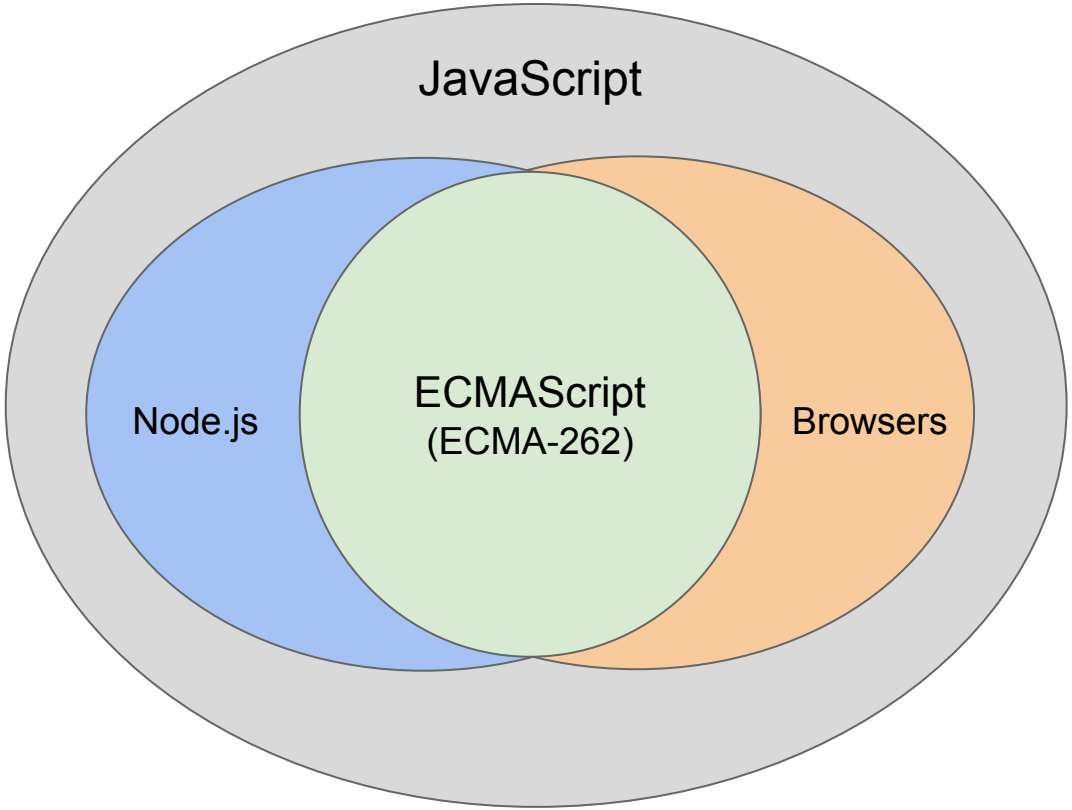
ES6

What is ECMAScript



coding avenue

Outline

- Block bindings
- Template literals
- Functions
  - Default parameters
  - Rest parameters
  - Spread operator
  - Arrow functions
- Objects
  - Property initializer shorthand
  - Concise methods
- Destructuring
- Classes

coding avenue

Block Bindings

var & Block-Level Declarations

- Bindings using *var* are treated as if they're declared at the top of the function or global scope.
- Block-level declarations - Variables are inaccessible outside of a given block scope.
- Block scopes are created inside a block enclosed by curly braces, { }.
- *let*
  - Similar to *var* but limits its scope to its enclosing block.
- *const*
  - Declares *constants*, i.e. their values cannot change once set.
  - Must be initialized when being declared.
- Already-defined variables cannot be redeclared with *let* or *const.*
- Best Practice: use *const* by default, and only use *let* when values need to change.

coding avenue

Strings

Template Literals

- Allows for:
  - Multiline strings
  - String substitution
- Wrapped in backticks, ``, instead of single or double quotes.
- Substitutions are delimited by an opening *${* and closing *}*.

coding avenue

Functions

Default Parameters

- In ES6, parameters can be initialized by providing default values.
- Initialization using default values happens when parameters are not formally passed with values.

coding avenue

Functions

Rest Parameters

- Named parameter that becomes an array containing the “rest of the parameters” passed to a function.
- Indicated by three dots (...) preceding the name of the parameter.
- There can only be one rest parameter and it should always be the last.

coding avenue

Functions

Spread Operator

- Allows for passing an array that will be split with its items as separate arguments to a function.
- Similar to rest parameters, it is indicated by three dots (...) preceding the name of the parameter.
- Unlike rest parameters, there can be several spread operators in a function's parameter declaration.

coding avenue

Functions

Arrow Functions

- Concise way of writing anonymous functions, e.g. event handlers or callbacks.
- Syntax uses an “arrow” (=>).
- Behaves differently from traditional Javascript functions:
  - Value of *this* binding is the closest non-arrow function.
  - Cannot be called with *new,* thus cannot be used as constructors.
  - No *arguments* object - rely on named and rest parameters to access function arguments

coding avenue

Objects

Property Initializer Shorthand

- Eliminates duplication in property names and local variables.
- When a property name is the same as the local variable, one can omit the colon and value part.

Objects

Concise Methods

- In ES6, syntax for writing object methods is made more concise.
- Done by eliminating the colon and function keyword.

Destructuring

For Easier Data Access

- Destructuring - process of breaking down a data structure into smaller parts.
- Applies to both objects and arrays.
- In ES5 and earlier, several lines of code may be needed to assign object properties or array values into local variables

coding avenue

Destructuring

Object Destructuring

- Uses an object literal, “{}”, on the left side of the assignment operation.
- When using with *var, let, const*, an initializer must be supplied in the right side of the operation.
- Default values can be assigned to local variables when a property name does not exist on an object.

coding avenue

Destructuring

Array Destructuring

- Uses an array literal, “[]”, on the left side of the assignment operation.
- Destructuring operates on positions within an array.
- When using with *var, let, const*, an initializer must be supplied in the right side of the operation.
- Default values can also be assigned to local variables.

coding avenue

Classes

Class Declarations

- In ES5, closest equivalent was to create a constructor and assign methods to the constructor's prototype.
- Class declarations begin with the *class* keyword followed by the class name.
- Rest of the class syntax looks like concise methods in object literals.
- *First-class citizens*, i.e. these can be passed into a function, returned from a function or assigned to a variable.

Classes

Accessor Properties

- To create a *getter*, use the *get* keyword followed by a space and identifier.
- To create a *setter*, do the same but using the *set* keyword.

Classes

Static Members

- Use *static* keyword on any method or accessor property.
- Cannot use *static* with the constructor method.
- Not accessible from instances.

Classes

Inheritance

- Classes that inherit from other classes are called *derived classes*.
- Use *extends* keyword to specify from which parent class to inherit.
- Derived classes requires *super()* if a constructor is specified.
- *super()* must be called before accessing *this* in the constructor.

coding avenue

ES6 Support

- Node.js
  - 99% ES6 support since v6.9.2 (http://node.green)
- Browsers
  - IE 11 - 11%
  - Latest Edge, FF, Chrome, Safari - ~92%
  - http://kangax.github.io/compat-table/es6/
  - Babel - transforms ES6 to ES5
    - Needs *npm* and a *task runner* or *bundler*, e.g. grunt, webpack, etc.

coding avenue

That's All. Thanks!

coding avenue