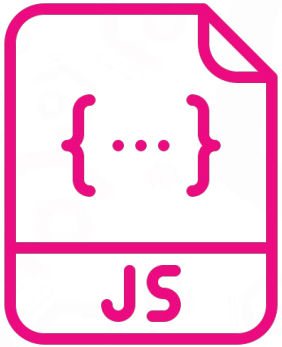
CodingLab
Powered by RBK

Junior phase



ES6 Syntax

An introduction to ES6, 7, 8 and beyond

Duration: 30 minutes
Q&A: 5 minutes by the end of the lecture

What is ES6?

ES6 is a recent specification change, or widely agreed-upon standard**,** of the JavaScript Language as specified by TC39 working group. It adds a host of new syntax and features.
ES6 is officially known as ECMAScript 2015. You will find many helpful resources online which use either name.

ES7, 8, 9 and Beyond

Since the introduction of ES6, the TC39 working group has released annual updates to the language, known as:

- ES 2016
- ES 2017
- ES 2018
- ES 2019 ...

ES6 contains big changes to the language due to the gap from prior releases (ES5 was released in 2009, and ES3 in 1999). Since ES6, annual changes have been much smaller.

What we'll cover

- Destructuring
- Spread Operator
- Default Parameters
- Template Literals
- Arrow Functions
- for . . . of lopp

Destructuring (with Objects)

ES5 Syntax	ES6 Syntax

var user = {
first : ' Benji ' ,
last : ' Marianacci ' ,
age : 32
};
 	
var first = user.first;
var last = user.last;
 	
first; / / => ' Benji '
	var user = {
	first : ' Benji ' ,
	last : ' Marinacci ',
	age : 32
	};
	
	var { first , last } = user ;
	
	first; / / => ' Benji '

Destructuring is used to retrieve values from objects using a special syntax that resembles object literals. In object destructuring, variable names are used for key lookup in the source object, assigning values found at those keys.

Destructuring (with Arrays)

ES5 Syntax	ES6 Syntax

`var numbers = [1, 2, 3, 4 ];`
 	
`var uno = numbers[ 0 ];`
`var dos = numbers[ 1 ];`
`var tres = numbers[ 2 ];`
 	
`uno; / / =>  1`
	`var numbers = [1, 2, 3, 4 ];`
	
	`var [ uno, dos, tres  ] = numbers ;`
	
	`uno; / / => 1`

A similar technique can be used for arrays, but instead of matching the names of the variables to their corresponding keys, JavaScript matches the ordinal position of the variable to the same position in the array.

Destructuring (with Arrays)

ES5 Syntax	ES6 Syntax

`var numbers = [1, 2, 3, 4 ];`
 	
`var countToTwo = function ( numbers )`
`{`
`  var uno = numbers[ 0 ] ;`
`  var dos = numbers[ 1 ] ;`
 	
`console.log( uno + ‘ ,  ‘ + dos );`
 	
`};`
 	
`countToTwo (numbers ) ; / / =>  1, 2`
	`var numbers = [1, 2, 3, 4 ];`
	
	`var countToTwo = function ( [ uno, dos ] ) {`
	
	`console.log( uno + ‘ ,  ‘ + dos );`
	
	`};`
	
	`countToTwo (numbers ) ; / / =>  1, 2`

A similar technique can be used for arrays, but instead of matching the names of the variables to their corresponding keys, JavaScript matches the ordinal position of the variable to the same position in the array.

Destructuring (with Arrays)

ES5 Syntax	ES6 Syntax

`var numbers = [1, 2, 3, 4 ];`
 	
`var countToTwo = function ( numbers )`
`{`
`  var uno = numbers[ 0 ] ;`
`  var dos = numbers[ 1 ] ;`
 	
`console.log( uno + ‘ ,  ‘ + dos );`
 	
`};`
 	
`countToTwo (numbers ) ; / / =>  1, 2`
	`var numbers = [1, 2, 3, 4 ];`
	
	`var countToTwo = function ( [ uno, dos ] ) {`
	
	`console.log( uno + ‘ ,  ‘ + dos );`
	
	`};`
	
	`countToTwo (numbers ) ; / / =>  1, 2`

The destructuring technique is far more useful when passing values into functions...

Destructuring (with Objects)

ES5 Syntax

```
 var user = {
      first : ‘ Benji ‘,
      last : ‘ Marinacci ‘ ,
      age : 32
};
var greet = function (person) {
    var first = person.first ;
    var last = person.last ;

 console.log( ‘ Hi , ‘ + first + ‘  ‘
+ last);
};

greet (user) ; / / => ‘ Hi, benji
Marianacci’
```

ES6 Syntax

```
var user = {
      first : ‘ Benji ‘,
      last : ‘ Marinacci ‘ ,
      age : 32
};
var greet = function ({ first, last })
{

 console.log( ‘ Hi , ‘ + first + ‘  ‘
+ last);
};



greet (user) ; / / => ‘ Hi, benji
Marianacci’
```

… with object destructuring being the most common use case of them all.

Spread Operator

ES5 Syntax	ES6 Syntax

var someFunction (a, b, c, d) {
/ / do something with a, b, c, and d
};
 	
var args = [1, 2, 3, 16];
 	
someFunction.apply (null, args);
	var someFunction (a, b, c, d) {
	/ / do something with a, b, c, and d
	};
	
	var args = [1, 2, 3, 16];
	
	someFunction.apply (. . .args) ;

the **spread operator** allows you to expand multiple elements of an array. This is useful when you need to pass an array of values as an argument to a function which does not accept arrays.

Spread Operator

ES5 Syntax

```
var multiplyByNum = function (x) {
     var nums = Array.prototype.slice.call (arguments,
1);
     nums.forEach (function (num) {
     console.log (x * num);
     }
};
```

ES6 Syntax

```
var multiplyByNum = function (x, . . .nums) {
    nums.forEach ( function (num) {
    console.log (x * num) ;
    });
}
```

Rest parameters allow you to capture any numbers of unnamed function parameters into an array. This reduces the complexity involved in manipulating the arguments pseudo-array.

Default parameters

ES5 Syntax

```
var generateAddress = function (city, state, country) {
   country = country === undefined ?  ‘ USA ‘ : country ;
   return city + ‘ ,  ‘ +  state.toUpperCase ( ) + ‘ ,  ‘  +
country ;
} ;
generateAddress( ‘ Oakland ’, ‘ CA ‘ ) ; / / => ‘ Oakland,
CA, USA ‘
generateAddress( ‘ Calgary ‘ , ‘ AB ‘, ‘ Canada ‘ ) ; / / =>
‘ Calgary, AB, Canada ‘
```

ES6 Syntax

```
var generateAddress = function (city, state, country = ‘ USA
‘) {
   return city + ‘ ,  ‘ +  state.toUpperCase ( ) + ‘ ,  ‘  +
country ;
} ;
generateAddress( ‘ Oakland ’, ‘ CA ‘ ) ; / / => ‘ Oakland,
CA, USA ‘
generateAddress( ‘ Calgary ‘ , ‘ AB ‘, ‘ Canada ‘ ) ; / / =>
‘ Calgary, AB, Canada ‘
```

ES6 support supplying **default values** for function parameters within the function signature.

Template literals

ES5 Syntax

```
var generateAddress = function (city, state, country)
{
   country = country === undefined ?  ‘ USA ‘ :
country ;
   return city + ‘ ,  ‘ +  state.toUpperCase ( ) + ‘ ,
‘  +  country ;
} ;
```

ES6 Syntax

```
var generateAddress = function (city, state, country =
‘ USA ‘) {
   return city + ‘ ,  ‘ +  state.toUpperCase ( ) + ‘ ,
‘  +  country ;
} ;
```

Template literals simplify working with strings that include references to variables. They’re also known by the name “template strings.” Instead of concatenating strings using the + operator . . .

Template literals

ES5 Syntax

```
var generateAddress = function (city, state, country)
{
   country = country === undefined ?  ‘ USA ‘ :
country ;
   return city + ‘ ,  ‘ +  state.toUpperCase ( ) + ‘ ,
‘  +  country ;
} ;
```

ES6 Syntax

```
var generateAddress = function (city, state, country =
‘ USA ‘) {
   return ` ${city} + ‘ ,  ‘ + ${state.toUpperCase ( )
} + ‘ ,  ‘  + ${country} `;
} ;
```

. . . we can embed variables directly into strings using the ${ } operator. It’s important to note . . .

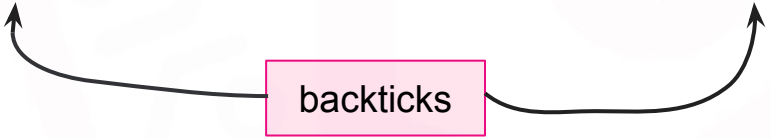
Template literals

ES5 Syntax

```
var generateAddress = function (city, state, country)
{
   country = country === undefined ?  ‘ USA ‘ :
country ;
   return city + ‘ ,  ‘ +  state.toUpperCase ( ) + ‘ ,
‘  +  country ;
} ;
```

ES6 Syntax

```
var generateAddress = function (city, state, country = ‘ USA ‘) {
  return ` ${city} + ‘ , ‘ + ${state.toUpperCase ( ) } + ‘ , ‘ + ${country} ` ;
}
```



. . . that template literals are denoted by backticks, and not single- or double-quotes.

Template literals (multiple lines)

ES5 Syntax	ES6 Syntax
var html = [‘ ’ , ‘ ‘ , message , ‘ ’ , ‘ ‘].join (‘\n’) ;	var html = ` <div> <span> { message } </span> </div> ` ;

Template literals improve the experience of working with strings which contain multiple lines or whitespace.

Arrow Function Expressions

ES5 Syntax	ES6 Syntax

`var multiplier = function ( x, y ) {`
`  return x * y ;`
`}`
	`var multiplier = ( x, y ) => {`
	`  return  x * y ;`
	`}`

Arrow functions offer a new syntax for writing functions expressions. Instead of using the function keyword before defining some parameters, you can use a “ fat arrow “ after the parameters.

Arrow Function Expressions

ES5 Syntax	ES6 Syntax

`var multiplier = function ( x, y ) {`
`  return x * y ;`
`}`
	`var multiplier = ( x, y ) => {`
	`  return  x * y ;`
	`}`
	
	`// also works :`
	
	`var multiplier = (x, y) => x * y`

If the function body contains only one expression, you may omit the curly braces and the return keyword, writing the entire function on one line.

Arrow Function Expressions

ES5 Syntax	ES6 Syntax

`var doubler = function ( num ) {`
`  return 2 * num ;`
`}`
	`var doubler =  num  =>  2 * num`

If a function expects exactly one named parameter, you may omit the parentheses around that parameter.

Arrow Function Expressions

ES5 Syntax

```
var doubleValues = function ( values ) {
   return values.map( function (value) {
   return value * 2 ;
   });
};
```

ES6 Syntax

```
var doubleValues = function ( values ) {
      return values.map(value => value * 2 ) ;
} ;
```

This abbreviated function syntax works nicely with higher order functions.

Arrow Function Expressions

ES5 Syntax

```
var doubleValues = function ( values ) {
   return values.map( function (value) {
   return value * 2 ;
   });
};
```

ES6 Syntax

```
var doubleValues = function ( values ) {
      return values.map(value => value * 2 ) ;
} ;
/ / also works :
var doubleValues = values => values.map ( value =>
value * 2 ) ;
```

We can refactor our ES6 version of doubleValues to be even more concise.

For...of loop

ES5 Syntax	ES6 Syntax

`var nums = [ 1, 4, 6, 7 ] ;`
 	
`for ( var i = 0 ; i < nums.length ; i ++ ) {`
`  console.log( nums[ i ] ) ;`
`}`
 	
`var string = ‘ hello world ‘ ;`
 	
`for ( var j = 0 ; j < string.length ; j ++ ) {`
`  console.log( string[ j ] );`
`}`
	`var nums = [ 1, 4, 6, 7 ] ;`
	
	`for ( var val of nums ) {`
	`  console.log( val ) ;`
	`}`
	
	`for ( var letter of ‘ hello world ‘ )`
	`{`
	`  console.log ( letter )`
	`}`

In ES6, some collections (like Arrays, Sets, and Maps) are considered to be “iterable.” You can use the for. . .of loop to iterate over every **value** stored in these kinds of collections.

Further Exploration

There are lots more new features in ES6 for you to explore :

- **Let** and **const**, alternatives to the keyword var
- Built-in **Promises**
- Built-in **module system**

Try ES6

Experimentation is the best way to learn new syntax.

- MDN has a handy guide to learn ES6
- MDN continues to be the best resource for JavaScript documentation
- There are lots more new features in ES6 for you to explore .

That’s it