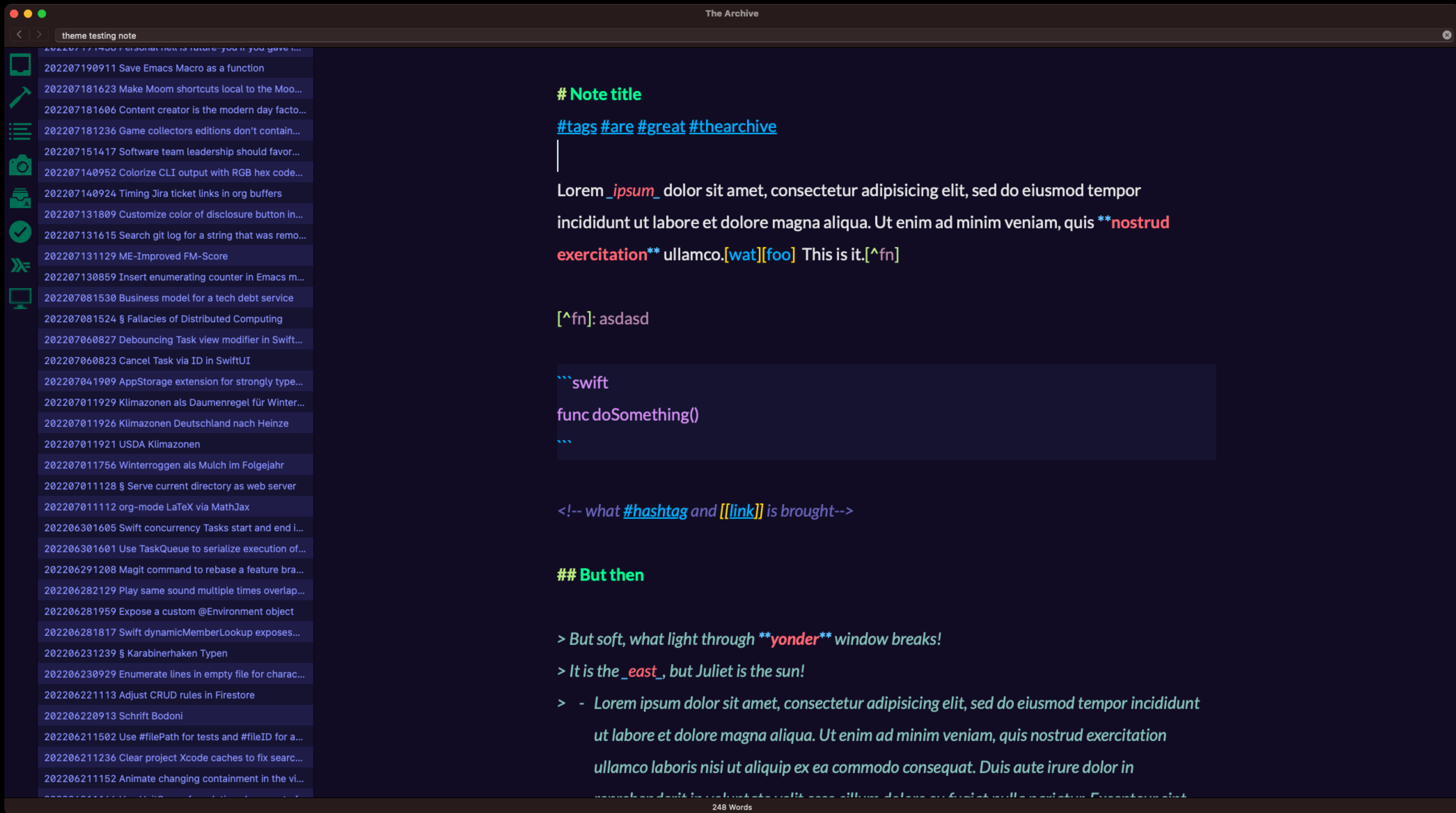


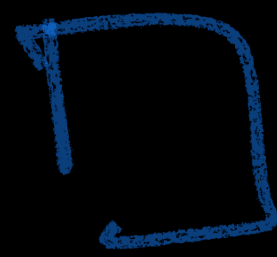
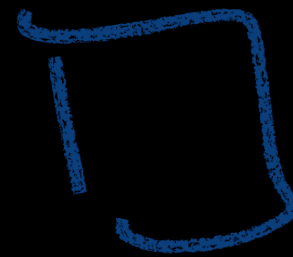
securely
Make your app extensible
with JavaScriptCore

`christiantietze.de // @ctietze`



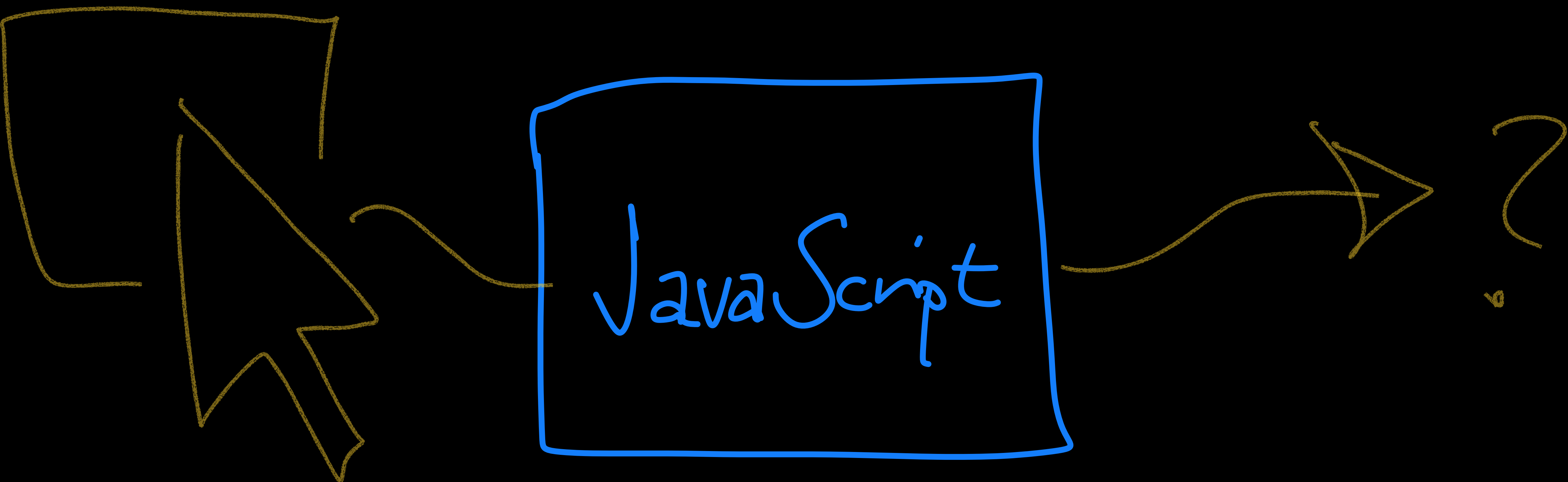


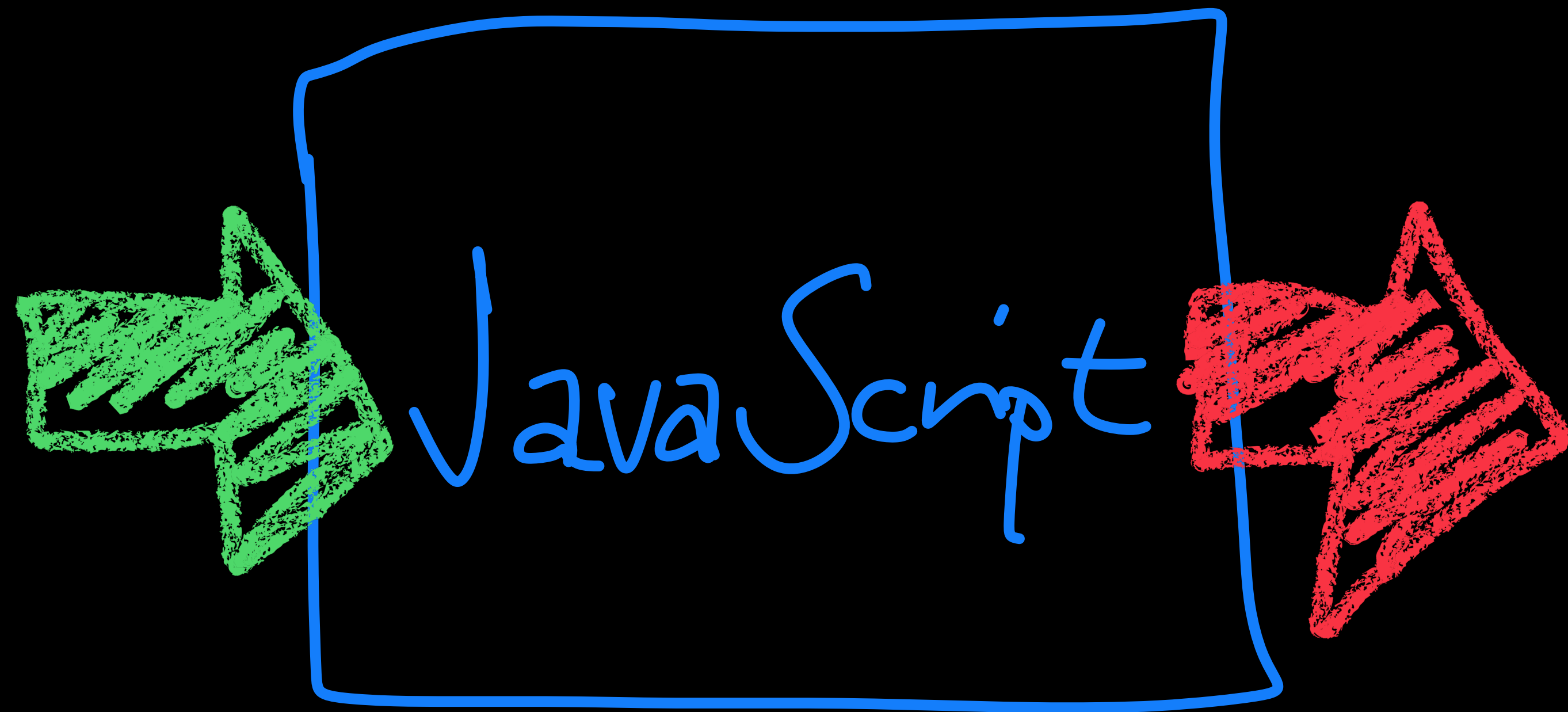
000



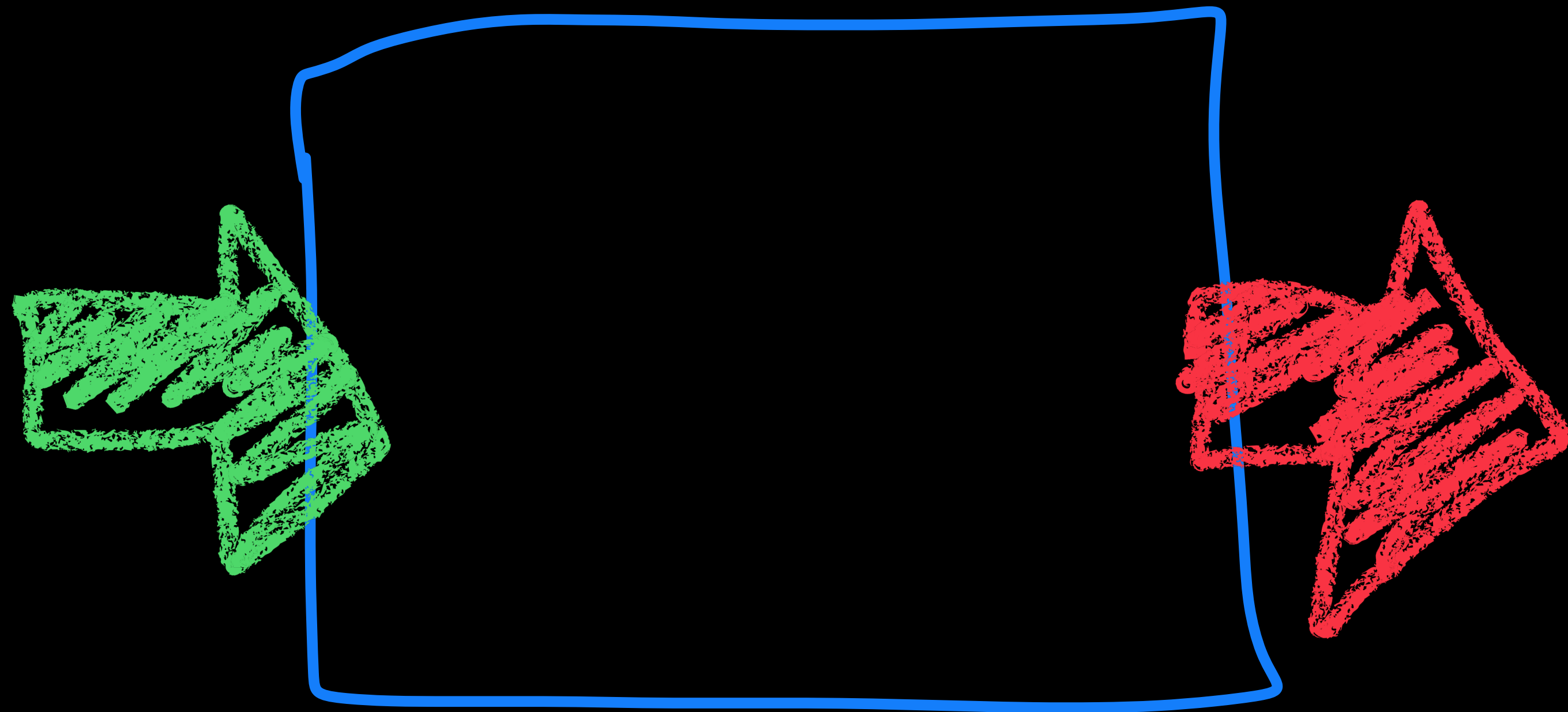
JavaScript

change





Black Box



$$\underline{y} = f(\underline{x})$$

JavaScriptCore

Execution Environment

> [C] JSVirtualMachine

> [C] JSContext

JavaScript Code

> [C] JSValue

> [C] JSManagedValue

Native Code

[P] JSEExport

C API

> [≡] C JavaScriptCore API

Reference

> [≡] JavaScriptCore Constants

[≡] Filter

[≡] Filter

Overview

The JavaScriptCore framework provides the ability to evaluate JavaScript programs from within Swift, Objective-C, and C-based apps. You can use also use JavaScriptCore to insert custom objects into the JavaScript environment.

Topics

Execution Environment

class JSVirtualMachine
A self-contained environment for JavaScript execution.

class JSContext
A JavaScript execution environment.

JavaScript Code

class JSValue
A JavaScript value.

class JSManagedValue
A JavaScript value with conditional retain behavior to provide automatic memory management.

Native Code

protocol JSEExport
The protocol for exporting Objective-C objects to JavaScript.

JavaScriptCore.framework

JavaScriptCore

JSContext

Creating JavaScript Contexts

init!()

init!(virtualMachine: JSVirtualMachine!)

Evaluating Scripts

func evaluateScript(String!) -> JSValue!

func evaluateScript(String!, withSourceURL:...

Inspecting Callback State in a Running Context

class func current() -> JSContext!

class func currentCallee() -> JSValue!

class func currentThis() -> JSValue!

class func currentArguments() -> [Any]!

Working with JavaScript Global State

var globalObject: JSValue!

var exception: JSValue!

var exceptionHandler: ((JSContext?, JSValu...

var virtualMachine: JSVirtualMachine!

var name: String!

Accessing JavaScript Global State with Subsc...

func objectForKeyedSubscript(Any!) -> JSV...

Filter

Filter

func objectForKeyedSubscript(Any!) -> JSV...

Topics

Creating JavaScript Contexts

`init!()`
Initializes a new JavaScript context.

`init!(virtualMachine: JSVirtualMachine!)`
Creates a new JavaScript context associated with a specific virtual machine.

Evaluating Scripts

`func evaluateScript(String!) -> JSValue!`
Executes the specified JavaScript code.

`func evaluateScript(String!, withSourceURL: URL!) -> JSValue!`
Executes the specified JavaScript code, treating the specified URL as its source location.

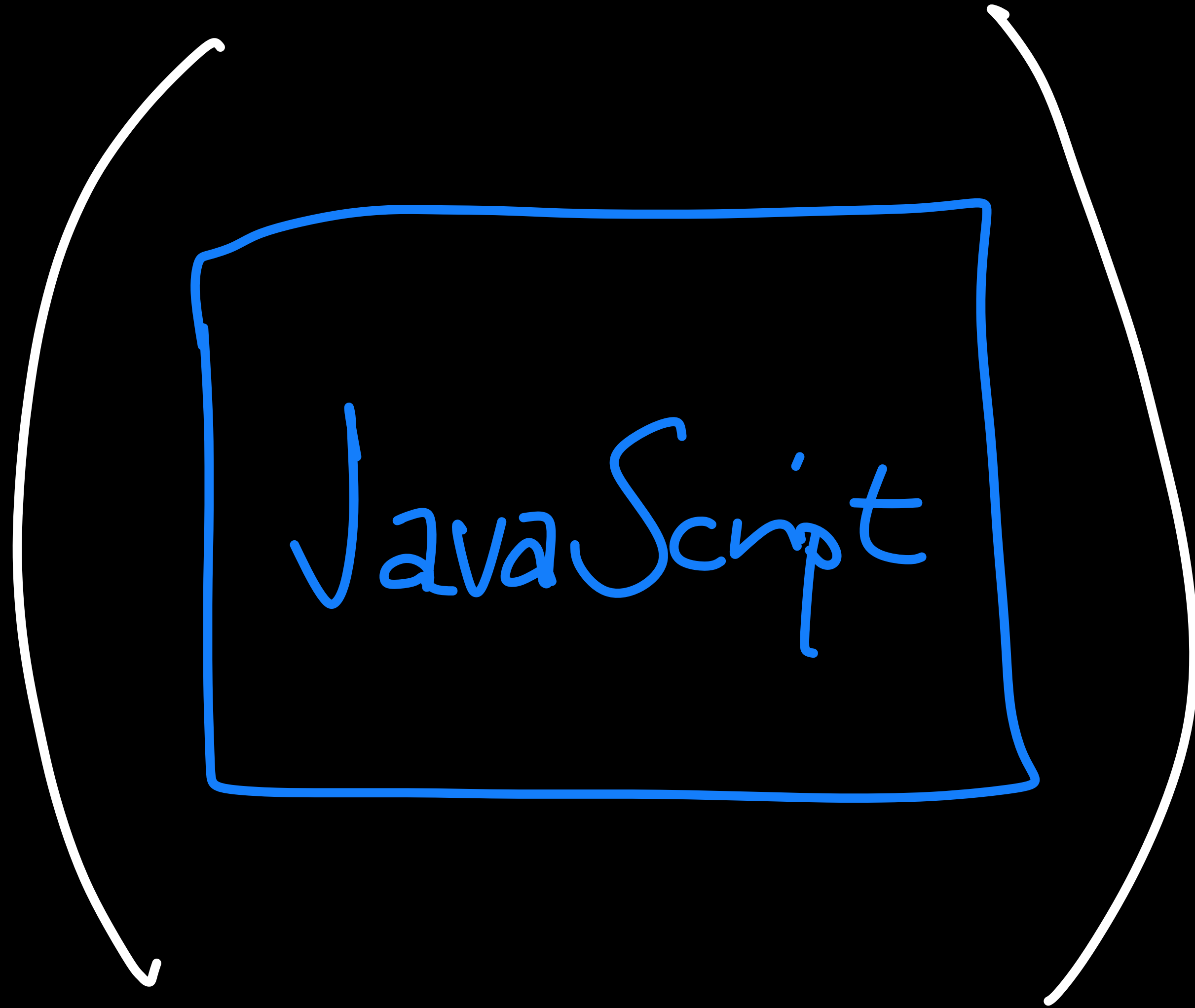
Inspecting Callback State in a Running Context

`class func current() -> JSContext!`
Returns the context currently executing JavaScript code.

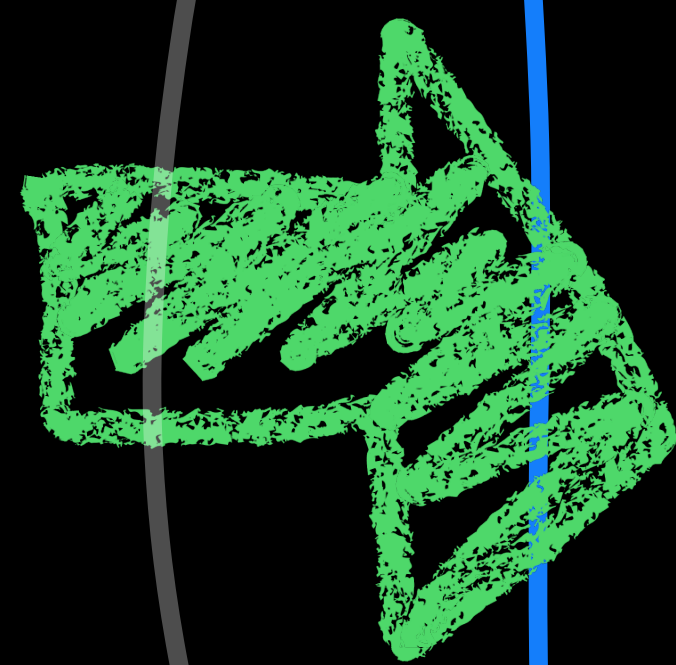
Returns the context currently executing JavaScript code.

`class func current() -> JSContext!`

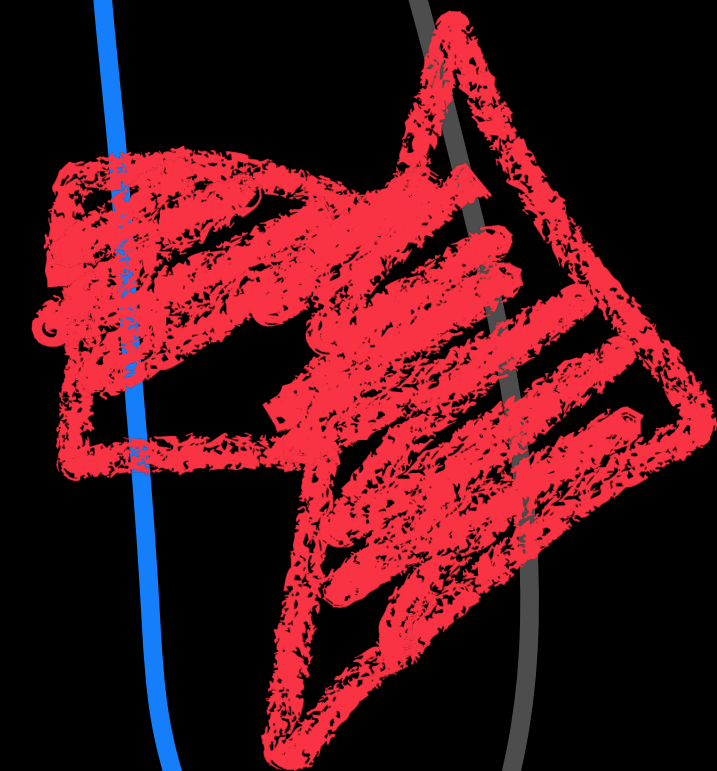
evaluate.



evaluate.



JavaScript



Functional

IO

Functional

f_I

f_O

evaluate(String)

const aValue = input();

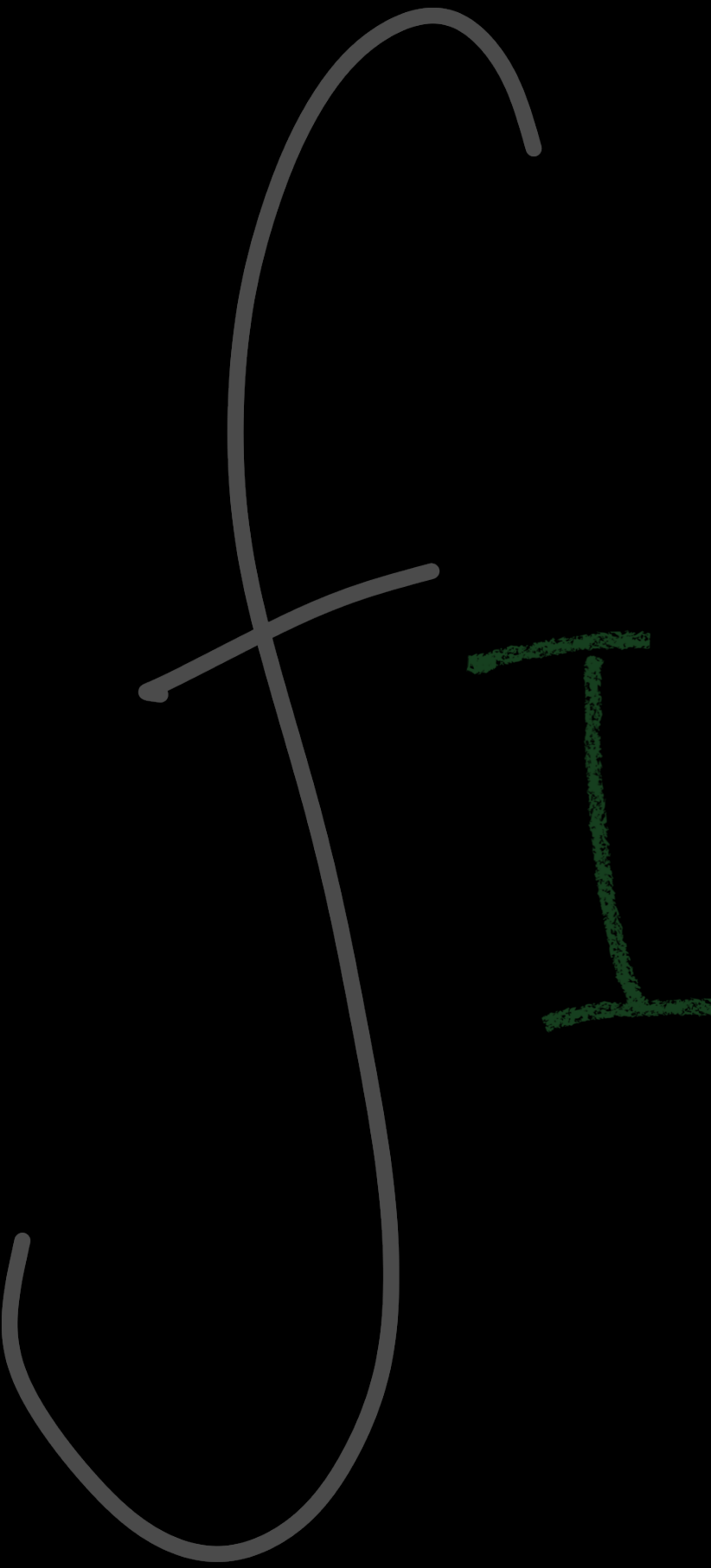
...

output (another Value);

Time for Code!


```
struct Plugin {  
    let manifest: Manifest  
    var name: String { manifest.name }  
    let javascript: JavaScript  
    let checksum: Checksum  
}
```

```
struct Manifest {  
    struct Author { ... }  
    struct Input { ... }  
    struct Output { ... }  
  
    let name: String  
    let title: String  
    let description: String  
    let version: Version  
    let releaseDate: Date  
    let appVersion: Version  
    let authors: [Author]  
    let input: Input  
    let output: Output  
    let dependencies: [String]  
  
    ...  
}
```



```
struct Manifest {  
  ...  
  struct Input {  
    enum Selection: String, Hashable {  
      case selected  
      case all  
    }  
  
    let notes: Set<Selection>  
    let text: Set<Selection>  
  }  
  ...  
}
```


f

```
struct Manifest {  
    ...  
    struct Output {  
        enum File {  
            case change(Filename)  
            case new  
        }  
  
        enum Text {  
            case insert  
        }  
  
        let file: File?  
        let text: Text?  
    }  
    ...  
}
```



```
/// A context defines available inputs and outputs for a `JavaScript` during execution.
class Context {
    let input: Environment.Input
    let output: Environment.Output
    let utils: Utils
    private let jsContext: JSContext

    ...

    func execute(javaScript: JavaScript) throws -> Effects {
        let outputCollector = output.outputCollector()
        configure(input: input.jsValue(context: jsContext),
                  output: outputCollector,
                  utils: utils)

        jsContext.evaluateScript(javaScript.code)
        if let exception = jsContext.exception {
            throw JavaScriptError(message: exception.toString())
        }

        return outputCollector.effects()
    }

    private func configure(input: JSValue?, output: JSExport?, utils: JSExport?) {
        jsContext["input"] = input
        jsContext["output"] = output
        jsContext["utils"] = utils
    }
}
```

/// A context defines available inputs and outputs for a `JavaScript` during execution.

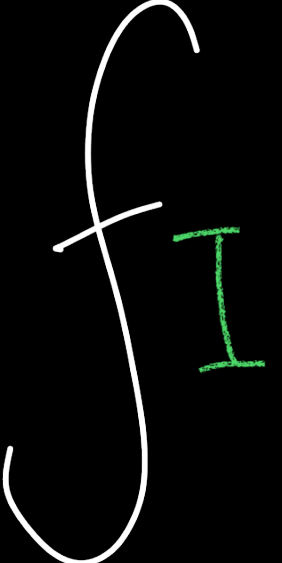
```
class Context {  
    let input: Environment.Input  
    let output: Environment.Output  
    let utils: Utils  
    private let jsContext: JSContext
```

...



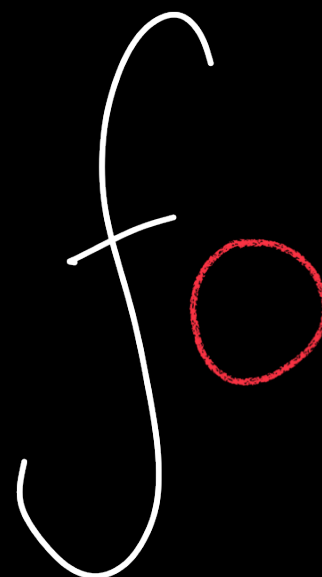
```
func execute(javaScript: JavaScript) throws -> Effects {  
    let outputCollector = output.outputCollector()  
    configure(input: input.jsValue(context: jsContext),  
              output: outputCollector,  
              utils: utils)  
  
    jsContext.evaluateScript(javaScript.code)  
    if let exception = jsContext.exception {  
        throw JavaScriptError(message: exception.toString())  
    }
```

```
    return outputCollector.effects()  
}
```



```
private func configure(input: JSValue?, output: JSExport?, utils: JSExport?) {  
    jsContext["input"] = input  
    jsContext["output"] = output  
    jsContext["utils"] = utils  
}
```

```
}
```



```
/// A context defines available inputs and outputs for a `JavaScript` during execution.
class Context {
    let input: Environment.Input
    let output: Environment.Output
    let utils: Utils
    private let jsContext: JSContext

    ...

    func execute(javaScript: JavaScript) throws -> Effects {
        let outputCollector = output.outputCollector()
        configure(input: input.jsValue(context: jsContext),
                  output: outputCollector,
                  utils: utils)

        jsContext.evaluateScript(javaScript.code)
        if let exception = jsContext.exception {
            throw JavaScriptError(message: exception.toString())
        }

        return outputCollector.effects()
    }

    private func configure(input: JSValue?, output: JSExport?, utils: JSExport?) {
        jsContext["input"] = input
        jsContext["output"] = output
        jsContext["utils"] = utils
    }
}
```




```
/// A context defines available inputs and outputs for a `JavaScript` during execution.
class Context {
    let input: Environment.Input
    let output: Environment.Output
    let utils: Utils
    private let jsContext: JSContext

    ...

    func execute(javaScript: JavaScript) throws -> Effects {
        let outputCollector = output.outputCollector()
        configure(input: input.jsValue(context: jsContext),
                  output: outputCollector,
                  utils: utils)

        jsContext.evaluateScript(javaScript.code)
        if let exception = jsContext.exception {
            throw JavaScriptError(message: exception.toString())
        }

        return outputCollector.effects()
    }

    private func configure(input: JSValue?, output: JSExport?, utils: JSExport?) {
        jsContext["input"] = input
        jsContext["output"] = output
        jsContext["utils"] = utils
    }
}
```



```
protocol NoteContentCollectorJSEExports: JSEExport {  
    var filename: String? { get }  
    var content: String { get set }  
}
```

```
protocol InsertCollectorJSEExports: JSEExport {  
    var text: String { get set }  
}
```



```
protocol OutputCollectorJSEExports: JSEExport {  
    var newFile: NoteContentCollectorJSEExports? { get }  
    var changeFile: NoteContentCollectorJSEExports? { get }  
    var insert: InsertCollectorJSEExports? { get }  
}
```



```
protocol NoteContentCollectorJSEExports: JSEExport {  
    var filename: String? { get }  
    var content: String { get set }  
}
```

```
protocol InsertCollectorJSEExports: JSEExport {  
    var text: String { get set }  
}
```

```
protocol OutputCollectorJSEExports: JSEExport {  
    var newFile: NoteContentCollectorJSEExports? { get }  
    var changeFile: NoteContentCollectorJSEExports? { get }  
    var insert: InsertCollectorJSEExports? { get }  
}
```

```
struct Manifest {  
    ...  
    struct Output {  
        enum File {  
            case change(Filename)  
            case new  
        }  
  
        enum Text {  
            case insert  
        }  
  
        let file: File?  
        let text: Text?  
    }  
    ...  
}
```

```
protocol NoteContentCollectorJSEExports: JSEExport {  
    var filename: String? { get }  
    var content: String { get set }  
}
```

```
protocol InsertCollectorJSEExports: JSEExport {  
    var text: String { get set }  
}
```

```
protocol OutputCollectorJSEExports: JSEExport {  
    var newFile: NoteContentCollectorJSEExports? { get }  
    var changeFile: NoteContentCollectorJSEExports? { get }  
    var insert: InsertCollectorJSEExports? { get }  
}
```

```
struct Manifest {  
    ...  
    struct Output {  
        enum File {  
            case change(Filename)  
            case new  
        }  
  
        enum Text {  
            case insert  
        }  
  
        let file: File?  
        let text: Text?  
    }  
    ...  
}
```

```
protocol NoteContentCollectorJSEExports: JSEExport {  
    var filename: String? { get }  
    var content: String { get set }  
}
```

```
protocol InsertCollectorJSEExports: JSEExport {  
    var text: String { get set }  
}
```

```
protocol OutputCollectorJSEExports: JSEExport {  
    var newFile: NoteContentCollectorJSEExports? { get }  
    var changeFile: NoteContentCollectorJSEExports? { get }  
    var insert: InsertCollectorJSEExports? { get }  
}
```


Time for a demo! + playground

But first:

questions?

```
.forEach { try $0.ask() }
```

christiantietze.de // @ctietze