

DANIEL TULL

MAKING SCRIPTABLE APPS

WHY PROVIDE SCRIPTING?

Empower users

Enable features you don't want to actually build

Allow you to debug the app on device

It's fun?!

SCRIPTING LANGUAGE

Swift

Javascript

Python

AppleScript

JAVASCRIPT

Familiar to many

Very “forgiving”

JavaScriptCore included on iOS and macOS

Available on other platforms

JAVASCRIPTCORE

Objective-C wrapper around WebKit's JavaScript engine.

Insert **custom objects** into the environment.

JAVASCRIPTCORE

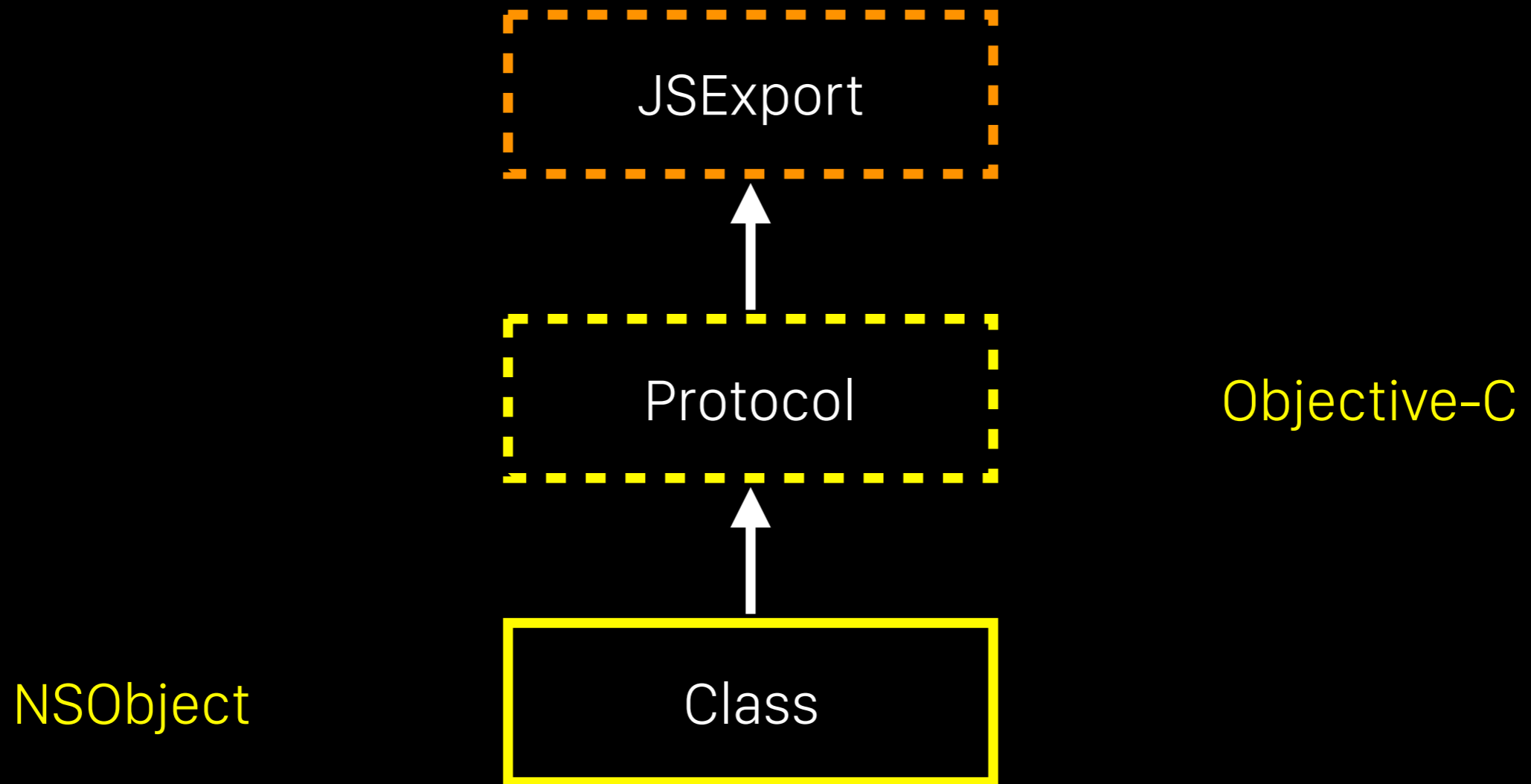
JSContext A JS execution environment

JSValue Conversion between JS and Objective-C types

JSExport A protocol to export Objective-C classes, methods and properties to JS

EXPORTING TYPES

EXPORTING TYPES



SWIFT TYPE

```
struct Position {  
  
    var x: Double  
    var y: Double  
  
    init(x: Double, y: Double) {  
        self.x = x  
        self.y = y  
    }  
}
```

EXPORT PROTOCOL

```
@objc protocol PositionExport: JSExport {  
  
    init(x: Double, y: Double)  
  
    var x: Double { get set }  
    var y: Double { get set }  
}
```

OBJECTIVE-C WRAPPER

```
final class JSPosition: NSObject {  
  
    var position: Position  
  
    init(_ position: Position) {  
        self.position = position  
    }  
}
```

IMPLEMENT PROTOCOL

```
extension JSPosition: PositionExport {  
  
    convenience init(x: Double, y: Double) {  
        let position = Position(x: x, y: y)  
        self.init(position)  
    }  
  
    dynamic var x: Double {  
        get { return position.x }  
        set { position.x = newValue }  
    }  
}
```

EXPORTING TYPES

```
let context = JSContext()  
context.setObject(JSPosition.self, for: "Position")  
context.evaluateScript(script)
```

JAVASCRIPT

```
var position = new Position(0,0);  
position.x = 13;  
position.y = 42;
```

SOURCERY

Scans Swift code, generates files

Created by Krzysztof Zabłocki (@merowing)

SOURCERY

```
init?({% for parameter in method.parameters where parameter.name != "identifier"
%}{{ parameter.name }}: {% if parameter.isArray %}{% if
parameter.typeName.array.elementType.implements.JSGenerate %}
[JS{{ parameter.typeName.array.elementTypeName | replace:".", ""
}}]?{% elif parameter.typeName.array.elementTypeName ==
"Positive" or parameter.typeName.array.elementTypeName ==
"Percentage" %}[Double]?{% elif
parameter.typeName.array.elementTypeName == "String" or
parameter.typeName.array.elementTypeName == "Double" or
parameter.typeName.array.elementTypeName == "Boolean" %}
[{{ parameter.typeName.array.elementTypeName }}]1" %}
[{{ parameter.typeName.array.elementTypeName }}]?{% endif %}{%
else %}{% if parameter.type.implements.JSGenerate %}
JS{{ parameter.actualTypeName.unwrappedTypeName | replace:".", "" }}?{% elif
parameter.actualTypeName.unwrappedTypeName == "Positive" or
parameter.actualTypeName.unwrappedTypeName == "Percentage" %}Double{% elif
parameter.actualTypeName.unwrappedTypeName == "String" or
parameter.actualTypeName.unwrappedTypeName == "Double" or
parameter.actualTypeName.unwrappedTypeName == "Boolean" %}
{{ parameter.actualTypeName.unwrappedTypeName }}{% endif %}{% endif %}{% if not
forloop.last %}, {% endif %}{% endfor %})
```


GOTCHAS

NOT A NUMBER

```
var position = new Position();
```

```
JSPosition(x: NaN, y: NaN)
```

```
Position(x: NaN, y: NaN)
```

Possibly unexpected?

```
struct Area {  
  
    var position: Position  
    var size: Size  
  
    init(position: Position, size: Size) {  
        self.position = position  
        self.size = size  
    }  
}
```

NIL ON NON OPTIONALS!

```
var area = new Area();
```

```
JSArea(position: nil, size: nil)
```

```
Area(position: nil, size: nil)
```

Crash

SANITISE INPUT

```
init?(jsposition: JSPosition?, jssize: JSSize?) {  
  
  guard  
    let jsposition = jsposition,  
    let jssize = jssize  
  else {  
    return nil  
  }  
  
  let position = Position(jsposition)  
  let size = Size(jssize)  
  let area = Area(position: position, size: size)  
  self.init(area)  
}
```

INCORRECT TYPES

```
var size = new Size(10, 20);
```

```
var area = new Area(size, size);
```

```
JSArea(position: size, size: size)
```

```
Area(position: size, size: size)
```

Crash

SANITISE INPUT

```
init?(jsposition: Any?, jssize: Any?) {  
  
  guard  
    let jsposition = jsposition as? JSPosition,  
    let jssize = jssize as? JSSize  
  else {  
    return nil  
  }  
  
  let position = Position(jsposition)  
  let size = Size(jssize)  
  let area = Area(position: position, size: size)  
  self.init(area)  
}
```

THREADING

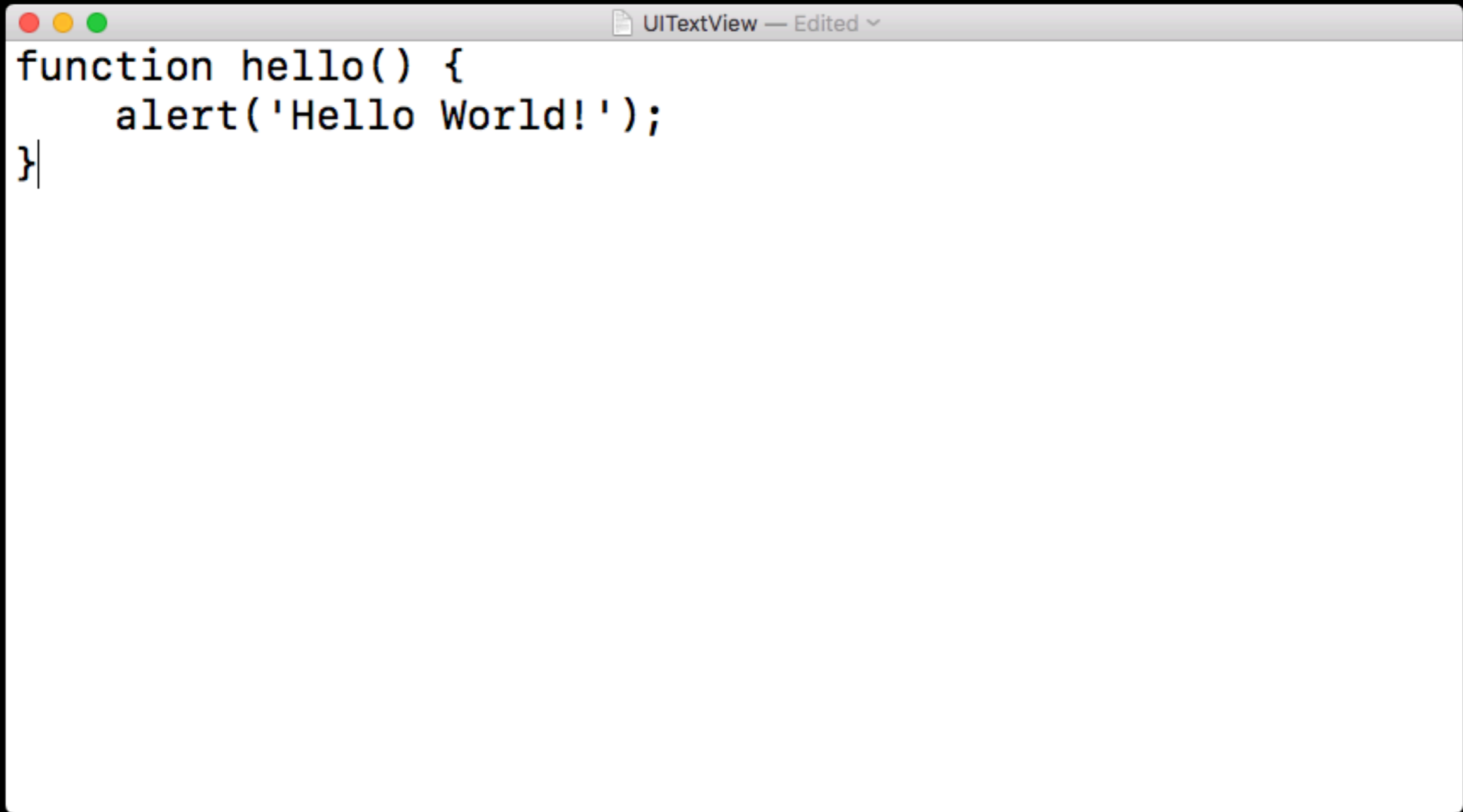
Calls to **your code** made on the thread of the JSContext.

That thread is whatever one you **instantiate** the JSContext on.

Maybe best not to block the **main thread**.

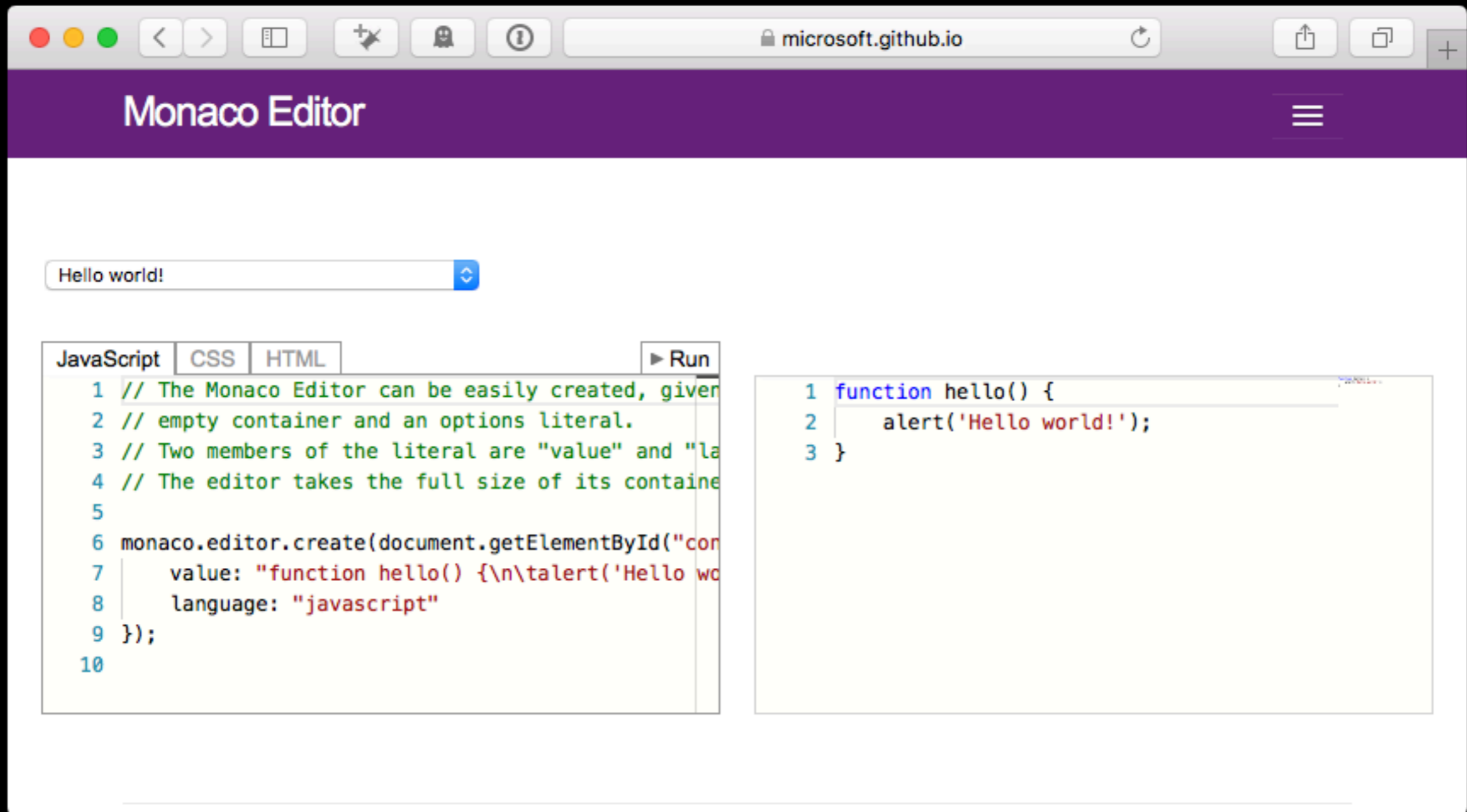
CODE EDITOR

UITextView



```
function hello() {  
    alert('Hello World!');  
}
```

MONACO EDITOR



ACE

The image shows a browser window at `ace.c9.io` displaying the ACE code editor. The editor is configured with the following settings:

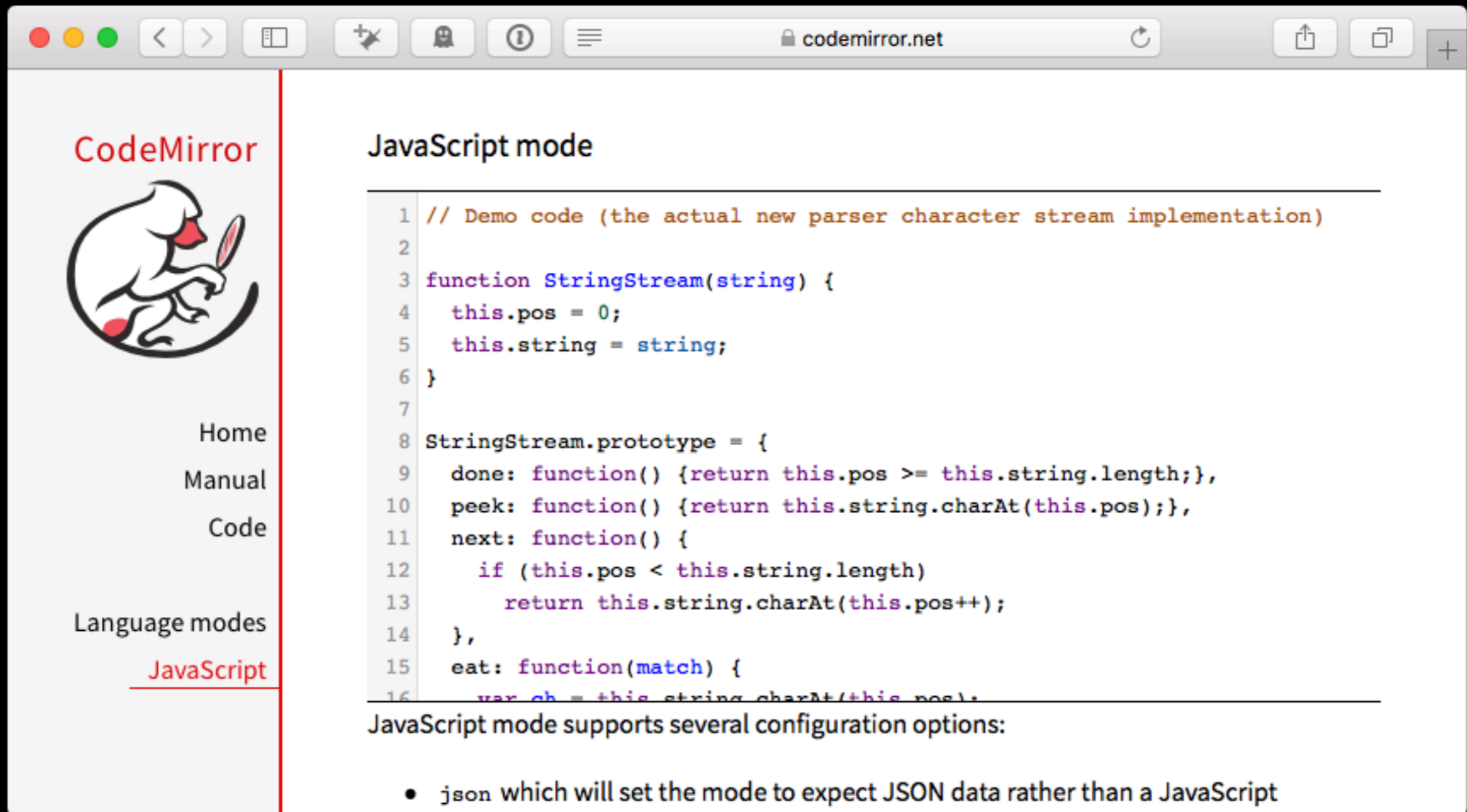
- Document: JavaScript
- Mode: JavaScript
- Split: None
- Theme: Chrome
- Font Size: 20px
- Code Folding: mark begin and end
- Key Binding: Ace
- Soft Wrap: Off
- Full Line Selection:
- Highlight Active Line:
- Show Invisibles:
- Show Indent Guides:
- Persistent HScroll: VScroll:
- Animate scrolling:

The code editor contains the following JavaScript code:

```
1- function foo(items, nada) {  
2-     for (var i=0; i<items.length; i++) {  
3-         alert(items[i] + "juhu\n");  
4-     } // Real Tab.  
5- }
```

The status bar at the bottom right shows a zoom level of 4:1.

CODEMIRROR



The screenshot shows a web browser window with the URL `codemirror.net`. The page features a sidebar on the left with the CodeMirror logo (a monkey holding a mirror) and navigation links: [Home](#), [Manual](#), [Code](#), and [Language modes](#). The [JavaScript](#) link is highlighted. The main content area is titled "JavaScript mode" and displays a code editor with the following JavaScript code:

```
1 // Demo code (the actual new parser character stream implementation)
2
3 function StringStream(string) {
4   this.pos = 0;
5   this.string = string;
6 }
7
8 StringStream.prototype = {
9   done: function() {return this.pos >= this.string.length;},
10  peek: function() {return this.string.charAt(this.pos);},
11  next: function() {
12    if (this.pos < this.string.length)
13      return this.string.charAt(this.pos++);
14  },
15  eat: function(match) {
16    var ch = this.string.charAt(this.pos);
```

Below the code editor, the text "JavaScript mode supports several configuration options:" is followed by a bulleted list:

- `json` which will set the mode to expect JSON data rather than a JavaScript

CODEEDITOR.HTML

```
<!DOCTYPE html>
<head>
<meta name="viewport" content="user-scalable=no,
width=device-width">
<script src="CodeMirror/lib/codemirror.js"></script>
</head>
<body>

<textarea id="CodeEditor"></textarea>

<script src="CodeEditor.js"></script>

</body>
</html>
```

CODEEDITOR.JS

```
var editor =
CodeMirror.fromTextArea(document.getElementById
("CodeEditor"), {
  lineNumbers: true,
  lineWrapping: true,
  mode: "javascript",
  matchBrackets: true,
  autoCloseBrackets: true
});
```

CODEEDITOR.SWIFT

```
webView.evaluateJavaScript("editor.getValue();")
{ (result, error) in

    guard let script = (result as? String) else {
        // Handle the error
        return
    }

    // Do stuff with the script
}
```


JUST ADD MORE

Language	files	blank	comment	code
JavaScript	84	5628	11323	43066
JSON	1	0	0	1929
CSS	11	107	24	747
HTML	4	34	0	257
Swift	2	44	21	121
Bourne Shell	5	27	12	66
C/C++ Header	1	2	0	3
SUM:	108	5842	11380	46189

EDITOR



A screenshot of a code editor window. The window title bar shows the file path: `file:///Users/danielctull/Developer/Projects/Alloy/Co`. The editor contains the following JavaScript code:

```
1 function hello() {  
2  
3   var hello = "Hello world!";  
4  
5   console.log(hello);  
6 }  
7
```

LINTING



The image shows a code editor window with a file path in the address bar: `file:///Users/danielctull/Developer/Projects/Alloy/Co`. The code in the editor is as follows:

```
1 function hello() {  
2  
3   var hello = "Hello world!"  
4  
5   console.log(hello);  
6 }  
7
```

A linting error is highlighted in a red box on line 1, stating: **Missing semicolon.** The error points to the opening curly brace of the function definition.

AUTOCOMplete



The image shows a code editor window with a file path of `file:///Users/danielctull/Developer/Projects/Alloy/Co`. The code in the editor is as follows:

```
1 function hello() {  
2  
3   var hello = "Hello world!";  
4   hello.s  
5   consol  
6 }  
7
```

An autocomplete menu is displayed over the code, listing several methods for the `hello.s` object:

- search
- slice
- split**
- startsWith
- substr
- substring

The `split` method is highlighted in blue. A tooltip for this method is shown to the right, containing the text: "Splits a String object into an array of strings by separating the string into substrings."

JSON DEFINITION

```
"Position": {  
  "!type": "fn(x: number, y: number) -> Position",  
  "prototype": {  
    "x": "number",  
    "y": "number"  
  }  
},
```

@DANIELCTULL

DANIELTULL.CO.UK