Swift プロジェクト

開発アプリについて

目的

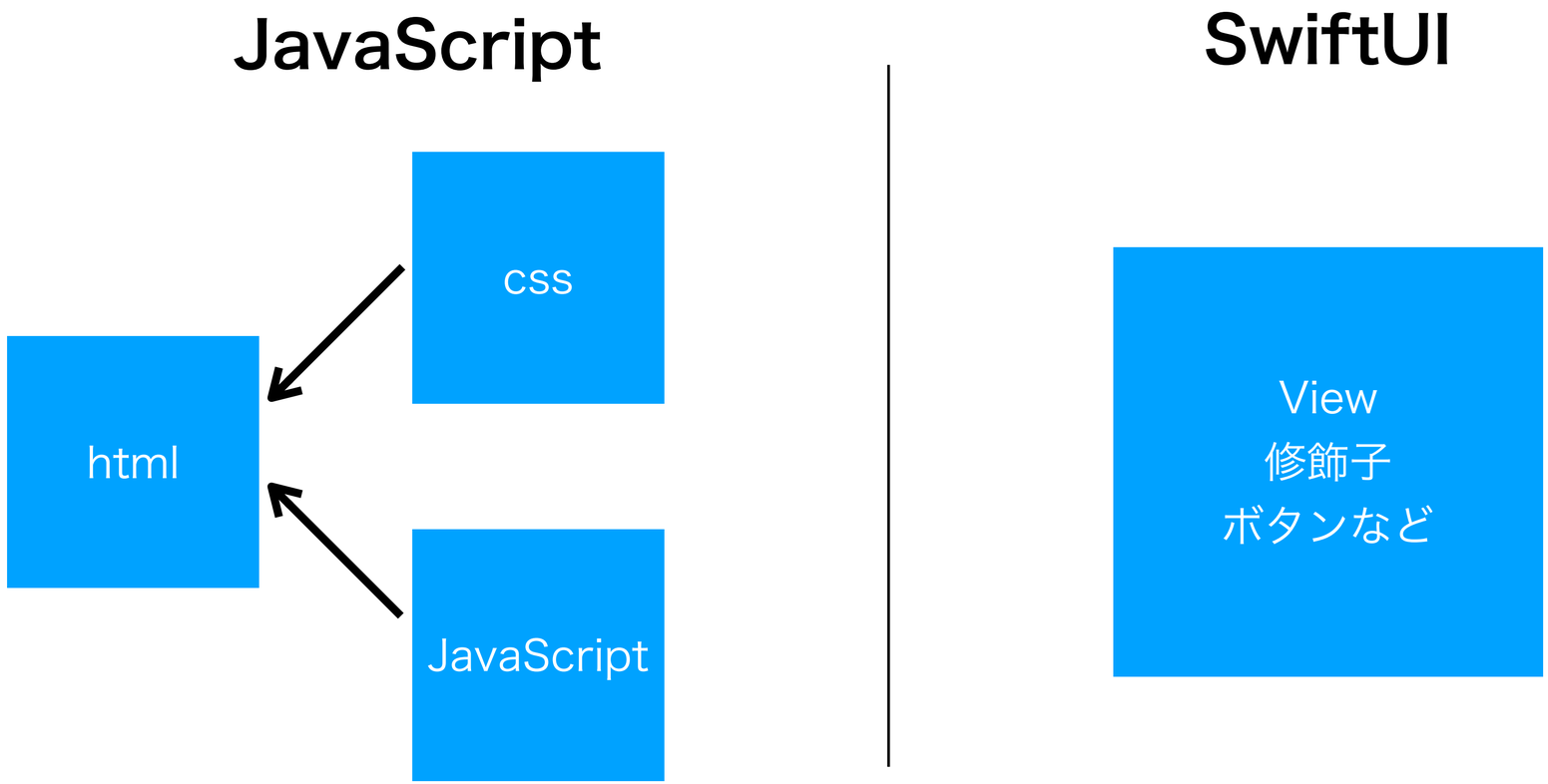
- 新しい言語 Swift の学習とアプリを実装
- JavaScriptとSwiftの比較

Soloプロジェクトと同様のアプリ開発
JavaScriptとSwiftの違いを確認

アプリ：お店へGo!! (iOS版)

Swiftについて学んだこと

1. 開発手法が2通り：storybord, **swiftUI ←こちらを選択**
2. swiftUIだけで、html, css, JavaScript相当を簡単に実装できる



アプリのソースコード


🛒お店へGo!!
追加
買物リスト
Aaaaaaaaaa
Bbbbbbbbb
Ccccccccc
Ddddddddd
選択した商品を買ったよ！

アプリのソースコード

ContentView.swift

ContentView body

VStack

Text

HStack

TextEditor

Button

VStack

Image

Text

HStack

Image

Text

ScrollView

VStack

Divider

ForEach

HStack

Button

Image

Divider

Text

Divider

Button

HStack

Image

Text

アプリのソースコード

ContentView.swift

ContentView body

VStack
Text

HStack
TextEditor
Button

VStack
Image
Text

HStack
Image
Text

ScrollView

VStack
Divider

ForEach

HStack
Button
Image
Divider
Text

Divider

Button

HStack
Image
Text

```
addItemsToList(
    inputText: &inputText,
    shoppingItems: &shoppingItems
)
```

JavaScriptとの違い
関数に引数を渡すときに引数名と値が必要

‘Button’ を押すと ‘addItemsToList’ 関数に ’inputText(入力文字), shoppingItems’ の情報を渡し実行する

```
func addItemsToList(
   inputText: inout String, shoppingItems: inout [ShoppingItem]) {
   let lines = inputText.split(separator: "\n")
   let newItems = lines.map { ShoppingItem(text: String($0))}
   inputText = ""
}
```

JavaScriptとの違い
関数も引数名と値が必要

関数は、改行があれば改行単位で、‘ShoppingItem’ を生成し、’shopingItems’ に追加する

```
struct ShoppingItem: Identifiable {
 let id = UUID()
 var text: String
 var isTapped: Bool = false
}
```

```
@State var inputText: String = ""
@State var shoppingItems: [ShoppingItem] = []
```

‘shoppingItems’ に ‘ShoppingItem’ を追加された状態

```
[
 swift_project_ohta_gs.ShoppingItem(
  id: 1B514023-4F29-4805-BEFF-782FC8F4AB1B,
  text: “商品1",
  isTapped: false
 ),
 swift_project_ohta_gs.ShoppingItem(
  id: E3E71628-76AE-4E81-AF07-0BFC137A42CC,
  text: “商品2",
  isTapped: false
 )
]
```

JavaScriptとの違い
定数と変数の定義が微妙

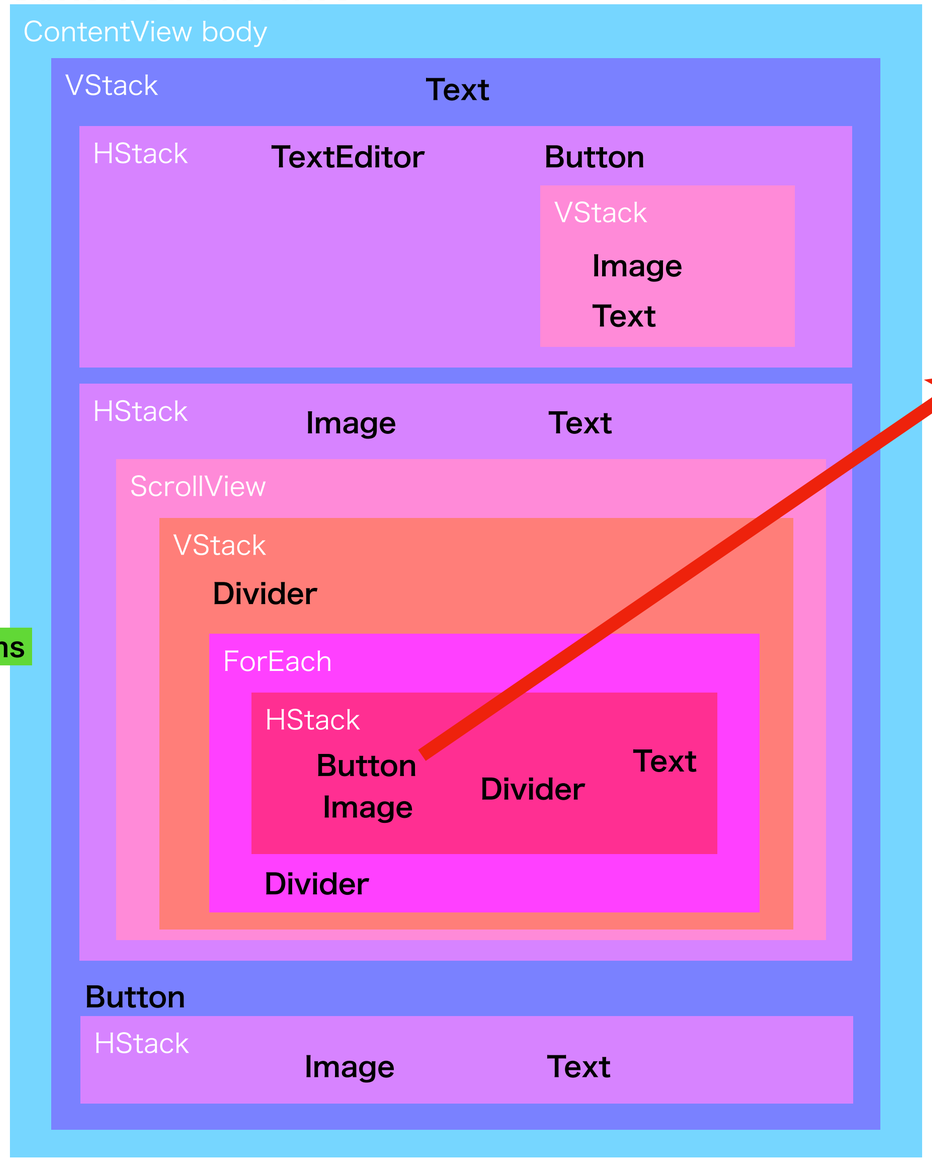
JS
定数：const
変数：let

Swift
定数：let
変数：var

アプリのソースコード

shoppingItems

ContentView.swift




```
deleteItem(arg: index, from: &shoppingItems)
```

‘Button’ を押すと ‘deleteItem’ 関数に ’index(番号), shoppingItems’ の情報を渡し実行する

関数は、’shopingItems’ から ’index’ の番号を削除する

```
func deleteItem(
    arg index: Int, from shoppingItems: inout [ShoppingItem]) {
    shoppingItems.remove(at: index)
}
```

アプリのソースコード

shoppingItems

ContentView.swift

ContentView body

- VStack
  - Text
  - HStack
    - TextEditor
    - Button
      - VStack
        - Image
        - Text
  - HStack
    - Image
    - Text
    - ScrollView
      - VStack
        - Divider
        - ForEach
          - HStack
            - Button
              - Image
            - Divider
            - Text
          - Divider
  - Button
    - HStack
      - Image
      - Text

```
toggleItemSelection(arg: index, from: &shoppingItems)
```

‘文字’ をタップすると ‘toggleItemSelection’ 関数に ’index(番号), shoppingItems’ の情報を渡し実行する

関数は、’shopingItems’ から ’index’ の番号の要素の、
’isTapped’ のブーリアン値を反転させる
true → false
false → true

```
func toggleItemSelection(
    arg index: Int, from shoppingItems: inout [ShoppingItem]) {
  shoppingItems[index].isTapped.toggle()
}
```

アプリのソースコード

ContentView.swift

shoppingItems

ContentView body

VStack
Text
HStack
TextEditor
Button
VStack
Image
Text
HStack
Image
Text
ScrollView
VStack
Divider
ForEach
HStack
Button
Image
Divider
Text
Divider
Button
HStack
Image
Text

```
removeAllSelectedItems(shoppingItems: &shoppingItems)
```

‘Button’ を押すと ‘removeAllSelectedItems’ 関数に
’shoppingItems’ の情報を渡し実行する

関数は、’shopingItems’ から ’isTapped’ が ‘True’ の要素を削除する

```
func removeAllSelectedItems(
shoppingItems: inout [ShoppingItem]) {
    shoppingItems.removeAll(where: { $0.isTapped })
    }
```

主な違い

		JavaScript	SwiftUI
コンソール		console.log(表示したい変数名など)	print(表示したい変数名など)
変数	定数	const 定数名 = 値 const name = “Bob”	let 定数名 = 値 let name = “Bob”
	変数	let 変数名 = 値 let name = “Bob”	var 変数名 = 値 var name = “Bob”
	型宣言	TypeScriptで記述 let 変数名: 型 = 値 let name: string = "John";	var 変数名: 型 = 値 var name: String = “John”
	文字列に変数埋込み	`${変数名}` const name = “JS”; console.log(`私の名前は${name}です`);	\(変数名) let name = “Swift” print(“私の名前は\(name)です”)

条件分岐		If (条件1) {
		// 条件1がtrueの処理
		} elseif 条件2 {
		// 条件2がtrueの処理
		} else {
		// それ以外の処理
		};
			If 条件1 {
			// 条件1がtrueの処理
			} else if 条件2 {
			// 条件2がtrueの処理
			} else {
			// それ以外の処理
			}

主な違い

		JavaScript	SwiftUI
配列	定義	const 配列名 = [要素1, 要素2, 要素3];	const 配列名 = [要素1, 要素2, 要素3]
	要素参照方法	console.log(変数名[1]); // 要素2	print(配列名[1]) // 要素2
	要素の追加	配列名.push(追加要素);	配列名.append(追加要素) 配列要素の追加
	要素の削除	配列名.pop();	配列名.remeove(at: index) 配列要素の削除

主な違い

		JavaScript	SwiftUI

関数	関数の定義	function 関数名(引数1, 引数2) {
		// 処理内容
		return 戻り値
		}
		
		`function hello(name, greetingNum) {`
		`var greeting`
		`  if (greetingNum === 1) {`
		`  greeting = "こんにちは"`
		`  } else {`
		`  greeting = "ありがとう"`
		`  }`
		`return \`${greeting}、${name}さん！\``
		`}`
			func 関数名(引数名1: 引数1の型, 引数名2: 引数2の型) -> 戻り値の型 {
			// 処理内容
			return 戻り値
			}
			
			`func hello(name: String, greetingNum: Int) -> String {`
			`var greeting: String`
			`  if greetingNum == 1 {`
			`  greeting = “こんにちは"`
			`  }`
			`  else {`
			`  greeting = "ありがとう"`
			`  }`
			`return "\(greeting)、\(name)さん！"`
			`}`

	関数の呼出し	関数名(引数1, 引数2)
		`console.log(hello(“Bob", 1))`
		`console.log(hello("Bob", 2))`
			関数名(引数名1: 引数1, 引数名2: 引数2)
			`print(hello(name: "Bob", greetingNum: 1))`
			`print(hello(name: "Bob", greetingNum: 2))`

	JavaScript	SwiftUI

反復処理	配列.forEach(引数 => {
	// 処理内容
	}
		ForEach(構造体の配列) { 変数 in
		// UI部品
		}

```
const array = ["Apple", "Banana", "Melon"];

array.forEach(value => {
  console.log(value);
});
```

```
import SwiftUI

struct Fruit: Identifiable {
    let id = UUID()
    let name: String
}

struct ContentView: View {
    let fruits = [
        Fruit(name: "Apple"),
        Fruit(name: "Banana"),
        Fruit(name: "Melon")
        ]

    var body: some View {
        List {
            ForEach(fruits) { fruit in
                Text(fruit.name)
            }
        }
    }
}
```

🎥 Demo 30s

END