# ECMAScript 6

ECMAScript 6, also known as ECMAScript 2015, is the latest version of the ECMAScript standard. ES6 is a significant update to the language, and the first update to the language since ES5 was standardized in 2009. Implementation of these features in major JavaScript engines is underway now. ES6 includes the following new features:

- arrows
- classes
- enhanced object literals
- template strings
- destructuring
- default + rest + spread
- let + const
- iterators + for..of
- generators
- Unicode
- modules
- module loaders
- map + set + weakmap + weakset
- proxies
- symbols
- sub classable built-ins
- promises
- math + number + string + array + object APIs
- binary and octal literals
- reflect api
- tail calls

# Shorthand Functions

**Shorthand method definition**

**Shorthand method definition** can be used in a method declaration on object literals and ES6 classes. You can define them using a function name, followed by a list of parameters in a pair of parenthesis (para1, ..., paramN) and a pair of curly braces { ... } that delimits the body statements.

The following example uses shorthand method definition in an object literal:

```
const collection = {
  items: [],
  add(...items) {
    this.items.push(...items); },
  get(index) {
    return this.items[index];  } };
collection.add('C', 'Java', 'PHP');
collection.get(1) // => 'Java'
```

add() and get() methods in collection object are defined using short method definition. These methods are called as usual: collection.add(...) and collection.get(...).

The short approach of method definition has several benefits over traditional property definition with a name, colon : and a function expression add: function(...) {...}:

- A shorter syntax is easier to read and write
- Shorthand method definition creates named functions, contrary to a function expression. It is useful for debugging.

Notice that class syntax requires method declarations in a short form:

```
class Star {
  constructor(name) {
    this.name = name;  }
  getMessage(message) {
    return this.name + message;   } }
const sun = new Star('Sun');
sun.getMessage(' is shining') // => 'Sun is shining'
```

# JavaScript subset and extensions

Subsets are mostly defined for security purposes; scripts written using secure language subsets can be executed safely even if its source is untrusted, for instance, an ad server. Some of these subsets will be described later.

As JavaScript continued to evolve and allowed explicit extensions, newer versions were released. Many of the features were standardized. These extensions are compatible with modern browsers such as Firefox and Chrome. However, the implementation of non-standard extensions may require an external compiler because these features are being updated in major JavaScript engines now.