Telerik Academy

CoffeeScript Overview

Coding JavaScript the expressive way

Telerik Software Academy

Learning & Development

http://academy.telerik.com

Telerik Academy

Table of Contents

- JavaScript-like languages
  - TypeScript & CoffeeScript
- CoffeeScript Overview
  - Installation & Compilation
- CoffeeScript Features
  - Variables and functions, string interpolation
  - Conditionals, Loops and Comprehensions
  - Operations, Aliases, Destructuring assignments
  - Classes & Inheritance

Telerik Academy

JavaScript-like Languages

Languages that compile to JavaScript

Telerik Academy

JavaScript-like Languages

- JavaScript is a not-mean-to-be-first-class-language
  - It was developed by Netscape for updating the UI of a web page
- In result, many languages appeared that compile to JavaScript
  - CoffeeScript and TypeScript are most famous

Telerik Academy

CoffeeScript

Installation and compilation

Installing CoffeeScript

- CoffeeScript has its own core compiler, and can even run inside the browser
  - Yet these are hard and slow
- CoffeeScript can be installed using Node.js:

```
$ npm install -g coffee-script
```

  - And then compiled using:

```
$ coffee --compile scripts.coffee
```

Telerik Academy

Installing and Compiling CoffeeScript

Live Demo

Telerik Academy

Tools for Compiling CoffeeScript

- Since CoffeeScript is widely used most of the Tools for JavaScript development support CoffeeScript as well:
  - Visual Studio – Web Essentials
  - Sublime text – Better CoffeeScript
  - WebStorm – built-in

Telerik Academy

Tools for Compiling CoffeeScript

Live Demo

Telerik Academy

CoffeeScript Features

What can we do with CoffeeScript?

Telerik Academy

CoffeeScript Features

- CoffeeScript is a whole new language to learn
  - Yet, has parts of Python and JavaScript
- Omit semicolons and brackets
  - But the indentation matters a lot!
    - They introduce scope
  - This is a valid CoffeeScript code:

```
CoffeeScript
person =
  name: 'Peter'
  age: 17
console.log person
```

Telerik Academy

CoffeeScript Features

- CoffeeScript is a whole new language to learn
  - Yet, has parts of Python and JavaScript
- Omit semicolons and brackets
  - But the indentation matters a lot!
    - They introduce scope
  - This is a valid CoffeeScript code:

```
CoffeeScript
person =
  name: 'Peter'
  age: 17
console.log person
```

Compiles to

```
JavaScript
var person;
person = {
  name: 'Peter',
  age: 17
};
console.log(person);
```

Telerik Academy

CoffeeScript: Functions

- In CoffeeScript all functions are expressions
  - No function declarations
  - Created like C# LAMBDA expressions:

CoffeeScript

```
sum = (numbers) ->
  sum = 0
  for n in numbers
    sum += n
  return sum
```

Telerik Academy

CoffeeScript: Functions

- In CoffeeScript all functions are expressions
  - No function declarations
  - Created like C# LAMBDA expressions:

CoffeeScript

```
sum = (numbers) ->
  sum = 0
  for n in numbers
    sum += n
  return sum
```

Compiles to

JavaScript

```
var sum;
sum = function(numbers){
  var sum, n, i;
  sum = 0;
  for(i = 0; i < numbers.length; i++){
    n = numbers[i];
    sum += n;
  }
  return sum;
}
```

Telerik Academy

CoffeeScript Features

Live Demo

Telerik Academy

Objects and Arrays

- When creating Objects and/or arrays curly brackets and colons can be omitted
  - Most of the time…

```
CoffeeScript
student =
  names:
    first: 'Peter'
    last: 'Petrov'
  marks: [
    5.5
    6.0
    4.5
  ]
```

Telerik Academy

Objects and Arrays

- When creating Objects and/or arrays curly brackets and colons can be omitted
  - Most of the time…

CoffeeScript

```
student =
  names:
    first: 'Peter'
    last: 'Petrov'
  marks: [
    5.5
    6.0
    4.5
  ]
```

Compiles to

JavaScript

```
var student;
student = {
  names: {
    first: 'Peter',
    last: 'Petrov'
  },
  marks: [5.5, 6.0, 4.5]
};
```

Telerik Academy

Objects and Arrays

Live Demo

Telerik Academy

CoffeeScript Conditional Statements

Telerik Academy

CoffeeScript Conditional Statements

- CoffeeScript is highly expressional language
  - And conditional statements can be done in many ways
    - Sometimes they are compiled to ternary expressions
  - CoffeeScript also adds extensions:
    - `unless` meaning `if not`:

```
unless condition
# same as
if not condition
```

Telerik Academy

CoffeeScript Conditional Statements

- CoffeeScript conditional statements:

```
if condition
  # do stuff
```

Coffee

```
obj = value if condition
```

Coffee

```
goToWork() unless todayIsWeekend()
```

Coffee

Telerik Academy

CoffeeScript Conditional Statements

- CoffeeScript conditional statements:

Coffee

```
if condition
  # do stuff
```

JS

```
if(condition) {
  //do stuff
}
```

Coffee

```
obj = value if condition
```

Coffee

```
goToWork() unless todayIsWeekend()
```

Telerik Academy

CoffeeScript Conditional Statements

- CoffeeScript conditional statements:

Coffee

```
if condition
  # do stuff
```

JS

```
if(condition) {
  //do stuff
}
```

Coffee

```
obj = value if condition
```

JS

```
if(condition) {
  obj = value;
}
```

Coffee

```
goToWork() unless todayIsWeekend()
```

Telerik Academy

CoffeeScript Conditional Statements

- CoffeeScript conditional statements:

Coffee

```
if condition
  # do stuff
```

JS

```
if(condition) {
  //do stuff
}
```

Coffee

```
obj = value if condition
```

JS

```
if(condition) {
  obj = value;
}
```

Coffee

```
goToWork() unless todayIsWeekend()
```

JS

```
if(!(todayIsWeekend())){
  goToWork();
}
```

Telerik Academy

CoffeeScript Conditional Statements

Live Demo

Telerik Academy

Loops and Comprehensions

Regular Loops

- CoffeeScript supports all the regular loops:

```coffee
while condition
  # code
```

```js
while (condition) {
  # code
}
```

```coffee
for item, i in items
  # code
```

```js
for(i = 0;i<items.length;i++){
  item = items[i];
}
```

```coffee
for key, value of items
  item
```

```js
for (key in items) {
  value = items[key];
  item;
}
```

Telerik Academy

Regular Loops

Live Demo

Comprehensions

- Comprehensions in CoffeeScript allows the use of loops as a result
  - Iterate over a collection and return a result

Coffee

```
serialized = (serialize item for item in items)
```

JS

```
serialized = (function() {
  var _i, _len, _results;
  _results = [];
  for (_i = 0, _len = items.length; _i < _len; _i++) {
    item = items[_i];
    _results.push(serialize(item));
  }
  return _results;
})();
```

Telerik Academy

Comprehensions

Live Demo

Telerik Academy

Ranges and Array Slicing

Creating ranges to easify the iteration of arrays

Ranges

- Ranges in Coffee create a array of number values in a given range:

```
numbers = [1..3] -> numbers = [1, 2, 3]
```

```
numbers = [1...3] -> numbers = [1, 2]
```

```
for number in [0..maxNumber] by 2 -> # i =0, 2, 4, …
  console.log "#{i} is an even number"
```

Telerik Academy

Ranges

Live Demo

Array Slicing

- Arrays can be sliced using ranges:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

`result = numbers[2..3]`	`result = [3, 4, 5]`
`result = numbers[..3]`	`result = [1, 2, 3, 4, 5]`
`result = numbers[6..]`	`result = [7, 8, 9]`
`result = numbers[..]`	`result = clone of numbers`
`numbers[1..2] = [-2, -3]`	`result = [1, -2, -3, 4, …]`

Telerik Academy

Array Slicing

Live Demo

Telerik Academy

Operators and Aliases

Telerik Academy

Operators and Aliases

- CoffeeScript aims to be as expressional as possible
  - So many of the operators have aliases:

`is`	compiles to	`===`
`isnt`	compiles to	`!==`
`not`	compiles to	`!`
`and`	compiles to	`&&`
`or`	compiles to	`||`
`true/yes/on`	compiles to	`true`
`false/no/off`	compiles to	`false`
`@, this`	compiles to	`this`
`of`	compiles to	`in`
`x ** n`	compiles to	`Math.Pow(x, n)`
`x // y`	compiles to	`Math.floor(x / y)`

Telerik Academy

Operators and Aliases

Live Demo

Telerik Academy

Classes and Inheritance

Telerik Academy

Classes in CoffeeScript

- CoffeeScript provides a way to create Functional/Classical OOP

```
class Shape
  constructor: (x, y) ->
    @x = x
    @y = y
  setX: (newX) ->
    @x = x
  getX: () ->
    @x
```

```
var Shape;
Shape = (function() {
  function Shape(x, y) {
    this.x = x;
    this.y = y;
  }
  Shape.prototype.setX=function(newX){
    return this.x = x;
  };
  Shape.prototype.getX = function() {
    return this.x;
  };
  return Shape;
})();
```

Telerik Academy

Classes in CoffeeScript

- CoffeeScript provides a way to create Functional/Classical OOP

```
class Shape
  constructor: (x, y) ->
    @x = x
    @y = y
  setX: (newX) ->
    @x = x
  getX: () ->
    @x
```

The constructor is compiled into function constructor

```
var Shape;
Shape = (function() {
  function Shape(x, y) {
    this.x = x;
    this.y = y;
  }
  Shape.prototype.setX=function(newX){
    return this.x = x;
  };
  Shape.prototype.getX = function() {
    return this.x;
  };
  return Shape;
})();
```

Telerik Academy

Classes in CoffeeScript

- CoffeeScript provides a way to create Functional/Classical OOP

```
class Shape
  constructor: (x, y) ->
    @x = x
    @y = y
  setX: (newX) ->
    @x = x
  getX: () ->
    @x
```

Methods are attached to the prototype

```
var Shape;
Shape = (function() {
  function Shape(x, y) {
    this.x = x;
    this.y = y;
  }
  Shape.prototype.setX=function(newX){
    return this.x = x;
  };
  Shape.prototype.getX = function() {
    return this.x;
  };
  return Shape;
})();
```

Telerik Academy

Creating Classes

Live Demo

Inheritance

- CoffeeScript supports also inheritance in a Classical OOP way
  - Using the keyword `extends`
  - Providing a `super()` method to call from parent

```
class Shape
  constructor: (x, y) ->
  setX
  getX: () ->
class Rect extends Shape
  constructor: (x, y, w, h) ->
    super x, y
    @width = w
    @height = h
```

Telerik Academy

Inheritance in CoffeeScript

Live Demo

Telerik Academy

String Interpolation

Putting executable code inside a string

String Interpolation

- String interpolation allows to execute script and put the result inside a string
  - Use double quotes (" ") and `#{script}`

```
class Person
  constructor: (@fname, @lname) ->
  introduce: () -> """Hi! I am #{@fname}
                  #{@lname}."""
```

```
Person = (function() {
  …
  Person.prototype.introduce = function() {
    return "Hi! I am " + this.fname + " " + this.lname + ".";
  };
  return Person;
})();
```

String Interpolation

- String interpolation allows to execute script and put the result inside a string
  - Use double quotes (" ") and `#{script}`

```
class Person
  constructor: (@fname, @lname) ->
  introduce: () -> """Hi! I am #{@fname}
                   #{@lname}."""
```

Interpolate @fname and @lname

```
Person = (function() {
  …
  Person.prototype.introduce = function() {
    return "Hi! I am " + this.fname + " " + this.lname + ".";
  };
  return Person;
})();
```

Telerik Academy

String Interpolation

- String interpolation allows to execute script and put the result inside a string
  - Use double quotes (" ") and `#{script}`

```
class Person
  constructor: (@fname, @lname) ->
  introduce: () -> """Hi! I am #{@fname}
                  #{@lname}."""
```

Triple quotes create strings on multiple lines

```
Person = (function() {
  …
  Person.prototype.introduce = function() {
    return "Hi! I am " + this.fname + " " + this.lname + ".";
  };
  return Person;
})();
```

Telerik Academy

String Interpolation

Live Demo

Telerik Academy

Using switch/when/else

Using switch/when/else

- CoffeeScript introduces a more expressive way for creating switch-case blocks

```
switch day
  when 'Mon' then console.log 'The week starts'
  when 'Tue' then console.log '''Still accommodating to the
                                 start of the week'''
  when 'Wed' then console.log 'Time to work!'
  when 'Thu' then console.log 'Well… end of the week is near'
  when 'Fri' then console.log 'Day of the master.'
  when 'Sat', 'Sun' then console.log 'Relax'
  else console.log 'Wrong day…'
```

Telerik Academy

Using switch/when/else

- CoffeeScript introduces a more expressive way for creating switch-case blocks

```
switch day
  when 'Mon' then console.log 'The week starts'
  when 'Tue' then console.log '''Still accommodating to the
                                 start of the week'''
  when 'Wed' then console.log 'Time to work!'
  when 'Thu' then console.log 'Well… end of the week is near'
  when 'Fri' then console.log 'Day of the master.'
  when 'Sat', 'Sun' then console.log 'Relax'
  else console.log 'Wrong day…'
```

`else` in CoffeeScript
is like
`default` in Javascript

Telerik Academy

Using switch/when/else

- CoffeeScript introduces a more expressive way for creating switch-case blocks

```
switch day
  when 'Mon' then console.log 'The week starts'
  when 'Tue' then console.log '''Still accommodating to the
                                 start of the week'''
  when 'Wed' then console.log 'Time to work!'
  when 'Thu' then console.log 'Well… end of the week is near'
  when 'Fri' then console.log 'Day of the master.'
  when 'Sat', 'Sun' then console.log 'Relax'
  else console.log 'Wrong day…'
```

- And they can be used as expressions:

```
mark = switch
  when score < 100 then 'Failed'
  when score < 200 then 'Success'
  else 'Excellent!'
```

Using switch/when/else

- CoffeeScript introduces a more expressive way for creating switch-case blocks

```
switch day
  when 'Mon' then console.log 'The week starts'
  when 'Tue' then console.log '''Still accommodating to the
                                 start of the week'''
  when 'Wed' then console.log 'Time to work!'
  when 'Thu' then console.log 'Well… end of the week is near'
  when 'Fri' then console.log 'Day of the master.'
  when 'Sat', 'Sun' then console.log 'Relax'
  else console.log 'Wrong day…'
```

- And they can be used as expressions:

```
mark = switch
  when score < 100 then 'Failed'
  when score < 200 then 'Success'
  else 'Excellent!'
```

Assigns the correct value to mark

Telerik Academy

Using switch/when/else

Live Demo

Telerik Academy

Destructuring Assignments

Telerik Academy

Destructuring Assignments

- Destructuring assignments are used to extract values from arrays or objects

```
findMaximums = (numbers) ->
  min = numbers [0]
  max = numbers [0]
  for number in numbers
    max = number if max < number
    min = number if number < min
  [min, max]

[min, max] = findMaximums [1, -2, 3, -4, 5, -6]
# min will have value -6 and max will have value 5
```

Telerik Academy

Destructuring Assignments

- Destructuring assignments are used to extract values from arrays or objects

```
findMaximums = (numbers) ->
  min = numbers [0]
  max = numbers [0]
  for number in numbers
    max = number if max < number
    min = number if number < min
  [min, max]

[min, max] = findMaximums [1, -2, 3, -4, 5, -6]
# min will have value -6 and max will have value 5
```

Assigns the corresponding values to min and max

Telerik Academy

Destructuring Assignments with Arrays

Live Demo

Telerik Academy

Destructuring Assignments with Objects

- Destructuring assignments are used to extract values from arrays or objects

```
class Person
  constructor: (name, @age) ->
    [@fname, @lname] = name.split ' '

person = new Person 'Peter Petrov', 17
{fname, age} = person
```

Telerik Academy

Destructuring Assignments with Objects

- Destructuring assignments are used to extract values from arrays or objects

```
class Person
  constructor: (name, @age) ->
    [@fname, @lname] = name.split ' '
```

```
person = new Person 'Peter Petrov', 17
{fname, age} = person
```

Compiles to

```
person = new Person('Peter Petrov', 17);
fname = person.fname, age = person.age;
```

Telerik Academy

Destructuring Assignments with Objects

Live Demo

Telerik Academy

CoffeeScript Overview

http://academy.telerik.com