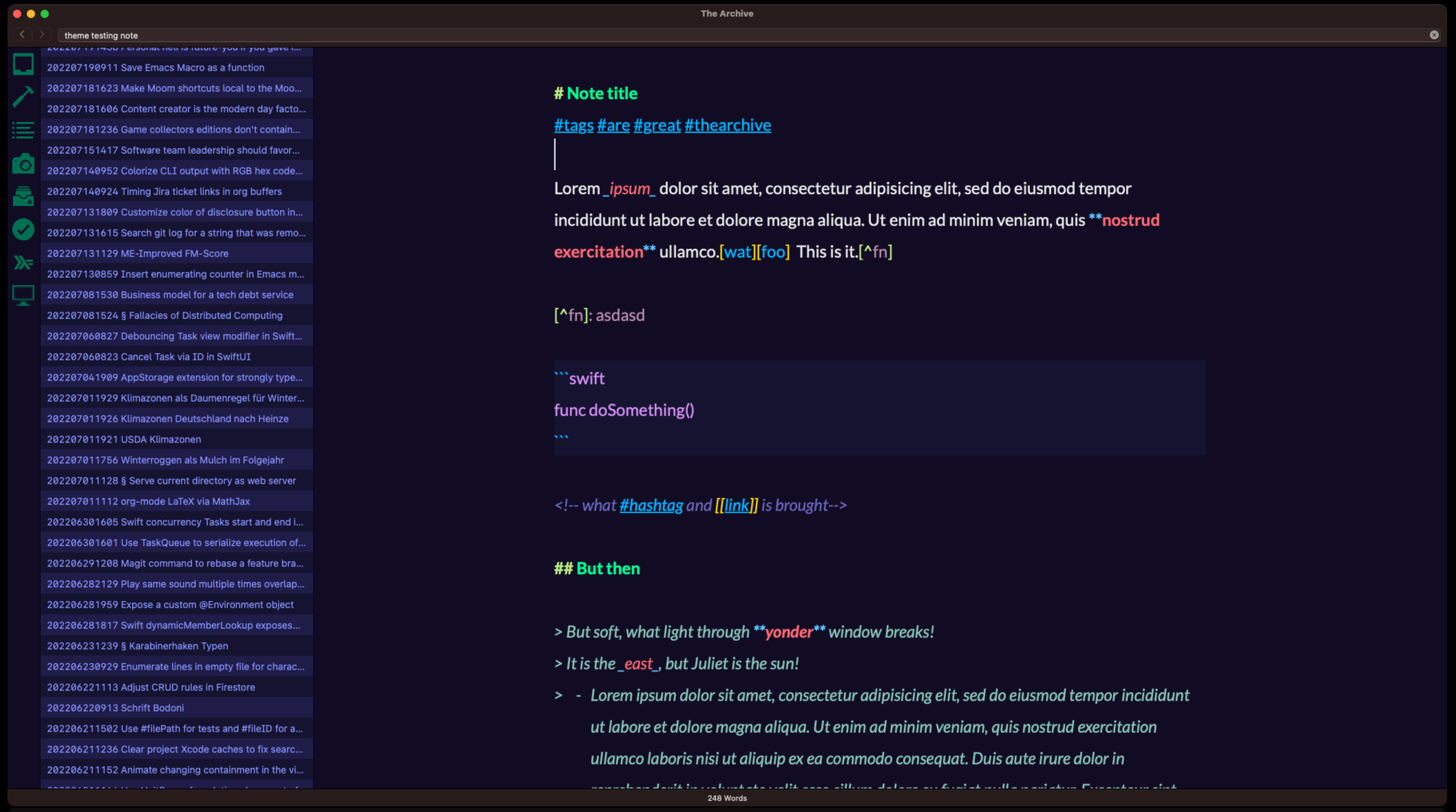
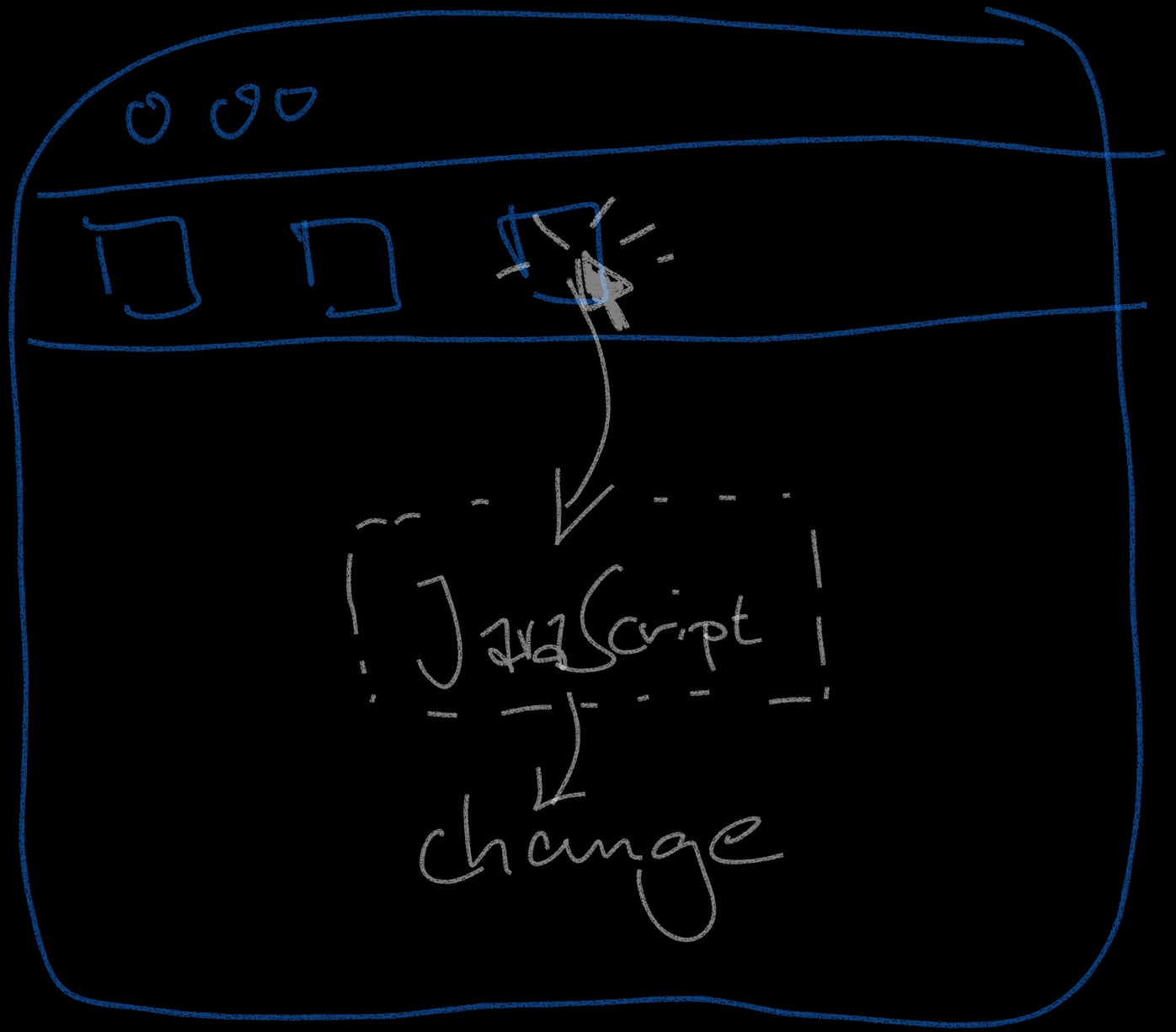
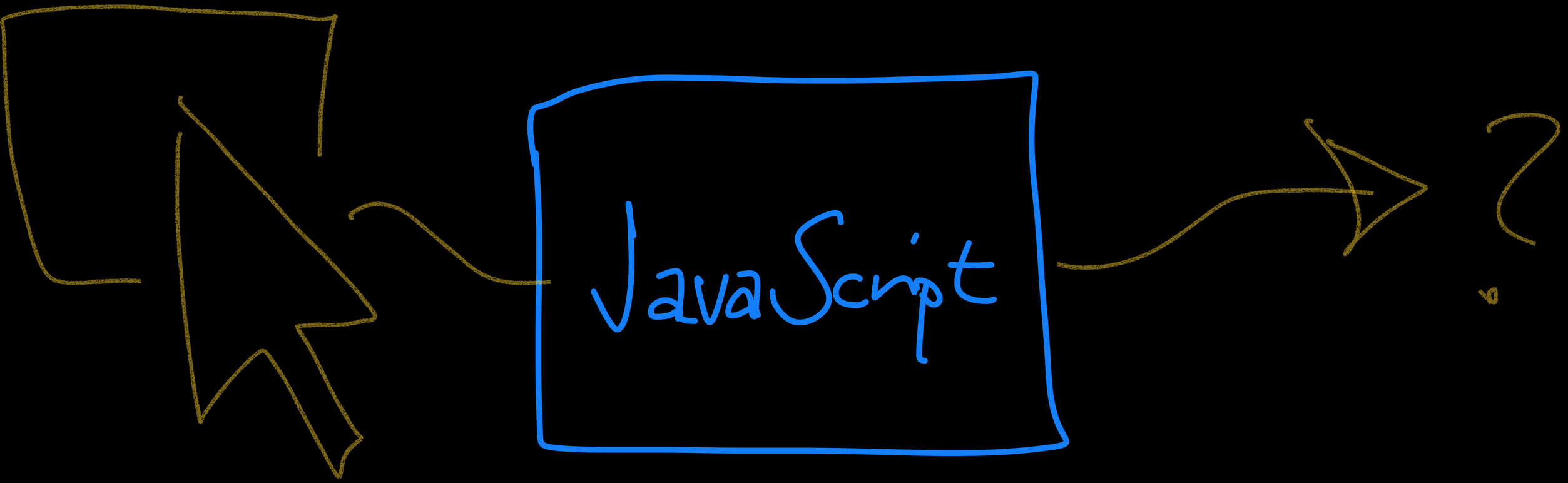
Make your app *securely* extensible with JavaScriptCore

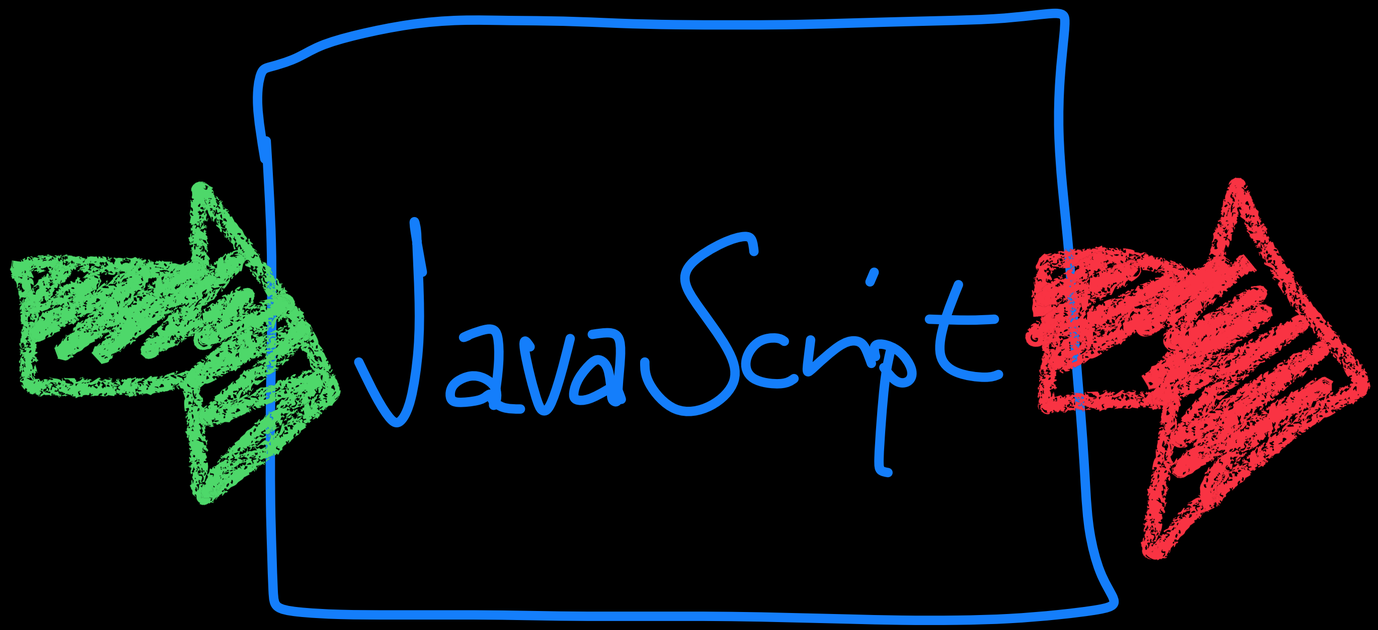
christiantietze.de // @ctietze



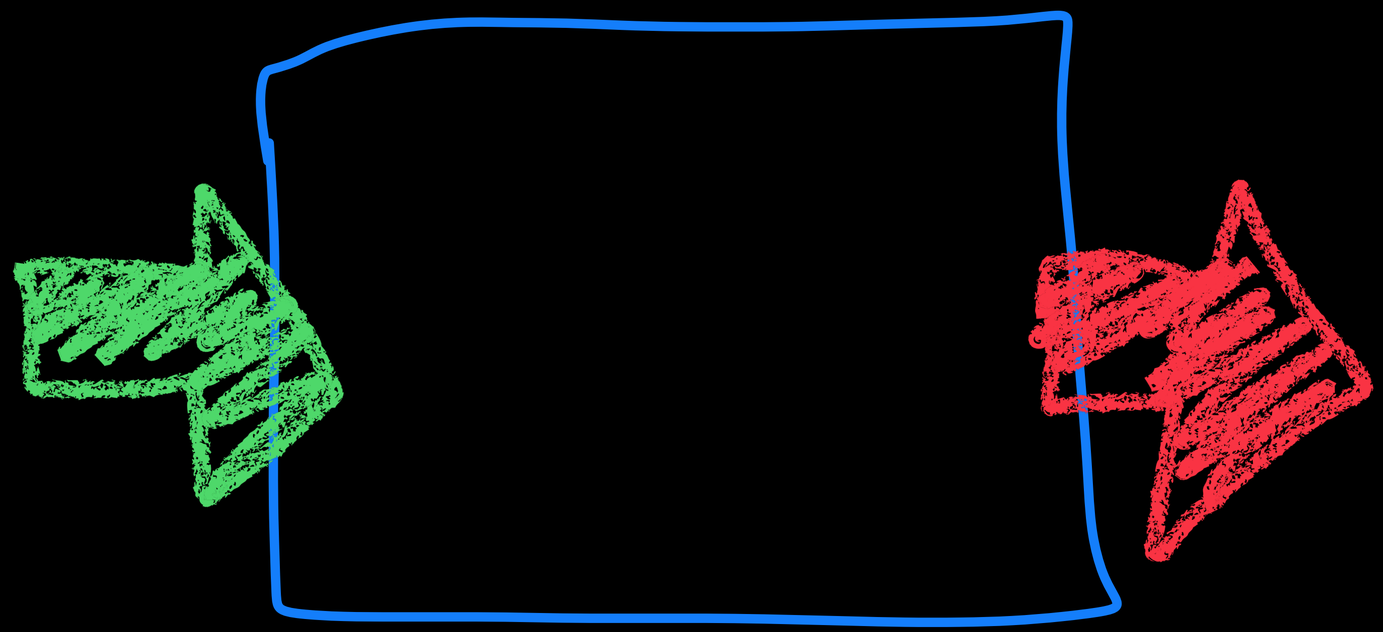
The Archive.app

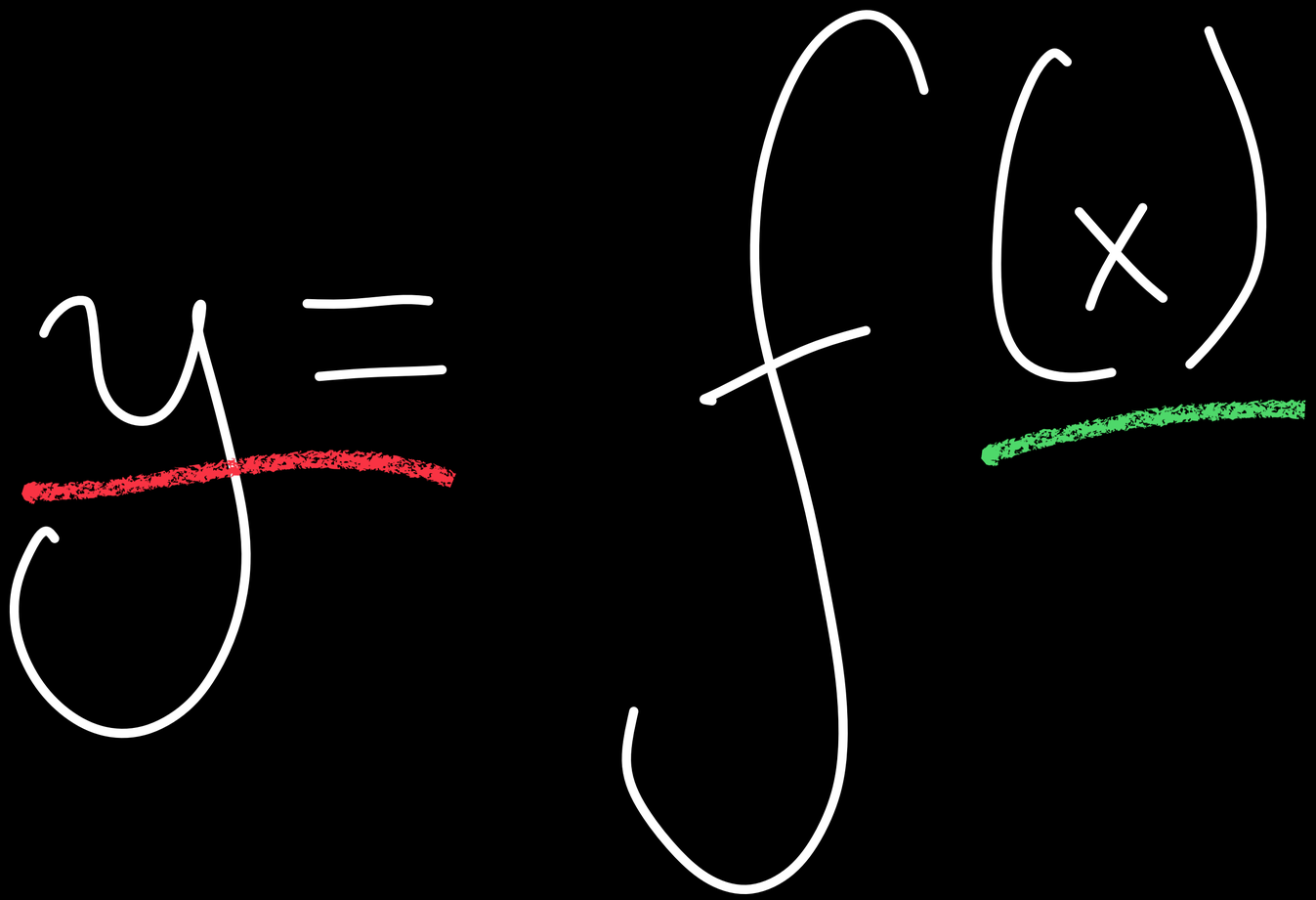

JavaScript
change

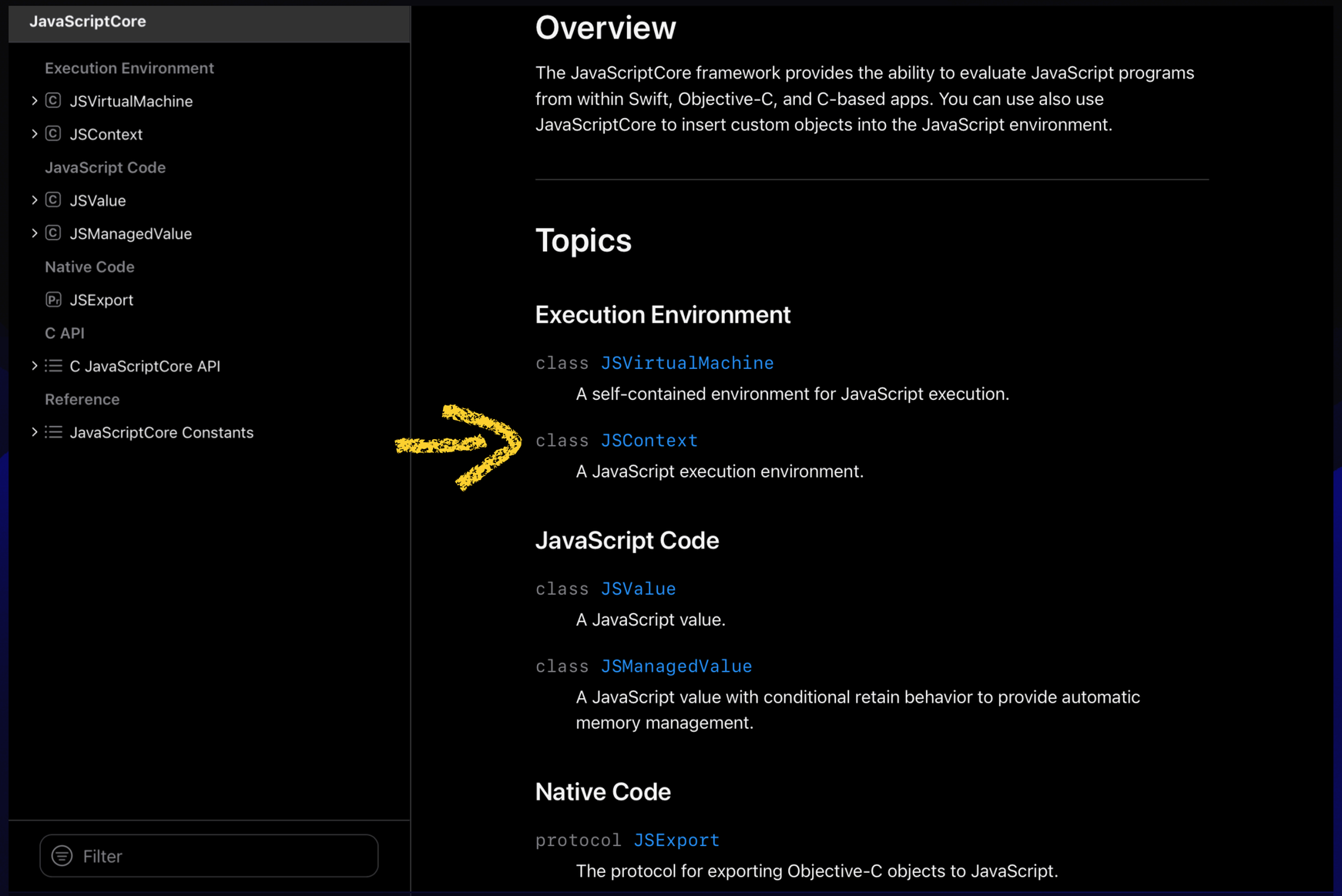

JavaScript


JavaScript

Black Box




y = f(x)



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...

Filter

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.

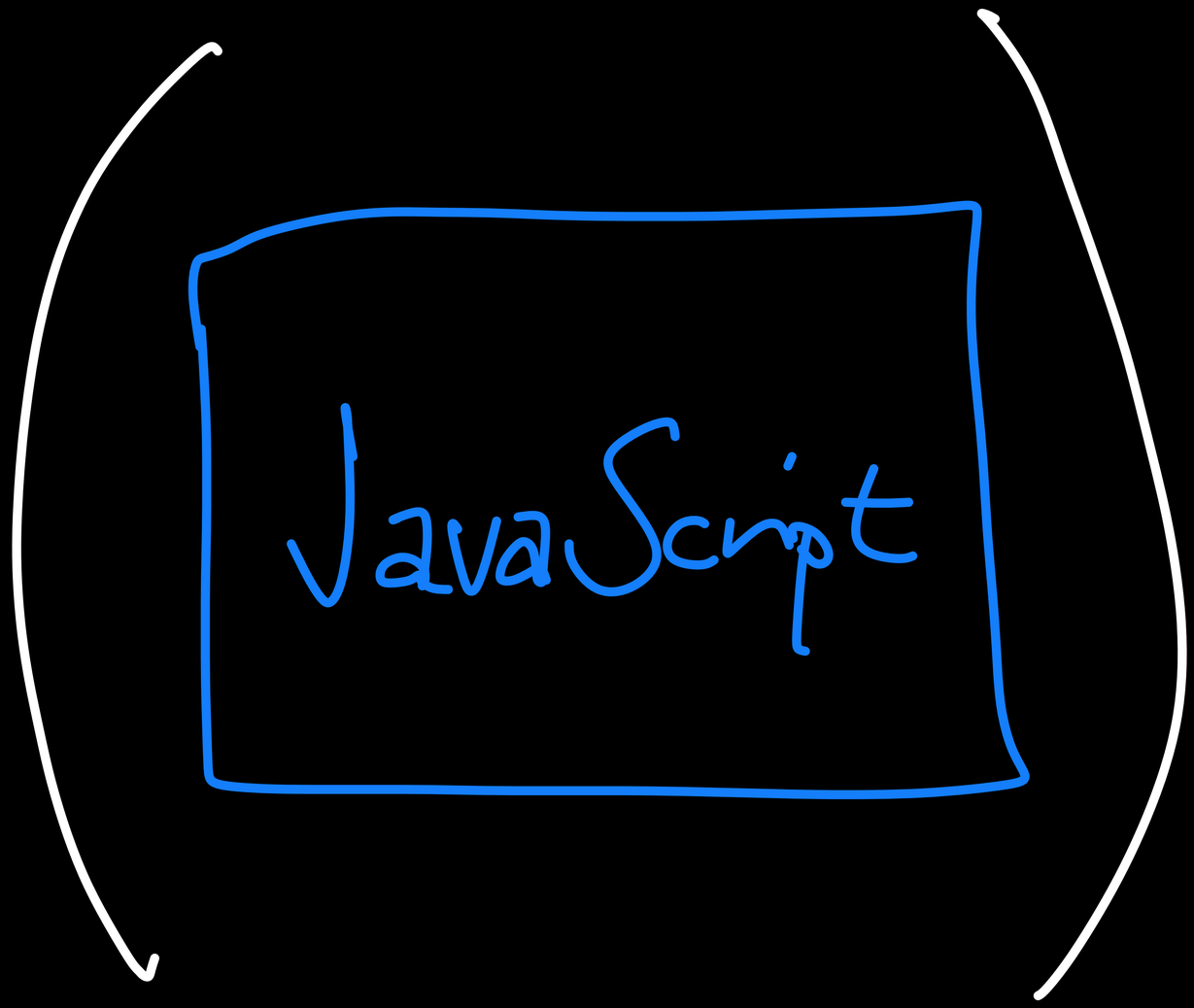
Inspecting Callback State in a Running Context

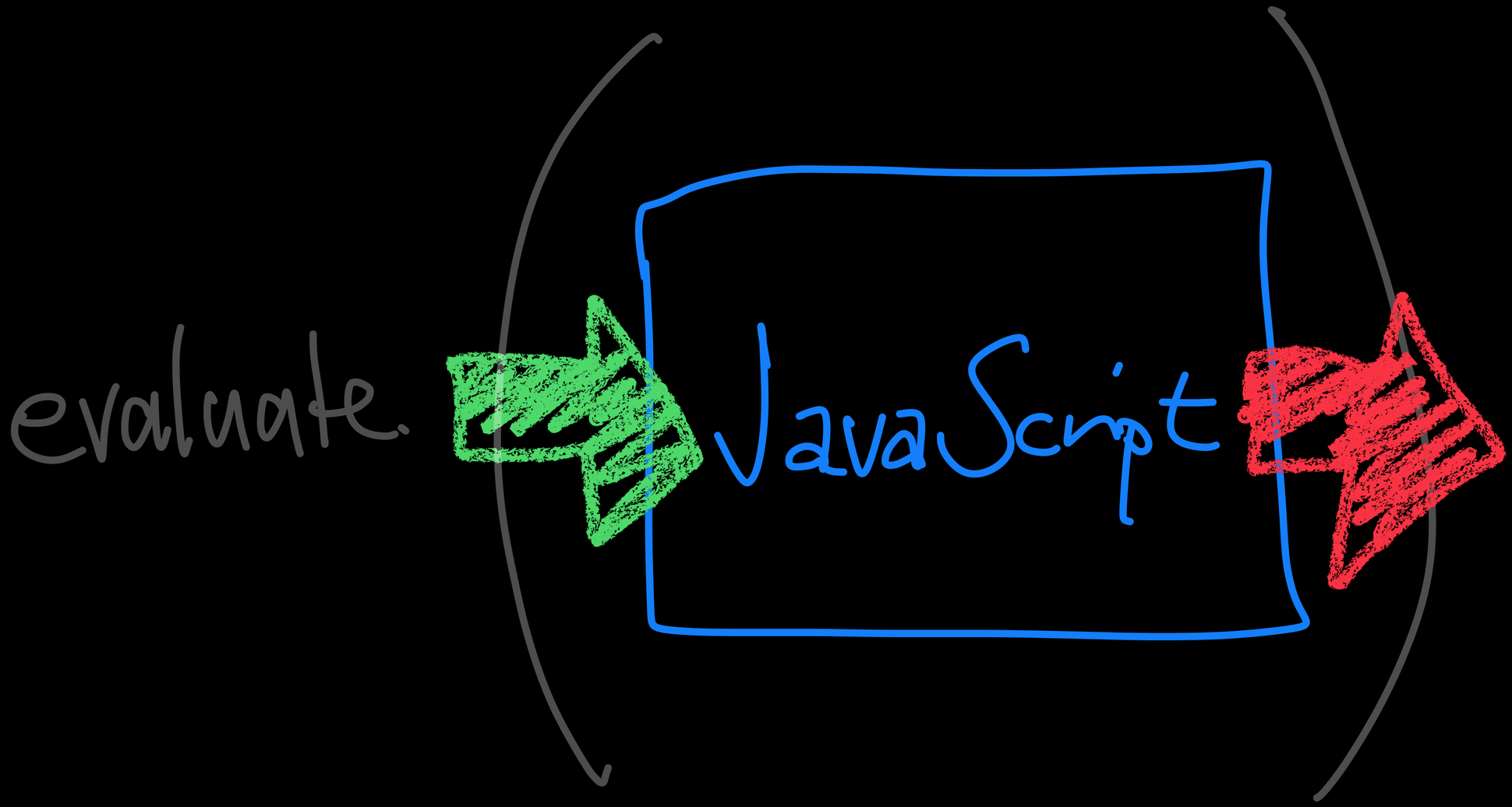
```
class func current() -> JSContext!
```

Returns the context currently executing JavaScript code.

JSContext

evaluate.




evaluate.
JavaScript

Functional

IO

Functional

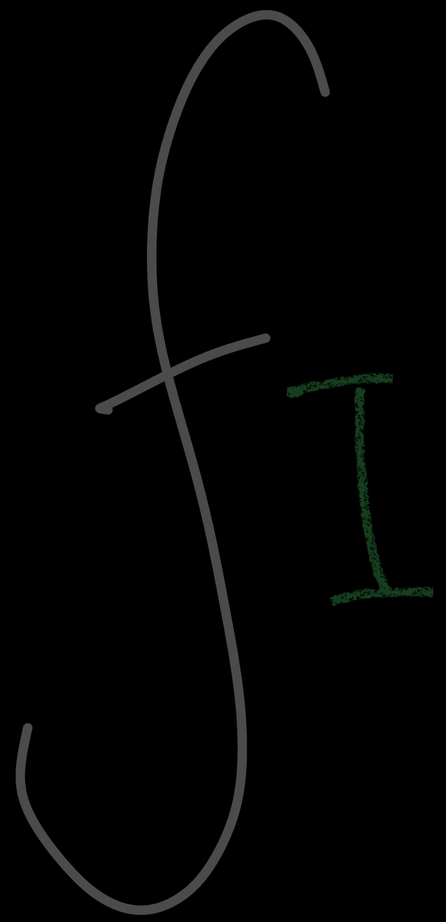
evaluate(String)

```
const aValue = input();

. . .

output(anotherValue);
```

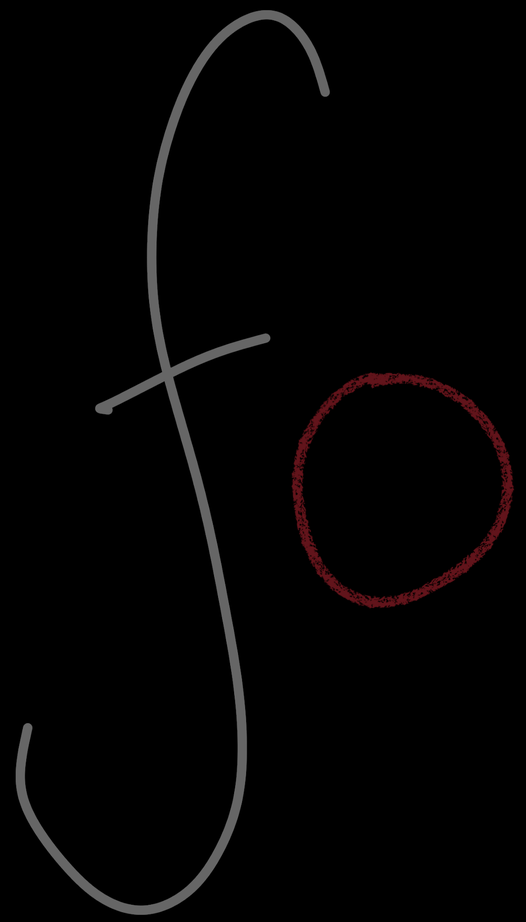
Time for Code!

```
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]

    ...
}
```



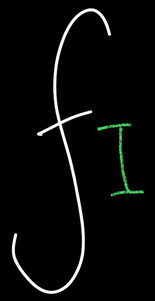
```
struct Manifest {
    ...
    struct Input {
        enum Selection: String, Hashable {
            case selected
            case all
        }

        let notes: Set<Selection>
        let text: Set<Selection>
    }
    ...
}
```

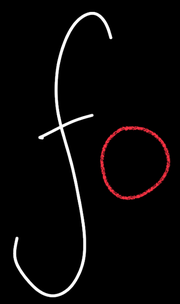


```
struct Manifest {
    ...
    struct Output {
        enum File {
            case change(Filename)
            case new
        }

        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
    }
}
```

```swift
protocol NoteContentCollectorJSExports: JSExport {
    var filename: String? { get }
    var content: String { get set }
}

protocol InsertCollectorJSExports: JSExport {
    var text: String { get set }
}
```

```swift
protocol OutputCollectorJSExports: JSExport {
    var newFile: NoteContentCollectorJSExports? { get }
    var changeFile: NoteContentCollectorJSExports? { get }
    var insert: InsertCollectorJSExports? { get }
}
```

```swift
protocol NoteContentCollectorJSExports: JSExport {
    var filename: String? { get }
    var content: String { get set }
}


protocol InsertCollectorJSExports: JSExport {
    var text: String { get set }
}


protocol OutputCollectorJSExports: JSExport {
    var newFile: NoteContentCollectorJSExports? { get }
    var changeFile: NoteContentCollectorJSExports? { get }
    var insert: InsertCollectorJSExports? { get }
}
```

```swift
struct Manifest {
    ...
    struct Output {
        enum File {
            case change(Filename)
            case new
        }

        enum Text {
            case insert
        }

        let file: File?
        let text: Text?
    }
    ...
}
```

```swift
protocol NoteContentCollectorJSExports: JSExport {
    var filename: String? { get }
    var content: String { get set }
}


protocol InsertCollectorJSExports: JSExport {
    var text: String { get set }
}


protocol OutputCollectorJSExports: JSExport {
    var newFile: NoteContentCollectorJSExports? { get }
    var changeFile: NoteContentCollectorJSExports? { get }
    var insert: InsertCollectorJSExports? { get }
}
```

```swift
struct Manifest {
    ...
    struct Output {
        enum File {
            case change(Filename)
            case new
        }

        enum Text {
            case insert
        }

        let file: File?
        let text: Text?
    }
    ...
}
```

```swift
protocol NoteContentCollectorJSExports: JSExport {
    var filename: String? { get }
    var content: String { get set }
}


protocol InsertCollectorJSExports: JSExport {
    var text: String { get set }
}


protocol OutputCollectorJSExports: JSExport {
    var newFile: NoteContentCollectorJSExports? { get }
    var changeFile: NoteContentCollectorJSExports? { get }
    var insert: InsertCollectorJSExports? { get }
}
```

Time for a demo! + playground

But first:

questions?

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