

# Array Methods

- **Example:** ArrayMethods\*.html

# Set Methods

- **Example:** SetMethods.html

# Immediately Invoked Function Expressions

- **Example:** ImmediatelyInvokedFunctionExpression.html

# Function Context

- Problem
  - **Example:** FunctionContextIncorrect.html
- How to address the problem
  - **Example:** FunctionContextCorrect/a/b/c.html

# Event Propagation

- **Example:** EventPropagation.html, EventPropagationControlled.html, AccessingElementEventOcurrred.html

# Features

- In E6 (ECMAScript6) the key of a property can be a string or a symbol
- Computed Property Keys
  - To specify the key of a property we can use a fixed name (e.g., `student.name = "Mary"`)
  - We can also use an expression in square brackets (computed property key approach)
    - **Example:** `ComputedPropKeys.html`
  - Definition of methods without using **function**
  - Main use case – symbols
    - Define a symbol
    - Use it as a property key that is unique

# Features

- `Object.assign(target, src1, src2, ...)`
  - Combines enumerable own (non-inherited) properties of sources into the target
  - Returns target
  - **Example:** `ObjectMethods.html`
  - Can also be used to assign methods
- `Object.is()`
  - Provides a more precise comparison alternative to `===`
  - **Example:** `ObjectMethods.html`

# JavaScript Classes

- E6 Classes provide a convenient syntax to define constructor functions
- Defined using the **class** construct
- A class body cannot contain properties (only methods)
- Use **static** to define class methods, no static data members
- Relies on **constructor** to define a constructor (not the class name)
- A class, unlike a function, a class constructor cannot be invoked unless you use **new**
  - Student() call (assuming Student is a class will not work)
- No need to use semicolons to separate method definitions. Using of commas is forbidden
- Unlike functions, class declarations are not hoisted
  - Class definitions must be seen before using the class
- **Example:** ClassDefinitionDeclaration.html
- Previous example relies on class declaration approach for class definition. Class expression is a second approach
- **Example:** ClassDefinitionExpression.html



# JavaScript Classes

- If you don't provide a constructor the following definition is used:
  - `constructor() {}`
- You can define a subclass using **extends**
- Only **single inheritance** is supported
- **Example:** Subclass.html
- If you don't provide a constructor in a derived class the following constructor is used:
  - ```
Constructor(..arguments) {  
    super(...arguments)  
}
```
- Private data
  - Can follow convention of using `_` for instance variables
    - Will not actually protect the data
  - Using WeakMaps
  - Other approaches available
- **Example:** Private.html

# Iterators

- Objects with interface designed for iteration
- An iterator object has:
  - **next()** method – returns a **result** object
  - result object – has two properties
    - **value** – next value
    - **done** – true when there are no more values
  - Keeps a pointer

# Generators

- Generator - function that returns an iterator
- Code that can be paused and resume
- Relies on **function\*** to define a generator function
- **yield** operator – allows the generator to pause
  - Generators can receive input/output using yield
  - yield can only be used within generators
    - yield cannot cross functions (e.g., yield cannot appear inside of a function that is inside of the generator function)
- **next** method – resumes execution
- The generator function returns a generator object
- Generators implement the interface Iterable
  - Can be used by constructs that support iterables (e.g., for-of)
  - **Example:** Generator1.html, Generator2.html, GeneratorInClass.html
  - You can have function expressions that create generators
    - `let mylterator = function *() { ... };`

# Iterable

- Iterable- object with a Symbol.iterator property
- Symbol.iterator specifies function that returns an iterator for the object
- In ECMAScript 6 the following are iterables
  - Arrays
  - Sets
  - Maps
  - Strings
- Iterables have been design to work with for-of loops
  - for-of calls next() on each loop execution
  - Loop stops when object's done property is true
  - **Example:** Iterable1.html
- for-of throws error on non-iterable object
- You can add iterator to custom types
  - **Example:** Iterable2.html

# Collection Iterators

- E6 has three collection types: arrays, sets, and maps. All are associated with the following built-in iterators
  - **keys()** → iterator for the keys
  - **values()** → iterator for the values
  - **entries()** → iterator for key/value pairs returned as a two-element array
- Default iterator used by for-of
  - Arrays and set → values()
  - Maps → entries()
- WeakMaps and Weak do not have built-in iterators

# Iterables and Spread Operator

- Spread operator (...) works on all iterables using the default iterator to determine the elements to insert into the array
- Direct approach to covert iterable into array

# References

- <http://exploringjs.com/es6/>
- **Understanding ECMAScript 6** - ISBN-13: 978-1-59327-757-4