



Let's build a VS Code extension for automated refactorings

WeAreDevs JS Congress 2021 – Workshop

👉 bit.ly/vscode-extension-slides (slides)

Slides

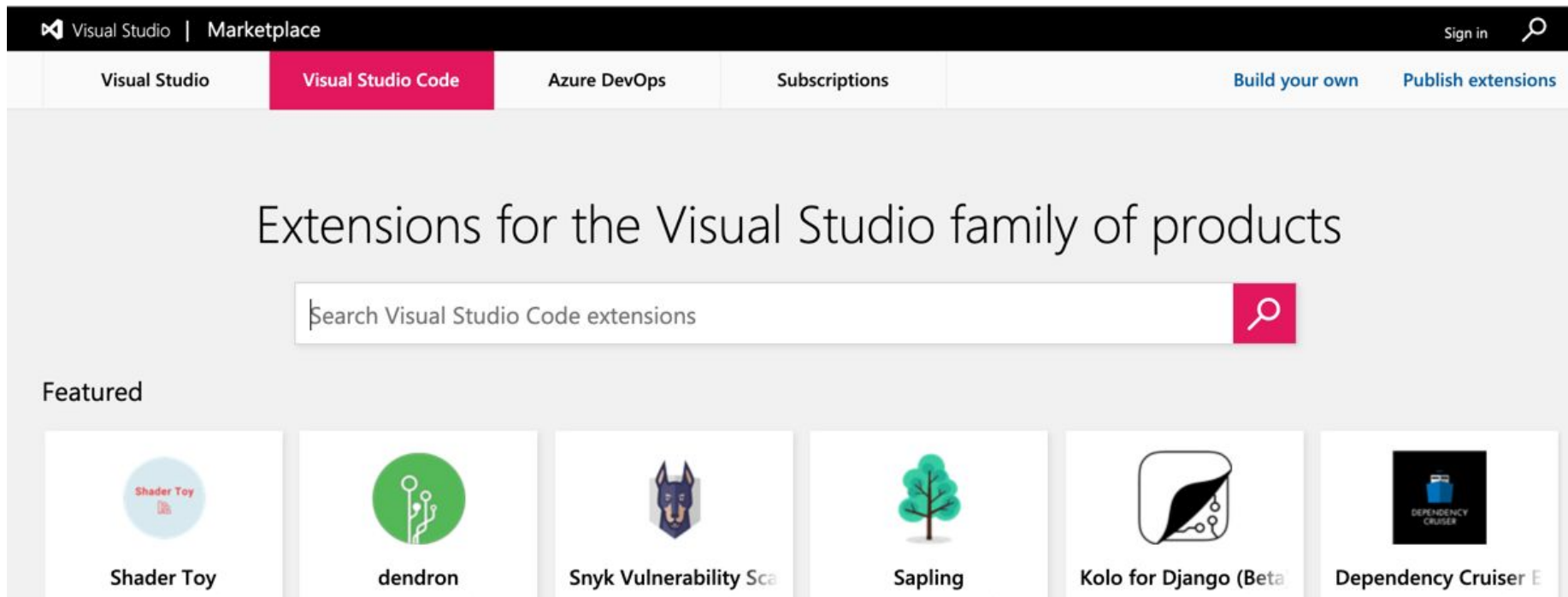


bit.ly/vscode-extension-slides

—

You can build your own extension, using JS/TS

👉 code.visualstudio.com/api/references/vscode-api (the documentation is helpful)



**Let's see how to build your own
extension to enhance your editor!**



Hi, I'm Nicolas 🖐️



@nicoespeon

Senior Software Developer
@ Centered

understandlegacycode.com



Author of Abracadabra 
(refactoring extension for JS/TS in VS Code)

Let's do it for real? 

—



A repository to follow along

👉 bit.ly/vscode-extension-code



Checkout the **tags** I give you 😊

You may code along... or simply follow

`git stash` if necessary

Let's get Centered

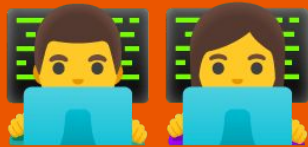
We'll use [Centered.app](#) as we work to make sure we're in time!



Centered



Let's do it!



```
git checkout start
```

Scaffolding



“Your First Extension”

👉 code.visualstudio.com/api/get-started/your-first-extension

The “Getting Started” docs is really good.

It would guide you until the publication of your extension 🏆

Yeoman will scaffold a new extension for you

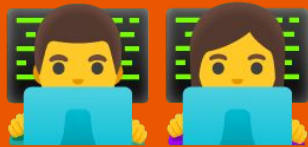
THE WEB'S
SCAFFOLDING
TOOL FOR
MODERN
WEBAPPS



Quickly get started with an
up-to-date infrastructure

 yeoman.io

Let's do it!



```
git checkout 1-yeoman
```

Interacting with the code



VS Code API Editor allows you to read code

👉 code.visualstudio.com/api/references/vscode-api

TypeScript helps getting familiar with the API

`window.activeTextEditor` returns the currently active [TextEditor](#)

Then you can get the text editor [document](#) to `getText()`



VS Code API Editor allows you to write code

`activeTextEditor.edit()` to edit the code

It takes a callback that provides an [TextEditorEdit](#) with a `replace()` method you can use

We can also use [Range](#) and [Position](#) to adapt the selection



You can configure keyboard shortcuts

👉 code.visualstudio.com/api/references/contribution-points

You can provide different type of “contributions”

[keybindings](#) is the one that allows you to configure a default shortcut

Users may customize it

```
{
  "contributes": {
    "keybindings": [
      {
        "command": "extension.sayHello",
        "key": "ctrl+f1",
        "mac": "cmd+f1",
        "when": "editorTextFocus"
      }
    ]
  }
}
```

Let's do it!



```
git checkout 2-read-write
```

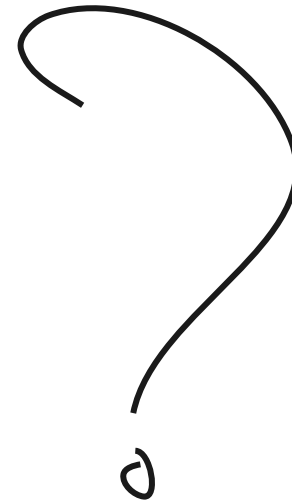
—

Transform the code (AST)



A data structure to represent code

```
console.log("Hello world!")
```





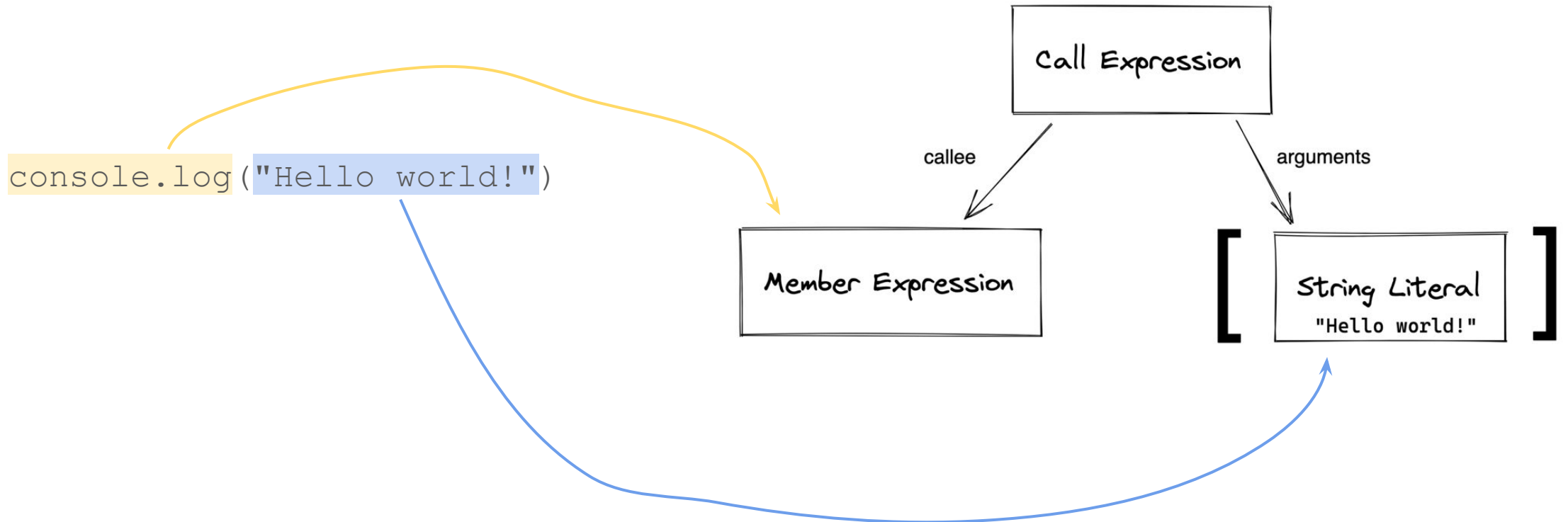
A data structure to represent code

```
console.log("Hello world!")
```

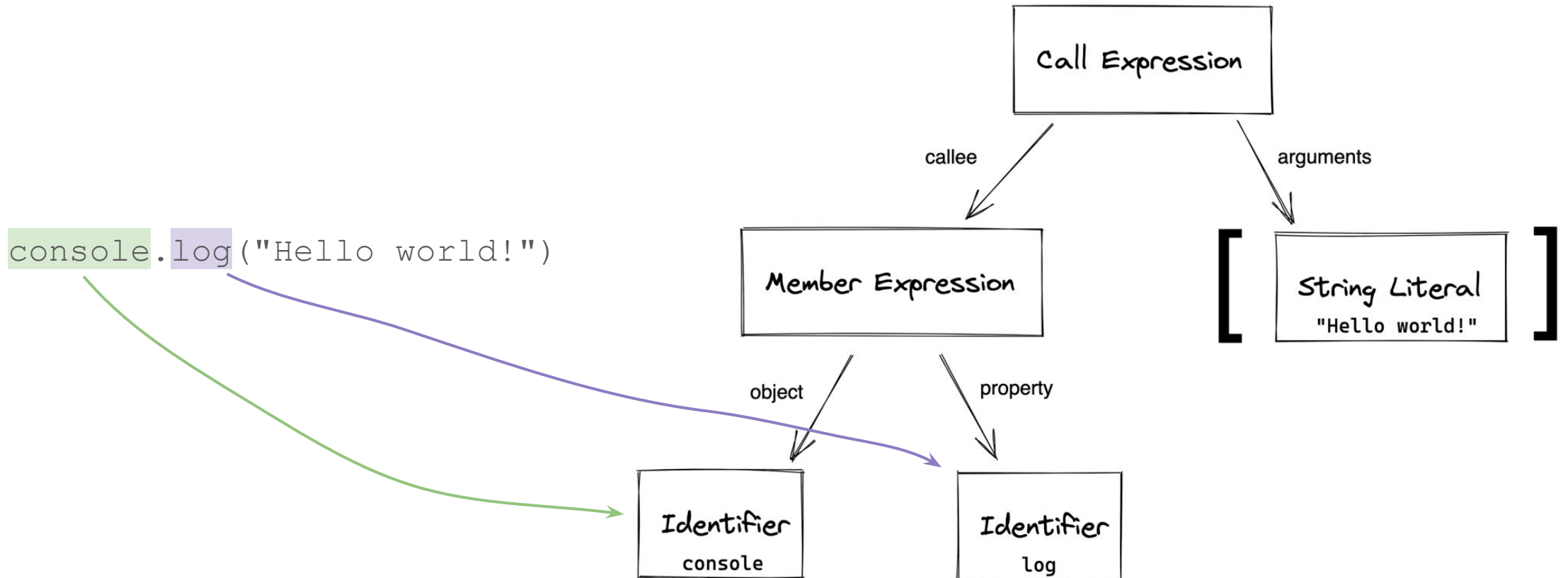


Call Expression

A data structure to represent code

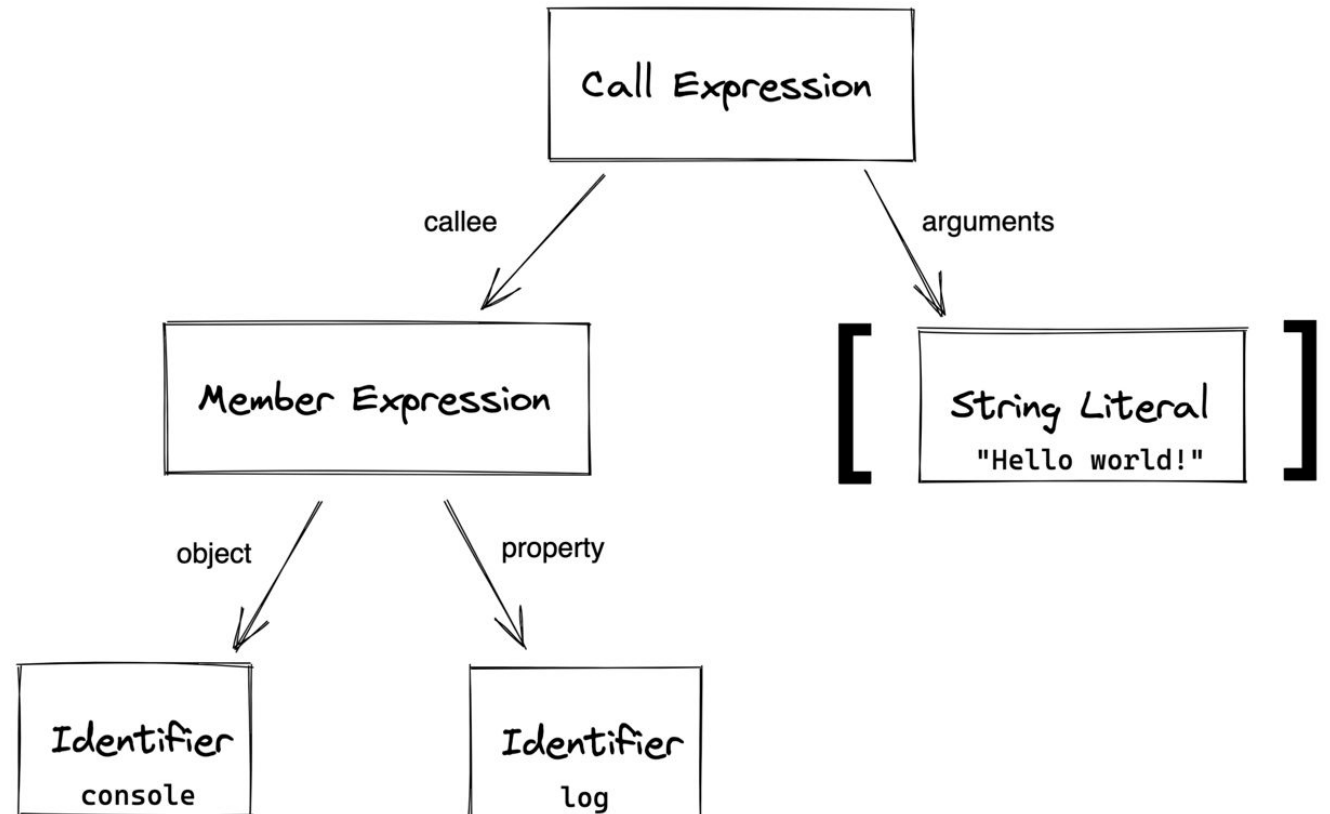


A data structure to represent code



A data structure to represent code

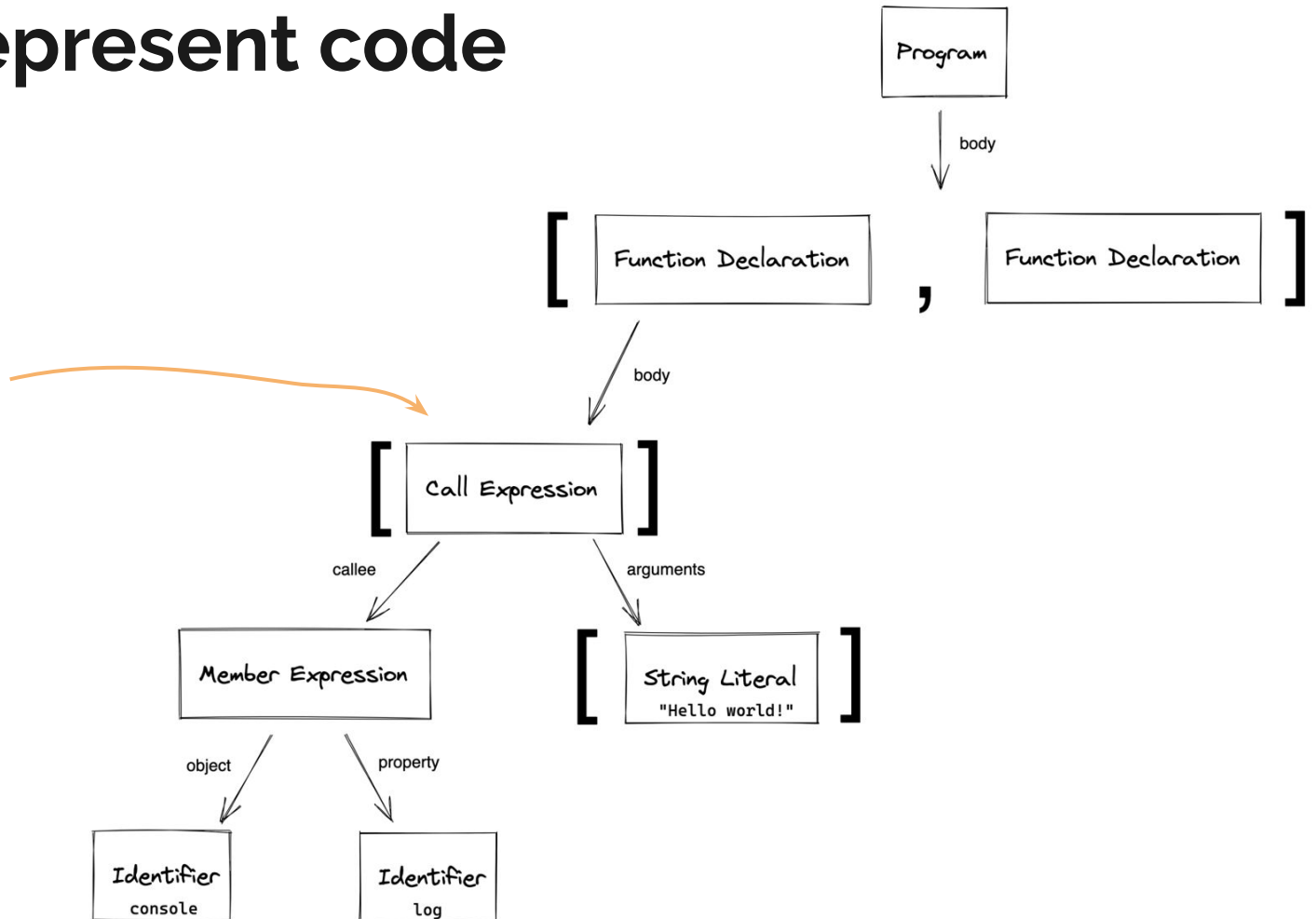
```
console.log("Hello world!")
```



A data structure to represent code

```
function sayHello() {  
  console.log("Hello world!")  
}
```

```
function sayBye() {}
```





The Abstract Syntax Tree (AST)

Tree representation of the structure (abstract syntax) of the code – [see definition](#)

A richer model of the source code we can:

- traverse
- transform
- generate

👉 astexplorer.net is super useful to visualize the AST of some code

One code, many ASTs

Many ASTs can represent the same code. Details vary.

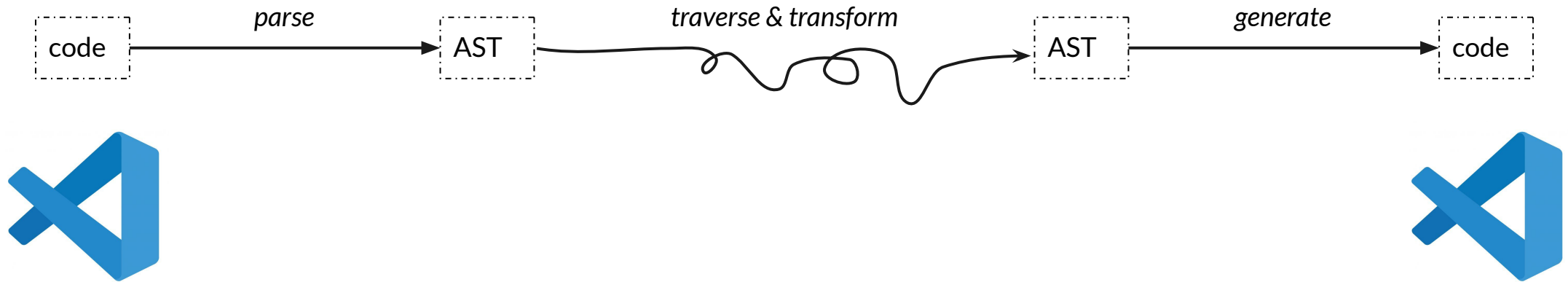
=> **Many** solutions exist (parsers). You may build your own. 🙌

Babel is a JavaScript compiler. To do so, it needs to parse code into AST, and vice-versa!

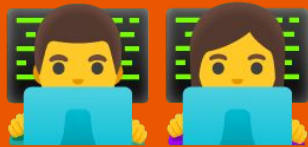
Useful libs: [@babel/parser](#), [@babel/traverse](#), [@babel/types](#) and [@babel/generator](#)



Parse code into AST. Print AST into code.



Let's do it!



```
git checkout 3-parse-ast
```

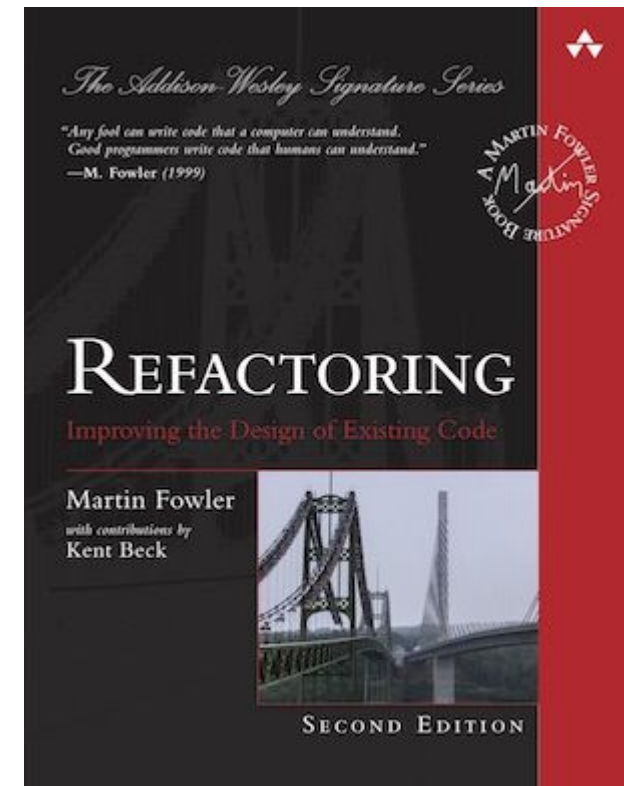
—

Refactorings

What I mean when I say “Refactoring”?

Refactoring (noun): a change made to the internal structure of software to make it easier to understand and **cheaper to modify without changing its observable behavior**.

— *Martin Fowler*






Make the change easy, *then* do the easy change

Ways to refactor JavaScript code fast & safe:


- Automated Refactorings
- Atomic Refactorings (manual)
- Automated Tests

 I'm building an interactive course to teach you **Atomic Refactoring** moves in JavaScript

Coming  early 2022

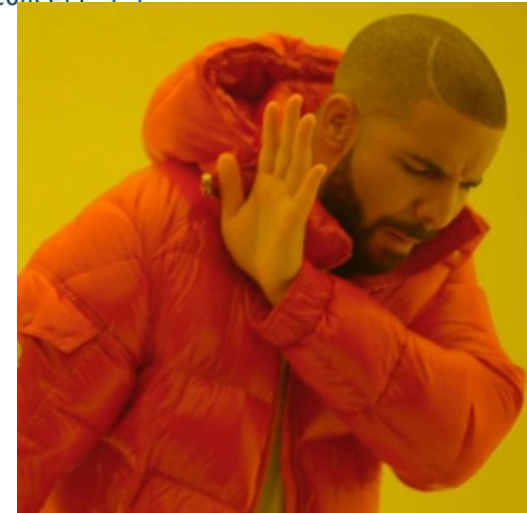


refactoringjavascript.dev

 *Subscribe, so you don't miss out*

The problem with nesting (bumpy road smell)

```
updateQuality() {  
  for (let i = 0; i < this.items.length; i++) { +1  
    if (this.items[i].name !== 'Aged Brie' && this.items[i].name !== 'Backstage passes to a TAFKAL80ETC concert') {  
      if (this.items[i].quality > 0) {  
        if (this.items[i].name !== 'Sulfuras, Hand of Ragnaros') {  
          this.items[i].quality = this.items[i].quality - 1;  
        }  
      }  
    } else { +2  
      if (this.items[i].quality < 50) { +3  
        this.items[i].quality = this.items[i].quality + 1;  
        if (this.items[i].name === 'Backstage passes to a TAFKAL80ETC concert') { +4  
          if (this.items[i].sellIn < 11) {  
            if (this.items[i].quality < 50) {  
              this.items[i].quality = this.items[i].quality + 1;  
            }  
          }  
          if (this.items[i].sellIn < 6) { +5  
            if (this.items[i].quality < 50) {  
              this.items[i].quality = this.items[i].quality + 1; ←  
            }  
          }  
        }  
      }  
    }  
  }  
}
```



... need to tweak code while keeping
5 concepts in our working memory 🤯

Guard Clauses to the rescue!

```
updateQuality() {  
  for (let i = 0; i < this.items.length; i++) { +1  
    if (  
      this.items[i].name !== "Aged Brie" &&  
      this.items[i].name !== "Backstage passes to a TAFKAL80ETC concert"  
    ) { ...  
      }  
  
    if (this.items[i].quality ≥ 50) continue;  
  
    this.items[i].quality = this.items[i].quality + 1;  
    if (this.items[i].name !== "Backstage passes to a TAFKAL80ETC concert") continue;  
  
    if (this.items[i].sellIn < 11) {  
      if (this.items[i].quality ≥ 50) continue;  
      this.items[i].quality = this.items[i].quality + 1;  
    }  
  
    if (this.items[i].sellIn ≥ 6) continue;  
    if (this.items[i].quality ≥ 50) continue;  
  
    this.items[i].quality = this.items[i].quality + 1; ←  
  }  
}
```




Early exits = not in your working memory anymore 😊

Refactoring: Introduce Guard Clauses

```
function getPayAmount() {  
  let result;  
  if (isDead) {  
    result = deadAmount();  
  } else {  
    if (isSeparated) {  
      result = separatedAmount();  
    } else {  
      if (isRetired) {  
        result = retiredAmount();  
      } else {  
        result = normalPayAmount();  
      }  
    }  
  }  
  return result;  
}
```

traverse & transform

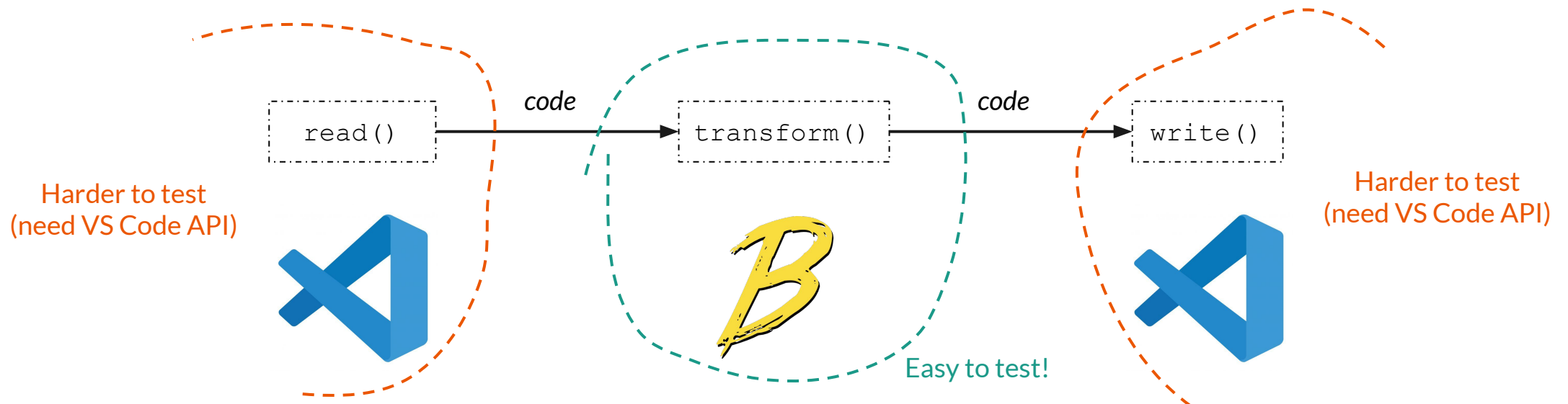


```
function getPayAmount() {  
  if (isDead) {  
    return deadAmount();  
  }  
  if (isSeparated) {  
    return separatedAmount();  
  }  
  if (isRetired) {  
    return retiredAmount();  
  }  
  return normalPayAmount();  
}
```

Let's automate this... guided by tests!

We want tests that make us **code faster**:

- High-level enough so it doesn't break on refactoring
- Not dependent on things that are hard/slow/fragile to test



Let's package it! 

```
git checkout 4-implement-refactoring
```


Package & Publish your extension



Follow the docs

 code.visualstudio.com/api/working-with-extensions/publishing-extension

You need to:

- **Install vsce** (tip: install it locally to your project & provide npm scripts to run it)
- [Create an org](#) in Azure DevOps and get a [Personal Access Token](#)
- [Create a publisher](#) on VS Marketplace
- Configure your extension for publication
- Publish! 



Just packaging the extension

 code.visualstudio.com/api/working-with-extensions/publishing-extension#packaging-extensions

`vsce package` in the root folder to generate a **VSIX** file

Then anyone can install it with:

`code --install-extension my-extension-0.0.1.vsix`

YOU DID IT 🙌 🎉 🙌

`git checkout end`

—



What you learned here 🎓

- ★ Use VS Code API to
 - Read code
 - Write code
- ★ Transform code
 - Parse code into AST
 - Traverse and transform the AST
 - Print AST into code
- ★ A bit about
 - **#testing** and **#softwareArchitecture**
 - **#refactoring** and making code easier to maintain
 - **#productivity** and staying focused
- ★ Package the extension so you can use & distribute it

Going further



What's next?

- Handle more cases (e.g. throw, continue, no explicit return but no sibling neither, etc.)
- Finer selection (e.g. only the `IfStatement` below cursor)
- Expose it as a [Quick Fix](#) 💡

Regarding tests:


- Most of the logic is tested
- Blind spots on the VS Code API => we can write a few tests to cover these
- Advanced scenario may require the editor in `transform()` => [Hexagonal Architecture](#)



Learn to Refactor JS safely & meet deadlines

Ways to refactor JavaScript code fast & safe:


- Automated Refactorings
- **Atomic Refactorings (manual)**
- Automated Tests

 I'm building an interactive course to teach you **Atomic Refactoring** moves in JavaScript

Coming  early 2022



refactoringjavascript.dev

 *Subscribe, so you don't miss out*



Wanna try Centered for yourself?

You can use the core features for free!



Centered


 Use **WEAREDEVS2021** before 2022 to get 1 month of FREE Premium 



Wanna continue the fun?

[Abracadabra](#) is open-source. More than 35 automated refactorings.

First-contributors-friendly, [check out the guide](#) (34+ contributors).



Abracadabra, refactor this!

Nicolas Carlo | 📄 19,316 installs | ★★★★★ (17) | Free

Automated refactorings for VS Code, in JavaScript and TypeScript.

[Install](#) [Trouble Installing?](#)



Useful links

👉 bit.ly/vscode-extension-slides to get the slides

👉 bit.ly/vscode-extension-code to follow along with the source code

👉 twitter.com/nicoespeon to contact me and ask questions (DMs open)

