

Key points

- Combine is a declarative, reactive framework for processing asynchronous events over time. It aims to solve existing problems, like unifying tools for asynchronous programming, dealing with mutable state and making error handling a starting team player.
- Combine revolves around three main types: **publishers** to emit events over time, **operators** to asynchronously process and manipulate upstream events and **subscribers** to consume the results and do something useful with them.

SINK operator:

it simply provides an easy way to attach a subscriber with closures to handle output from a publisher

```
var subscriptions = Set<AnyCancellable>()
```

```
let just = Just("Hello world!")
_ = just
    .sink(
        receiveCompletion: {
            print("Received completion", $0)
        },
        receiveValue: {
            print("Received value", $0)
        })
    .store(in: subscriptions)
```

Output:

```
Received value Hello world!
Received completion finished
```

ASSIGN operator:

the built-in `assign(to:on:)` operator enables you to assign the received value to a KVO-compliant property of an object.

```
func exampleOfAssign() {  
    // 1  
    class SomeObject {  
        var value: String = "" {  
            didSet {  
                print(value)  
            }  
        }  
    }  
    let object = SomeObject()  
    let publisher = ["Hello", "world!"].publisher  
    _ = publisher  
        .assign(to: \.value, on: object).cancel()  
}
```

Output:

Hello
World!

Code Explanation:

1. Define a class with a property that has a `didSet` property observer that prints the new value.
2. Create an instance of that class.
3. Create a publisher from an array of strings.
4. Subscribe to the publisher, assigning each value received to the value property of the object.

FUTURE:

- A Future is a publisher that will eventually produce a single value and finish, or it will fail. It does this by invoking a closure when a value or error is made available, and that closure is referred to as a promise
- Promise is a type alias to a closure that receives a Result containing either a single value published by the Future, or an error

CODE:

```
var futureSubscription: AnyCancellable?  
func exampleOfFuture() {  
    let ftr = Future<String, Never> { promise in  
        DispatchQueue.main.asyncAfter(deadline: .now() + 2) {  
            promise(.success("world")) /// delay block  
        }  
    }  
    futureSubscription = ftr.sink {  
        print("hello \($0)")  
    }  
}  
exampleOfFuture()
```

Code Explanation:

1. `futureSubscription` is used to store the subscription, if we don't store then code inside the delay block won't execute because the subscription will be deallocated after the end of function execution.

Resource:

1. <https://www.vadimbulavin.com/asynchronous-programming-with-future-and-promise-in-swift-with-combine-framework/>