using var
  - Using let
  - Using const
  - Using nothing (Not Recommended)
- Always declare JavaScript variables with var, let, or const.
- The var keyword is used in all JavaScript code from 1995 to 2015.
- The let and const keywords were added to JavaScript in 2015.
- If you want your code to run in older browser, you must use var.

Datatypes

- The set of types in the JavaScript language consists of **primitive values** and **objects**.
  - Primitive Values - Immutable datum represented directly at the lowest level of the language
    - Null – null
    - Undefined – undefined
    - Boolean() – boolean
    - Number() – number
    - String() – string
    - Symbol (ECMAScript 2015) – symbol – **We cannot use new keyword with Symbol()**
  - Objects - An object is a mutable value in memory which is possibly referenced by an identifier.
    - Object() - object

JavaScript types

- All types except objects define immutable values (that is, values which can't be changed)
- **Boolean type:** Boolean represents a logical entity and can have two values: true and false. See Boolean and Boolean for more details.
- **Null type:** The Null type has exactly one value: null. See null and Null for more details.
- **Undefined type:** A variable that has not been assigned a value has the value undefined. See undefined and Undefined for more details.
- **Number type:** The Number type is a double-precision 64-bit binary format value (numbers between -(2^53 − 1) and 2^53 − 1). In addition to representing floating-point numbers, the number type has three symbolic values: +Infinity, -Infinity, and NaN ("Not a Number").
- **String type:** JavaScript's String type is used to represent textual data. It is a set of "elements" of 16-bit unsigned integer values. Each element in the String occupies a position in the String.
- **Symbol type:** A Symbol is a unique and immutable primitive value and may be used as the key to an Object property.
- **Objects:** An object is a value in memory which is possibly referenced by an identifier. objects can be seen as a collection of properties. Properties are identified using key values. A key value is either a String value or a Symbol value.

Mutable vs Immutable

- JavaScript Engine handles mutable and immutable objects differently.
- Immutable are quicker to access than mutable objects.
- Mutable objects are great to use when you need to change the size of the object, example array, set etc..
- Immutable are used when you need to ensure that the object you made will always stay the same.
- Immutable objects are fundamentally expensive to “change”, because doing so involves creating a copy.
- Changing mutable objects is cheap.

Standard built-in objects

- Value properties
  - Infinity
  - NaN
  - undefined
  - globalThis

- Function properties
  - eval()
  - uneval()
  - isFinite()
  - isNaN()
  - parseFloat()
  - parseInt()
  - encodeURI()
  - encodeURIComponent()
  - decodeURI()
  - decodeURIComponent()

- Fundamental objects
  - Object
  - Function
  - Boolean
  - Symbol

- Error objects
  - Error
  - AggregateError
  - EvalError
  - InternalError
  - RangeError
  - ReferenceError
  - SyntaxError
  - TypeError
  - URIError

Standard built-in objects

- Numbers and dates
  - Number
  - Math
  - Date
- Text processing
  - String
  - RegExp
- Indexed collections
  - Array
  - Int8Array
  - Uint8Array
  - Uint8ClampedArray
  - Int16Array
  - Uint16Array
  - Int32Array
  - Uint32Array
  - Float32Array
  - Float64Array

Standard built-in objects

- Keyed collections
  - Map
  - Set
  - WeakMap
  - WeakSet
- Structured data
  - ArrayBuffer
  - DataView
  - JSON
- Control abstraction objects
  - Promise
  - Generator
  - GeneratorFunction
  - AsyncFunction
  - AsyncGenerator
  - AsyncGeneratorFunction
- Reflection
  - Reflect
  - Proxy

Operators

Table 5.1 Precedence and Associativity

Operator	*Description*	*Associativity*
()	Parentheses	Left to right
++ --	Auto increment, decrement	Right to left
!	Logical NOT	Right to left
* / %	Multiply, divide, modulus	Left to right
+ -	Add, subtract	Left to right
+	Concatenation	Left to right
< <=	Less than, less than or equal to	Left to right
> >=	Greater than, greater than or equal to	Left to right
== !=	Equal to, not equal to	Left to right
=== !==	Identical to (same type), not identical to	Left to right
&	Bitwise AND	Left to right
|	Bitwise OR	
^	Bitwise XOR	
~	Bitwise NOT	
<<	Bitwise left shift	
>>	Bitwise right shift	
>>>	Bitwise zero-filled, right shift	
&&	Logical AND	Left to right
||	Logical OR	Left to right
? :	Ternary, conditional	Right to left
= += -= *= /= %= <<= >>=	Assignment	Right to left
,	(comma)	

Symbol

- In JavaScript, a Symbol is a primitive data type that represents a unique identifier.
- One important thing to note about Symbol is that each Symbol value is unique, even if it has the same description. This makes it useful for creating object keys that won't clash with other keys.

Statements

- Control Flow
  - Block { }
  - break
  - continue
  - Empty
  - if…else
  - switch
  - throw
  - try…catch
- Iterations
  - for
  - for...in
  - for…of
  - while
  - do…while

Functions

- Functions are one of the fundamental building blocks in JavaScript.
- A function in JavaScript is like a procedure—a set of statements that performs a task or calculates a value, but for a procedure to qualify as a function, it should take some input and return an output.
- To use a function, you must define it somewhere in the scope from which you wish to call it.
- In JavaScript, functions are first-class objects, because they can have properties and methods just like any other object.
- What distinguishes them from other objects is that functions can be called.
- Every JavaScript function is a Function object.

Functions

- function
  - Declares a function with the specified parameters.
- function*
  - Generator Functions, enable writing iterators more easily
- async function
  - Declares an async function with the specified parameters.

Function Parameters & Arguments

- Function **Parameters** are the names that are define in the function definition and real values passed to the function in function definition are known as **arguments**.
- Parameter Rules:
  - There is no need to specify the data type for parameters in JavaScript function definitions.
  - It does not perform type checking based on passed in JavaScript function.
  - It does not check the number of received arguments.
- If a function is called with missing arguments (less than declared), the missing values are set to undefined.

The Arguments Object

- JavaScript functions have a built-in object called the arguments object.
- The argument object contains an array of the arguments used when the function was called (invoked).
- If a function is called with too many arguments (more than declared), these arguments can be reached using the arguments object.

Rest & Spread

- REST
  - The rest operator (also represented by three dots, "...") allows you to represent an indefinite number of arguments as an array.
  - This operator is used when defining a function that takes a variable number of arguments.
  - The rest operator also allows us in de-structuring array or objects.
- SPREAD
  - The spread operator (represented by three dots, "...") allows you to spread an array or object into another array or object.
  - This operator can be used to create a new array or object with the properties of the existing array or object, or to concatenate arrays.

Pure and Impure Functions

- A side-effect is any operation your function performs that is not related to computing the final output, including but not limited to:
  - Modifying a global variable
  - Modifying an argument
  - Making HTTP requests
  - DOM manipulation
  - Reading/writing files
- A pure function must both be predictable and without side-effects. If either of these criteria is not met, we're dealing with an impure function.
- A pure function, returns same result if same arguments are passed in, no matter how many times it runs.
- An impure function, may return different result if same arguments are passed in, on multiple runs

IIFE

- **Immediately Invoked Function Expression** (IIFE) is one of the most popular design patterns in JavaScript.
- It is pronounced as, iify.
- As name suggest, IIFE is a function expression that automatically invokes after completion of the definition.
- IEFE solves this problem by having its own scope and restricting functions and variables to become global.
- The functions and variables declare inside IIFE will not pollute global scope even they have same name as global variables & functions.

Callback

- A callback is a function that is passed as an argument to another function, which is expected to invoke it either immediately or at some point in the future.
- Callbacks can be seen as a form of the continuation-passing style (CPS), in which control is passed explicitly in the form of a continuation; in this case the callback passed as an argument represents a continuation.
- There are two types of callbacks.
  - A callback passed to a function can be invoked **synchronously** before return
  - Or it can be deferred to execute **asynchronously** some time after return.

Closure

- A closure is a function or reference to a function together with a referencing environment — a table storing a reference to each of the non-local variables of that function.
- Closures are functions that refer to independent (free) variables. In other words, the function defined in the closure 'remembers' the environment in which it was created.
- A closure is a special kind of object that combines two things: a function, and the environment in which that function was created. The environment consists of any local variables that were in-scope at the time that the closure was created.

Closure

- A closure is a function that “encloses” variables from its surrounding scope, allowing it to access those variable even after the surrounding scope has exited.
- A closure retains access to variables that were present in its surrounding scope when it was created, even if those variables are no longer in scope.
- Closures are often used in JavaScript to create private variables or to preserve the state of a function across multiple invocations.
- They can also be used to create callback functions or to create functions that are passed as arguments to other functions.

JavaScript ‘this’ keyword

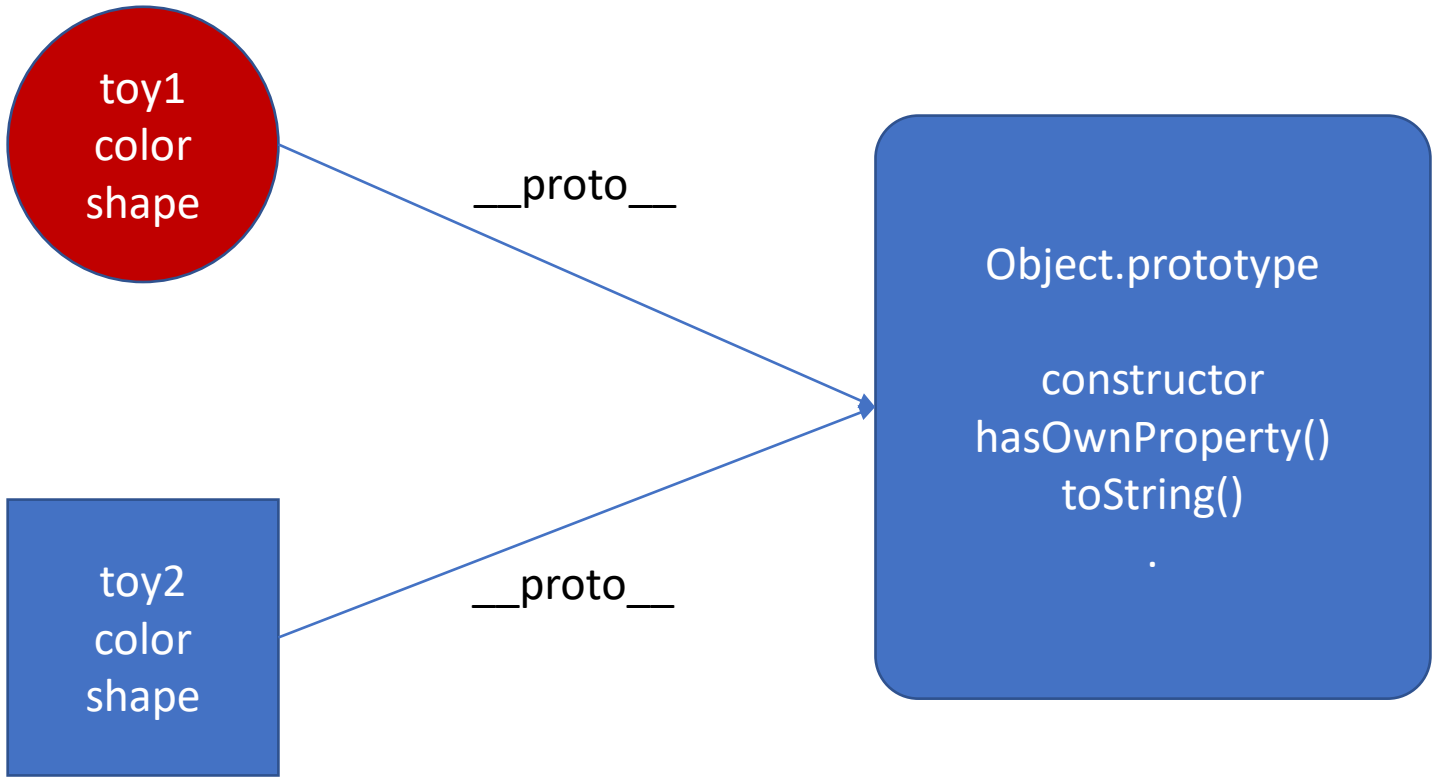
- In JavaScript, **“context”** refers to an object.
- Within an object, the keyword **“this”** refers to that object, and provides an interface to the properties and methods that are members of that object.
- When a function is executed, the keyword “this” refers to the object that the function is executed in.
- Scenarios
  - When a function executes in the global context, “this” refers to the global, or “window” object
  - When a function is a method of an Object, “this” refers to that object (unless it is manually executed in the context of a different object)
  - When a function executes inside of another function (no matter how deeply nested), “this” refers to the object whose context within which it is executed
  - When you instantiate a constructor function, inside of the instance object, “this” refers to the instance object

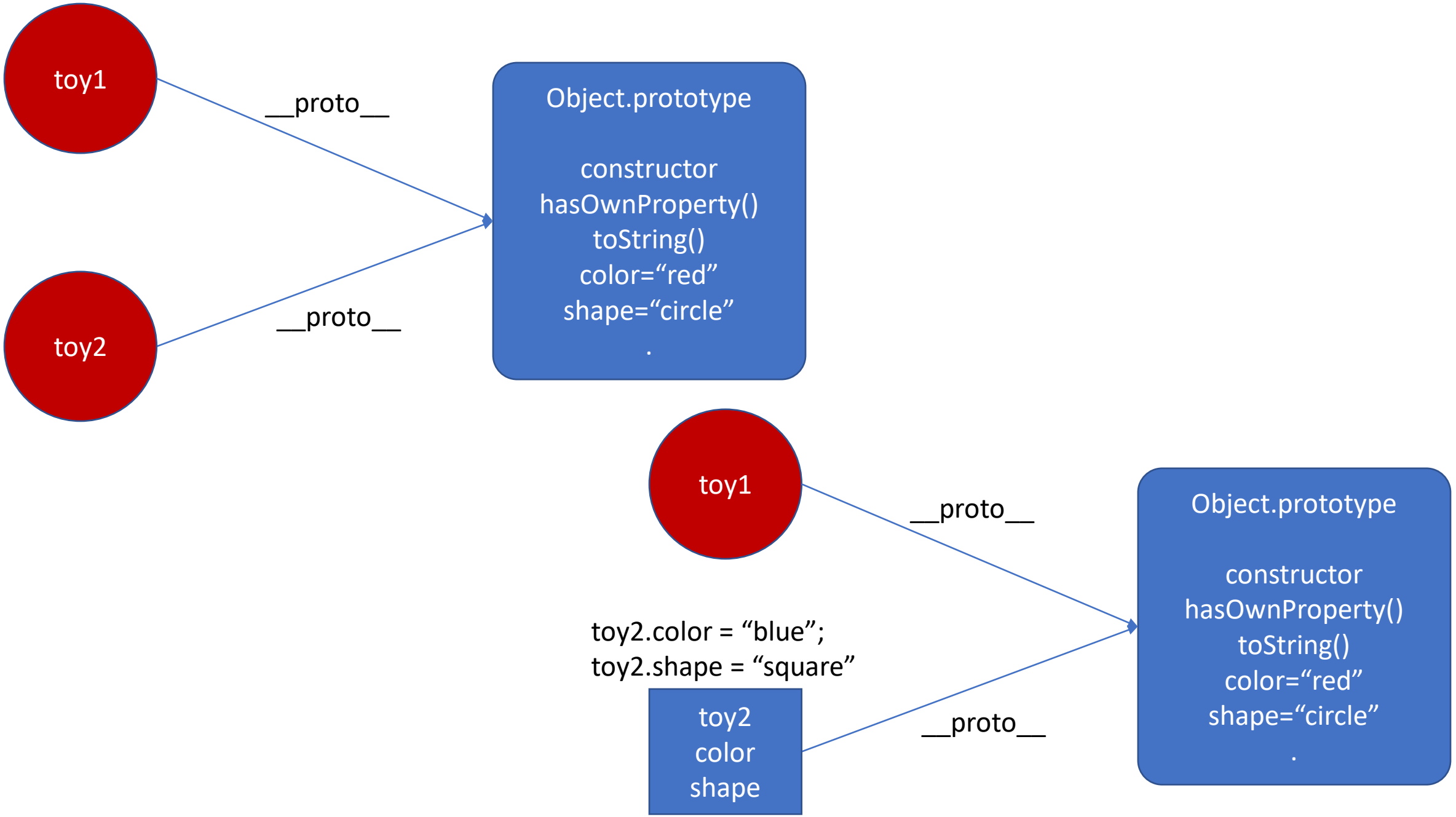
Function Currying

- Currying is a way of constructing functions that allows partial application of a function's arguments.
- What this means is that you can pass all the arguments a function is expecting and get the result or pass a subset of those arguments and get a function back that's waiting for the rest of the arguments.

Higher Order Functions

- Functions that operate on other functions, by taking them as arguments and by returning them, are called higher-order functions.
- Higher-order functions allow us to abstract over actions, not just values.
- Higher-order functions are functions that take one or more functions as arguments or return a function as a result.
- These functions are a fundamental concept of functional programming and allow developers to create more modular and reusable code.

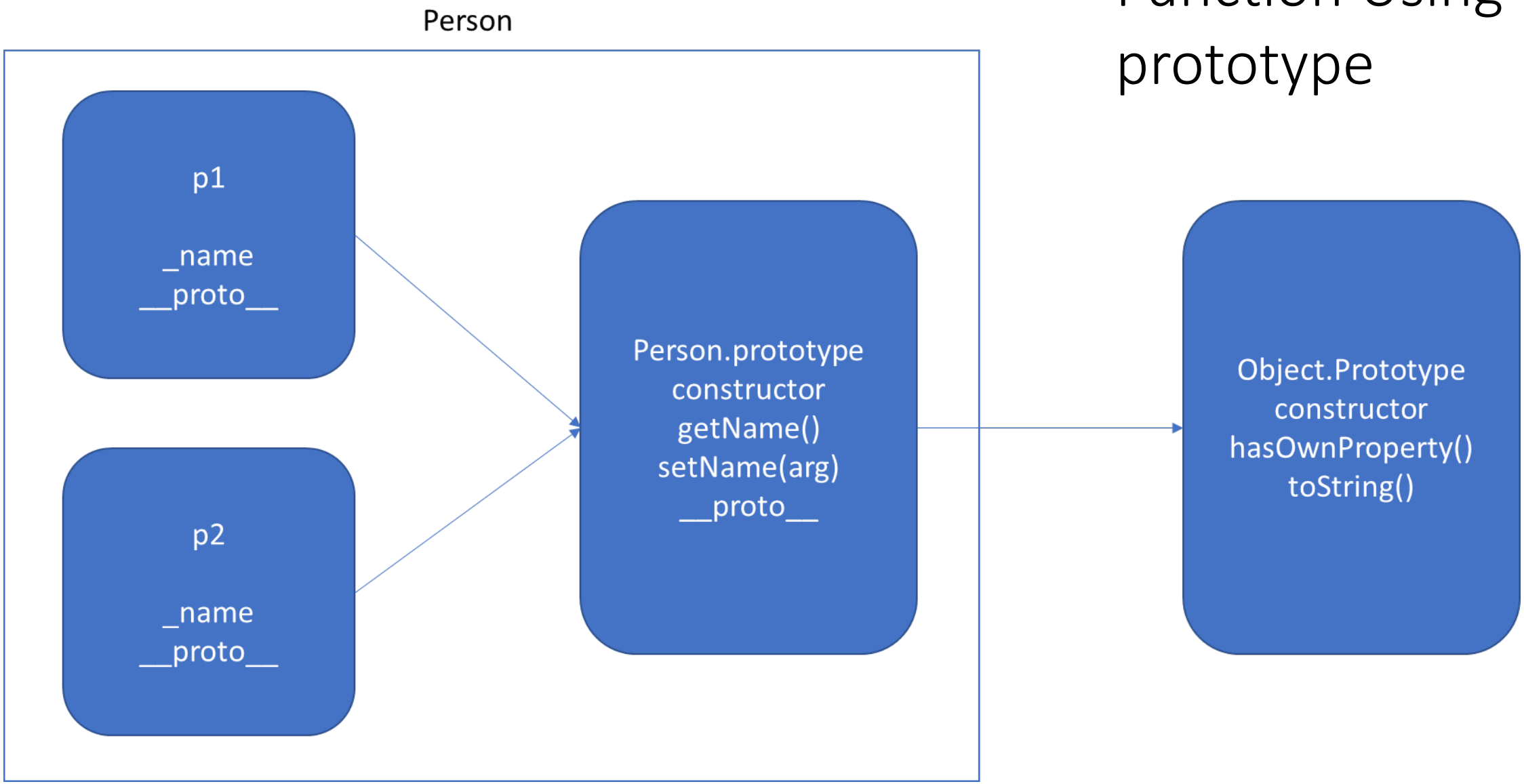

toy1
color
shape
__proto__
toy2
color
shape
__proto__
Object.prototype
constructor
hasOwnProperty()
toString()
.


toy1
__proto__
Object.prototype
constructor
hasOwnProperty()
toString()
color="red"
shape="circle"
.
toy2
__proto__
toy1
__proto__
Object.prototype
constructor
hasOwnProperty()
toString()
color="red"
shape="circle"
.
toy2.color = "blue";
toy2.shape = "square"
toy2
color
shape
__proto__

Constructor Function

- In JavaScript, a constructor function is used to create objects.
- To create an object from a constructor function, we use the new keyword.
- It is considered a good practice to capitalize the first letter of your constructor function.
- When this keyword is used in a constructor function, this refers to the object when the object is created.

Function Using prototype



ES6 Class Syntax

- A class is a type of object constructor that provides a template for creating objects with similar properties and methods.
- It is a way of organizing and encapsulating related data and functionality into a single unit.
- The class syntax provides a simpler and more concise way to define objects and their methods, compared to the older syntax that used constructor functions and prototypes.

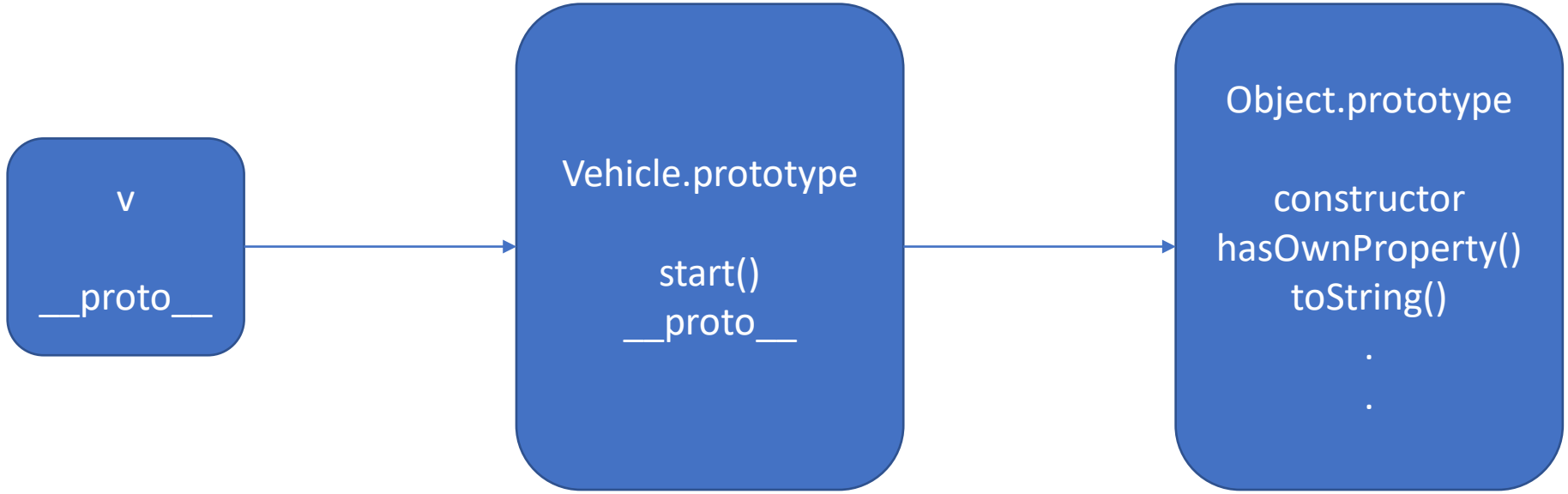
Accessor Property

- In JavaScript, accessor properties are methods that get or set the value of an object.
- Accessor properties are represented by “getter” and “setter” methods.
- Object.defineProperty() is a built-in function in JavaScript that allows you to define new properties on an object or modify the attributes of existing properties.
- It is a powerful tool for creating custom, non-enumerable properties with custom behaviors and constraints.
- We use these two keywords:
  - get - to define a getter method to get the property value
  - set - to define a setter method to set the property value

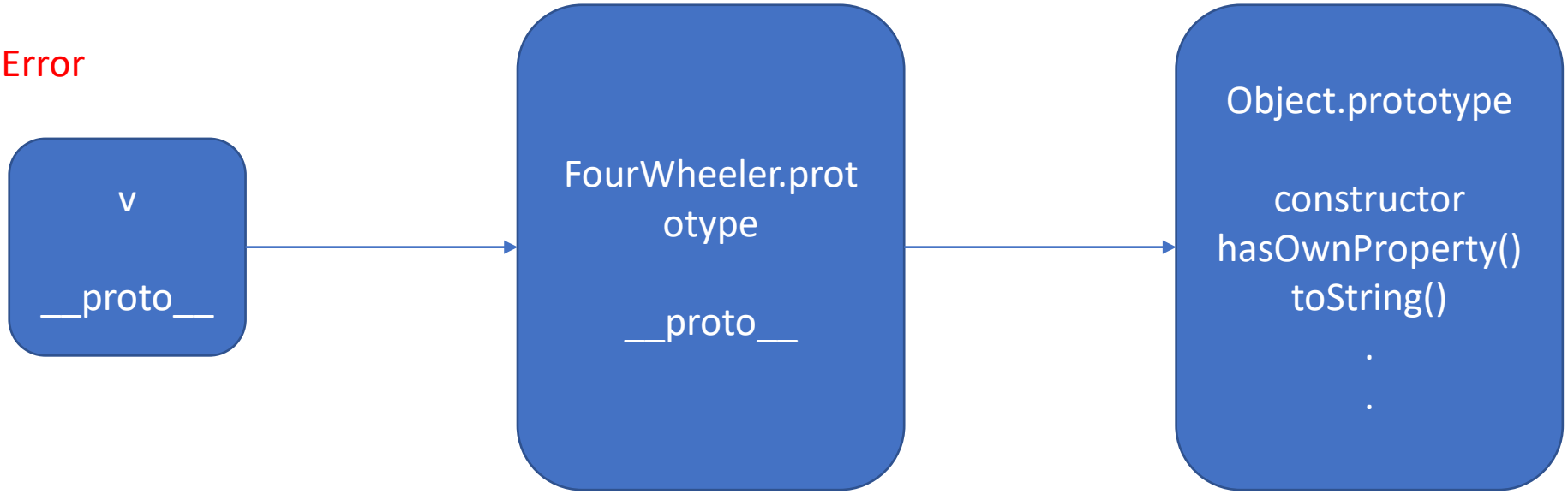
ES6 Static

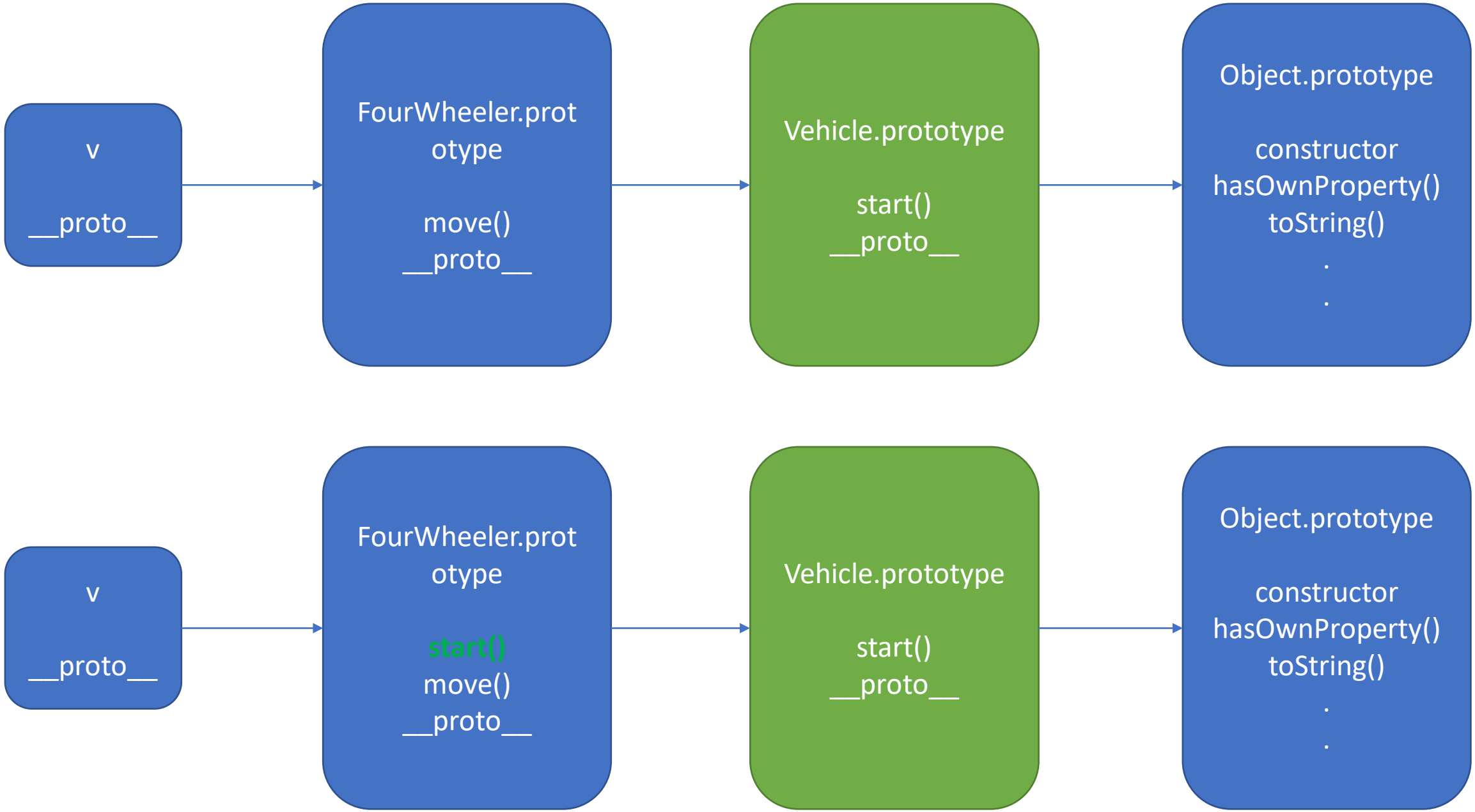
- Static members refer to class-level properties or methods that are shared among all instances of the class.
- These members are associated with the class itself rather than with individual instances of the class.
- Static members are useful for defining properties and methods that are shared among all instances of a class and do not depend on the state of a specific instance.
- They can be used for implementing utility functions, for example, or for defining constants that are used throughout the class.

v.start()



v.start() - Error




v
__proto__
FourWheeler.prototype
move()
__proto__
Vehicle.prototype
start()
__proto__
Object.prototype
constructor
hasOwnProperty()
toString()
.
.
v
__proto__
FourWheeler.prototype
start()
move()
__proto__
Vehicle.prototype
start()
__proto__
Object.prototype
constructor
hasOwnProperty()
toString()
.
.

Indexed collections

- Array
- Int8Array
- Uint8Array
- Uint8ClampedArray
- Int16Array
- Uint16Array
- Int32Array
- Uint32Array
- Float32Array
- Float64Array

Arrays

- **Arrays** are an ordered collection that can hold data of any type. In JavaScript, arrays are created with square brackets [...] or using Array constructor and allow duplicate elements.
- JavaScript objects and arrays were the most important data structures to handle collections of data.
- The combination of objects and arrays was able to handle data in many scenarios, however, there were a few shortcomings as follows
  - Object keys can only be of type string.
  - Objects don't maintain the order of the elements inserted into them.
  - Objects lack some useful methods, which makes them difficult to use in some situations. For example, you can't compute the size (length) of an object easily. Also, enumerating an object is not that straightforward.
  - Arrays are collections of elements that allow duplicates. Supporting arrays that only have distinct elements requires extra logic and code.

Keyed collections

- Map
- Set
- WeakMap
- WeakSet

Map

- Map is a collection of key-value pairs where the key can be of any type. Map remembers the original order in which the elements were added to it, which means data can be retrieved in the same order it was inserted in.
- In other words, Map has characteristics of both Object and Array:
  - Like an object, it supports the key-value pair structure.
  - Like an array, it remembers the insertion order.
- A new Map can be created using Map Constructor.
- Use Map when:
  - You may want to create keys that are non-strings. Storing an object as a key is a very powerful approach. Map gives you this ability by default.
  - You need a data structure where elements can be ordered. Regular objects do not maintain the order of their entries.
  - You are looking for flexibility without relying on an external library like lodash.

Set

- A Set is a collection of unique elements that can be of any type.
- Set is also an ordered collection of elements, which means that elements will be retrieved in the same order that they were inserted in.
- A Set in JavaScript behaves the same way as a mathematical set.
- Use Set when
  - You need to maintain a distinct set of data to perform set operations like union, intersection, difference, and so on.

Weak Map

- The WeakMap object is a collection of key/value pairs in which the keys are **objects only** and the values can be arbitrary values.
- The object references in the keys are held weakly, meaning that they are a target of garbage collection (GC) if there is no other reference to the object anymore.
- One difference to Map objects is that WeakMap keys are not enumerable.
- One use case of WeakMap objects is to store private data for an object, or to hide implementation details.

Weak Set

- WeakSet objects are collections of objects.
- An object in the WeakSet may only occur once.
- It is unique in the WeakSet's collection, and objects are not enumerable.
- The main differences to the Set object are:
  - In contrast to Sets, WeakSets are collections of objects only, and not of arbitrary values of any type.
  - The WeakSet is weak: References to objects in the collection are held weakly. If there is no other reference to an object stored in the WeakSet, they can be garbage collected. That also means that there is no list of current objects stored in the collection.
  - WeakSets are not enumerable.
- The use cases of WeakSet objects are limited.
  - They will not leak memory, so it can be safe to use DOM elements as a key and mark them for tracking purposes.

Well-Known symbols

- A list of symbols that JavaScript treats specially is published as well-known symbols.
- Well-known symbols are used by built-in JavaScript algorithms. For example **Symbol.iterator (@@iterator)** is utilized to iterate over items in arrays, strings, or even to define your own iterator function.

Well-known symbol

Specification Name	[[Description]]	Value and Purpose
@@hasInstance	"Symbol.hasInstance"	A method that determines if a constructor object recognizes an object as one of the constructor's instances. Called by the semantics of the instanceof operator.
@@isConcatSpreadable	"Symbol.isConcatSpreadable"	A Boolean valued property that if true indicates that an object should be flattened to its array elements by Array.prototype.concat.
@@iterator	"Symbol.iterator"	A method that returns the default Iterator for an object. Called by the semantics of the for-of statement.
@@match	"Symbol.match"	A regular expression method that matches the regular expression against a string. Called by the String.prototype.match method.
@@replace	"Symbol.replace"	A regular expression method that replaces matched substrings of a string. Called by the String.prototype.replace method.
@@search	"Symbol.search"	A regular expression method that returns the index within a string that matches the regular expression. Called by the String.prototype.search method.
@@species	"Symbol.species"	A function valued property that is the constructor function that is used to create derived objects.
@@split	"Symbol.split"	A regular expression method that splits a string at the indices that match the regular expression. Called by the String.prototype.split method.
@@toPrimitive	"Symbol.toPrimitive"	A method that converts an object to a corresponding primitive value. Called by the ToPrimitive abstract operation.
@@toStringTag	"Symbol.toStringTag"	A String valued property that is used in the creation of the default string description of an object. Accessed by the built-in method Object.prototype.toString.
@@unscopables	"Symbol.unscopables"	An object valued property whose own property names are property names that are excluded from the with environment bindings of the associated object.

Iterators & Generators

- Iterators and Generators bring the concept of iteration directly into the core language and provide a mechanism for customizing the behavior of for...of loops.
- In JavaScript an iterator is an object which defines a sequence and potentially a return value upon its termination.
- Specifically, an iterator is any object which implements the Iterator protocol by having a next() method that returns an object with two properties:
  - value
    - The next value in the iteration sequence.
  - done
    - This is true if the last value in the sequence has already been consumed. If value is present alongside done, it is the iterator's return value.
- Once created, an iterator object can be iterated explicitly by repeatedly calling next().

Generators

- In JavaScript, generators provide a new way to work with functions and iterators. Generators provide an easier way to implement iterators.
- Using a generator,
  - you can stop the execution of a function from anywhere inside the function
  - and continue executing code from a halted position
- To create a generator, you need to first define a generator function with function* symbol.
- You can pause the execution of a generator function without executing the whole function body by using yield keyword.
- The yield expression returns a value. However, unlike the return statement, it doesn't terminate the program. That's why you can continue executing code from the last yielded position.

Generator Methods

Method	Description
next()	Returns a value of yield
return()	Returns a value and terminates the generator
throw()	Throws an error and terminates the generator

Real-time use cases

- Some real-time use cases of generators:
  - **Stream Processing**:
    - Generators can be used for stream processing where we have a potentially infinite stream of data that we want to process in chunks. By using a generator function to produce chunks of data, we can process the stream one chunk at a time, rather than loading the entire stream into memory.
  - **Asynchronous Operations**:
    - Generators can also be used to perform asynchronous operations in a synchronous-looking code. By using the yield keyword inside the generator function, we can pause the execution of the generator and wait for an asynchronous operation to complete before resuming execution.
  - **Memory Optimization**:
    - Generators can help optimize memory usage in scenarios where we only need a small portion of a potentially large data set at any given time. By using a generator function to generate only the required portion of the data on-demand, we can avoid loading the entire data set into memory.
  - **Iterative Algorithms**:
    - Generators can also be used in iterative algorithms where we need to generate a sequence of values that depends on the previous value. By using a generator function to generate the next value in the sequence on-demand, we can create an iterative algorithm that is more efficient and memory-friendly.

Assignment - Generator

- Use a generator function to generate random passwords of the specified length.

Modules

- As our application grows bigger, we want to split it into multiple files, so called “modules”.
- Until ES6, JavaScript existed without a language-level module syntax.
- JavaScript community invented a variety of ways to organize code into modules, special libraries to load modules on demand.
  - AMD – one of the most ancient module systems, initially implemented by the library require.js.
  - CommonJS – the module system created for Node.js server.
  - UMD – one more module system, suggested as a universal one, compatible with AMD and CommonJS.
- The language-level module system appeared in the standard in ECMASCRIPT 2015

Modules

- A module is just a file. One script is one module. As simple as that.
- A module may contain a class or a library of functions for a specific purpose.
- Modules can load each other and use special directives export and import to interchange functionality, call functions of one module from another one:
  - **export** keyword labels variables and functions that should be accessible from outside the current module.
  - **import** allows the import of functionality from other modules.

Promise

- The Promise object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value.
- It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.
- This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future.

Fetch API

- The Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses.
- It also provides a global fetch() method that provides an easy, logical way to fetch resources asynchronously across the network.
- This kind of functionality was previously achieved using XMLHttpRequest.
- Fetch provides a better alternative that can be easily used by other technologies such as Service Workers.
- Fetch also provides a single logical place to define other HTTP-related concepts such as CORS and extensions to HTTP.

Assignment - Promise Chaining

- Do Calls to the following URL and get data to display the user details with all the posts
  - https://jsonplaceholder.typicode.com/users/1
  - https://jsonplaceholder.typicode.com/users/1/posts

Beyond ES6

- ES 7 (ECMAScript 2016) New Features
  - Exponentiation Operator
  - Array: includes()
- ES 8 (ECMAScript 2017) New Features
  - Object Static Methods
    - Object: entries() & values()
    - Object: getOwnPropertyDescriptors()
  - String Padding
  - Trailing commas in Function
  - Async Functions
    - Async Await Flow
    - Async Await
- ES 9 (ECMAScript 2018) New Features
  - Object rest/spread properties
  - Asynchronous Iteration
  - Promise: finally()
  - Template literal Revision

What’s next?

- Learn the features in:
  - ECMAScript 2019
  - ECMAScript 2020
  - ECMAScript 2021
  - ECMAScript 2022
  - ECMAScript 2023
- Learn HTML 5 Programming (HTML 5 API’s)
  - Fetch
  - Fetch-Interceptor
  - Web Storage API
  - Video API
  - All Other API’s also

Contact Me

Manish Sharma | Gmail

Manish Sharma | WhatsApp

Manish Sharma | Facebook

Manish Sharma | LinkedIn