Sun*

Filtering Operators

Outline

Sun*

1. Getting Started

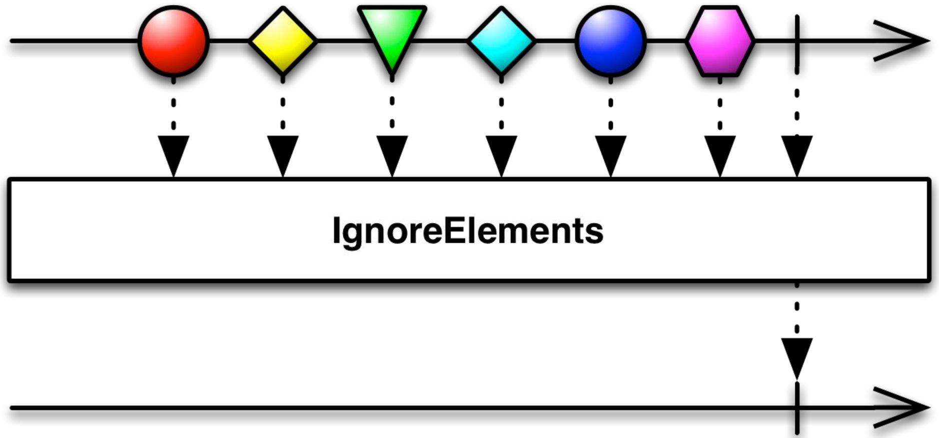
- ❖ Filtering operators allow you to apply conditional constraints to .next events, so that the subscriber only receives the elements it wants to deal with.
- ❖ If you’ve ever use the filter(_:) method in the Swift standard library, you’re already halfway there.

2. Ignoring operators

1. ignoreElements()
2. elementAt()
3. filter()

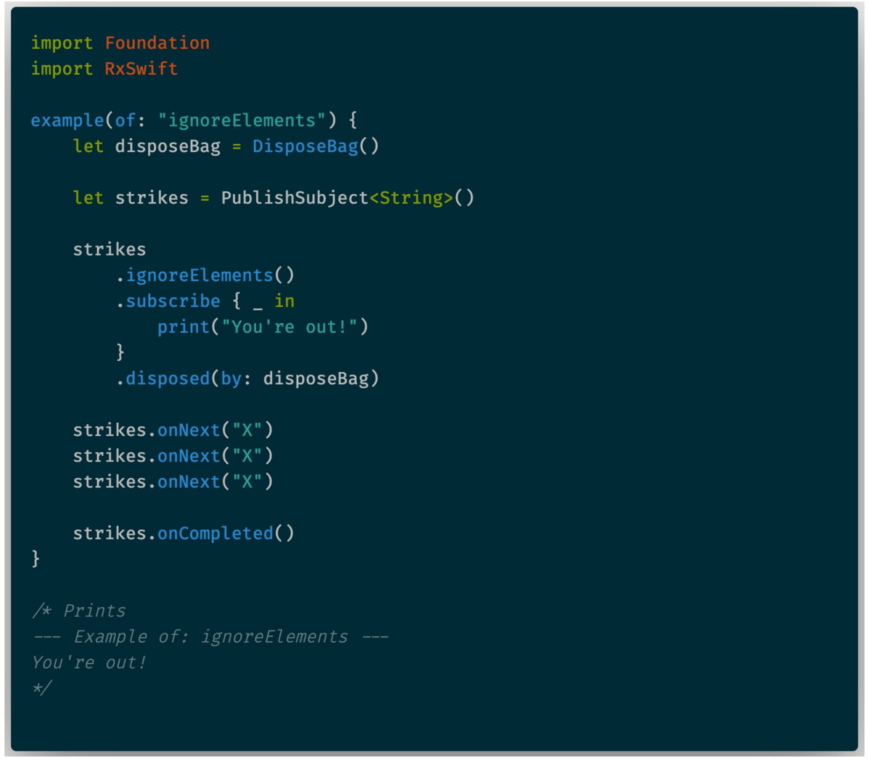
2.1. ignoreElements()

❖ As depicted in the following marble diagram, ignoreElements() will do that: ignore .next event elements. It will, however, allow stop events through, such as .completed or .error events.



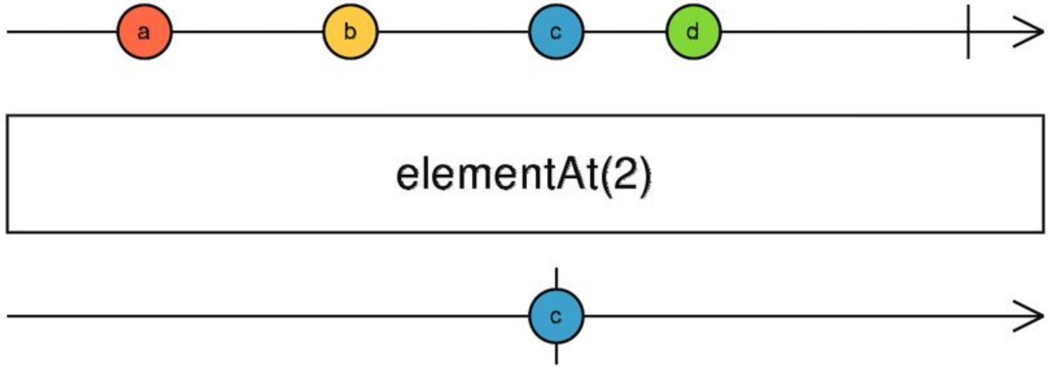
2.1. ignoreElements()

- Example:


```
import Foundation
import RxSwift

example(of: "ignoreElements") {
    let disposeBag = DisposeBag()

    let strikes = PublishSubject<String>()

    strikes
        .ignoreElements()
        .subscribe { _ in
            print("You're out!")
        }
        .disposed(by: disposeBag)

    strikes.onNext("X")
    strikes.onNext("X")
    strikes.onNext("X")

    strikes.onCompleted()
}

/* Prints
--- Example of: ignoreElements ---
You're out!
*/
```

2.2. elementAt()

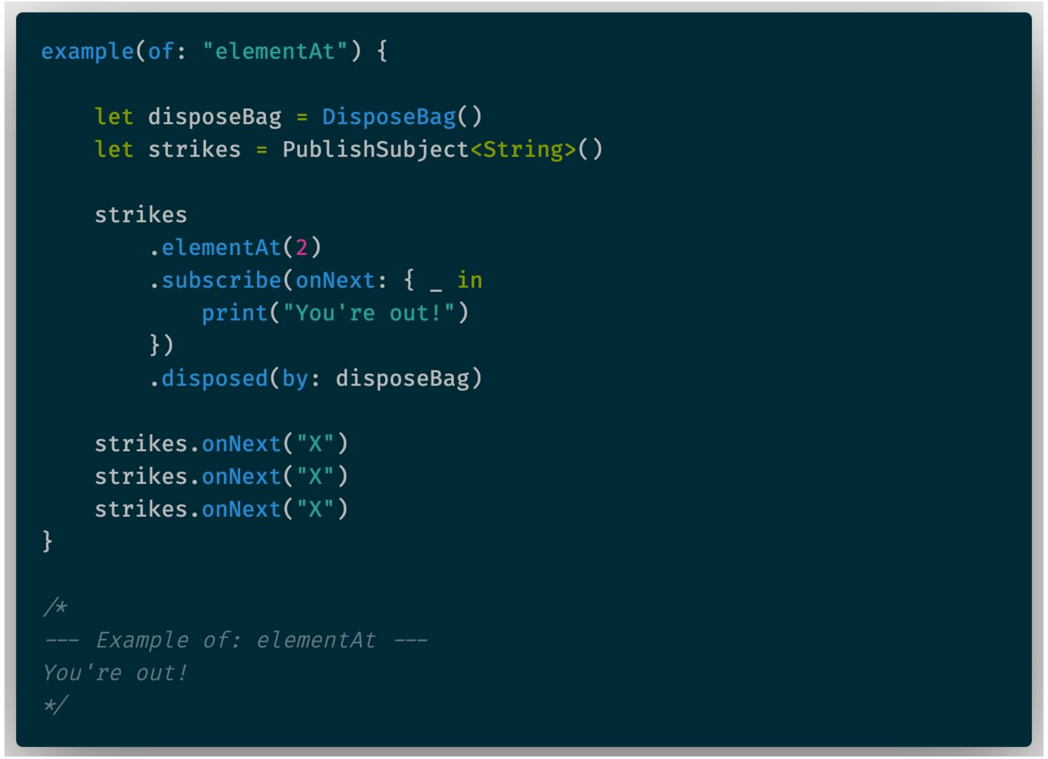
❖ When you only want to handle the *nth* (ordinal) element emitted by an observable, you can use elementAt() which takes the index of the element you want to receive and it ignores everything else.



Sun*

2.2. elementAt()

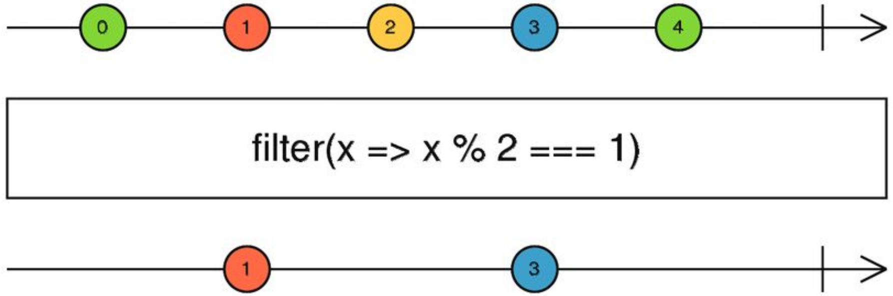
❖ Example:



```
example(of: "elementAt") {

    let disposeBag = DisposeBag()
    let strikes = PublishSubject<String>()

    strikes
        .elementAt(2)
        .subscribe(onNext: { _ in
            print("You're out!")
        })
        .disposed(by: disposeBag)

    strikes.onNext("X")
    strikes.onNext("X")
    strikes.onNext("X")
}

/*
--- Example of: elementAt ---
You're out!
*/
```

2.3. filter()

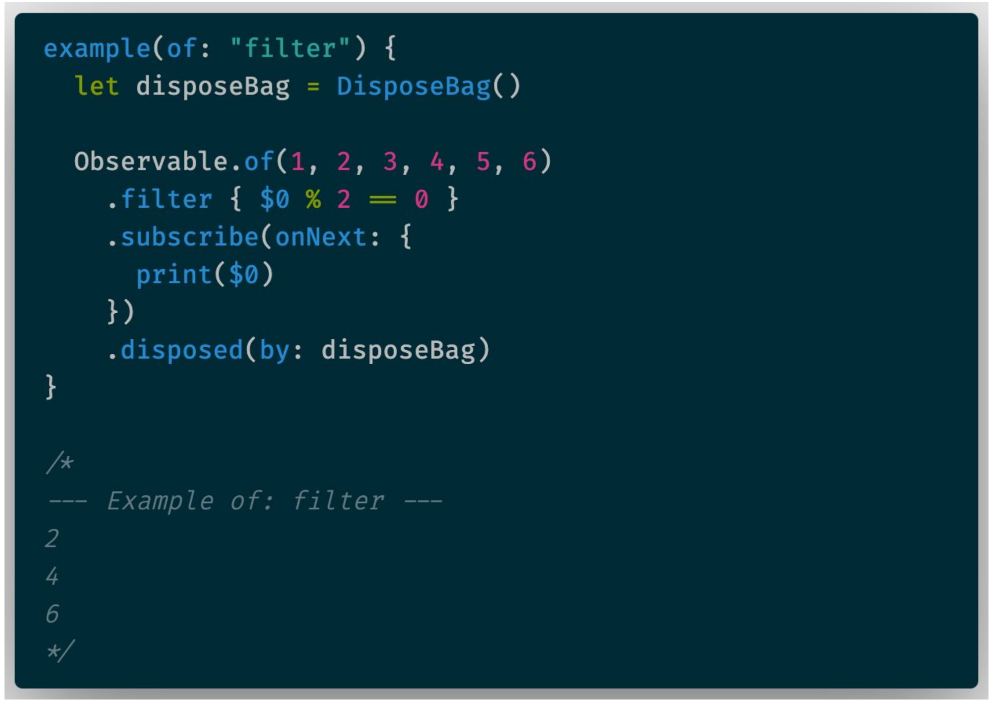
❖ ignoreElements() and elementAt() are filtering elements emitted by an observable. When your filtering needs go beyond all or one, there's filter. It takes a predicate closure, which it applies to every element emitted, allowing through only those elements for which the predicate resolves to true.



Sun*

2.3 filter()

❖ Example:



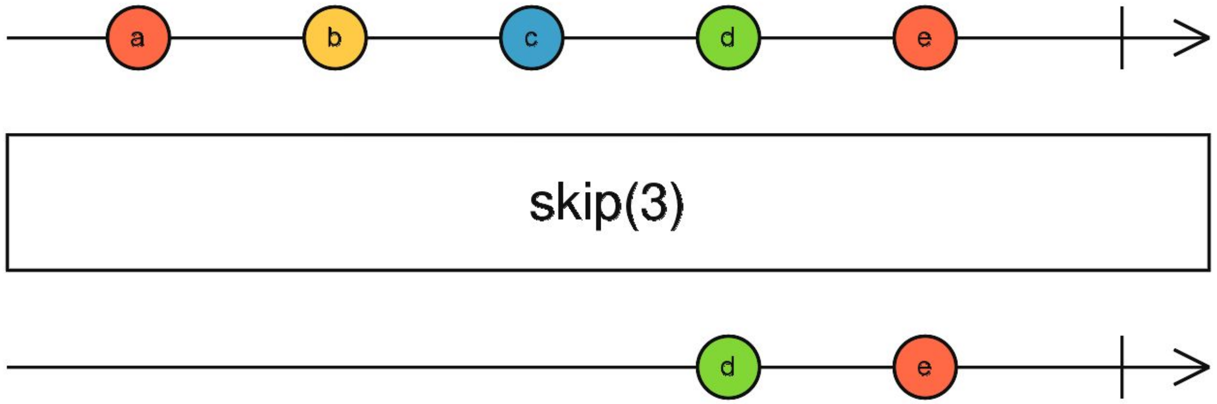
```
example(of: "filter") {
  let disposeBag = DisposeBag()

  Observable.of(1, 2, 3, 4, 5, 6)
    .filter { $0 % 2 == 0 }
    .subscribe(onNext: {
      print($0)
    })
    .disposed(by: disposeBag)
}

/*
--- Example of: filter ---
2
4
6
*/
```

3. Skipping operators

1. skip()
2. skipWhile()
3. skipUntil()

3.1. skip()

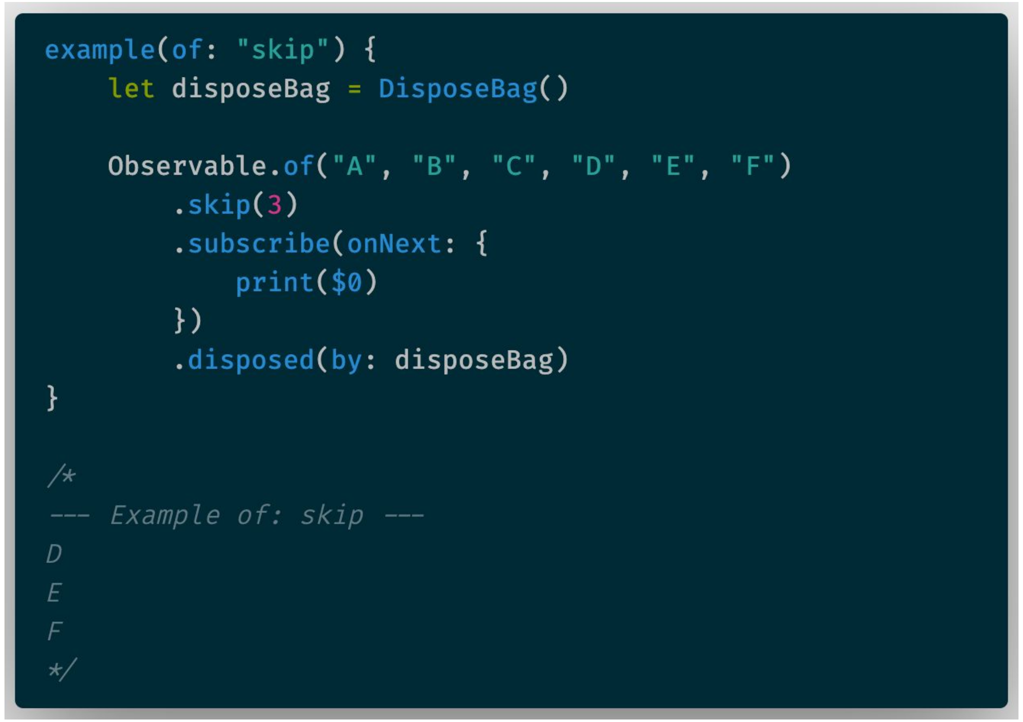
❖ The skip operator allows you to ignore from the 1st to the number you pass as its parameter.



Sun*

3.1. skip()

❖ Example:


```
example(of: "skip") {
    let disposeBag = DisposeBag()

    Observable.of("A", "B", "C", "D", "E", "F")
        .skip(3)
        .subscribe(onNext: {
            print($0)
        })
        .disposed(by: disposeBag)
}

/*
--- Example of: skip ---
D
E
F
*/
```

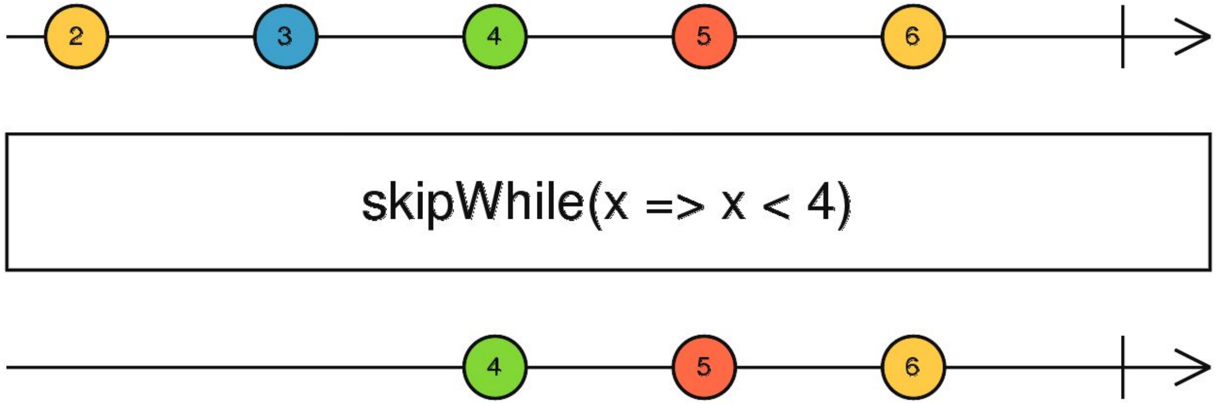
Sun*

3.2. skipWhile()

- Like filter(_:), skipWhile(_:) lets you include a predicate to determine what should be skipped.
- However, unlike filter(_:), which will filter elements for the life of the subscription, skipWhile will only skip until something is not skipped, and then it will let everything else through from that point on.

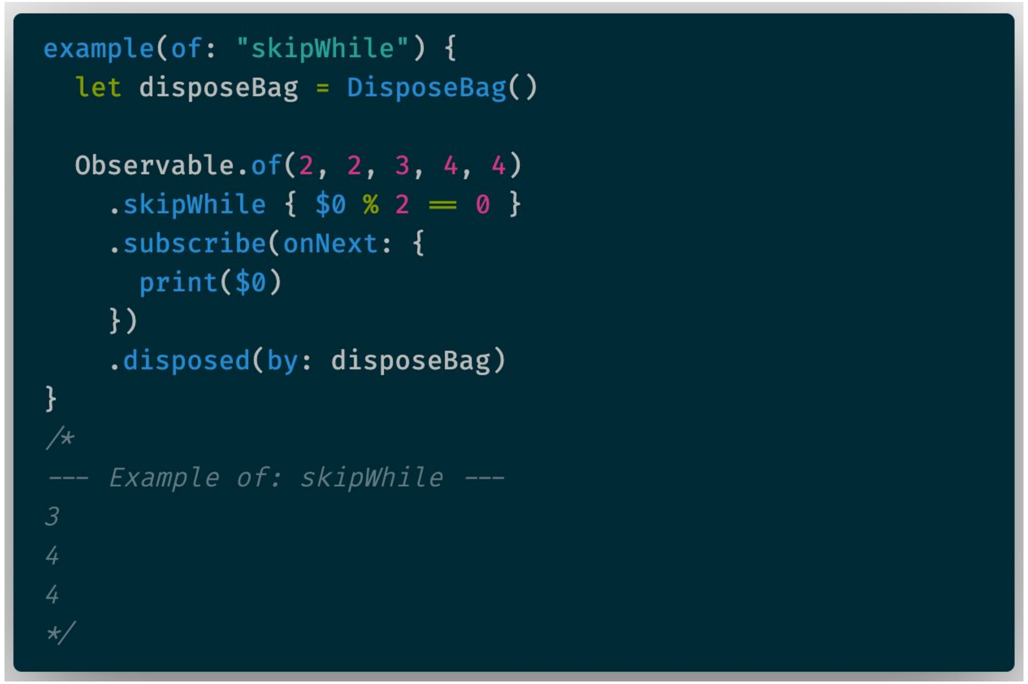
3.2. skipWhile()

❖ And with skipWhile(_:), returning true will cause the element to be skipped, and returning false will let it through. It's the opposite of filter.



3.2. skipWhile()

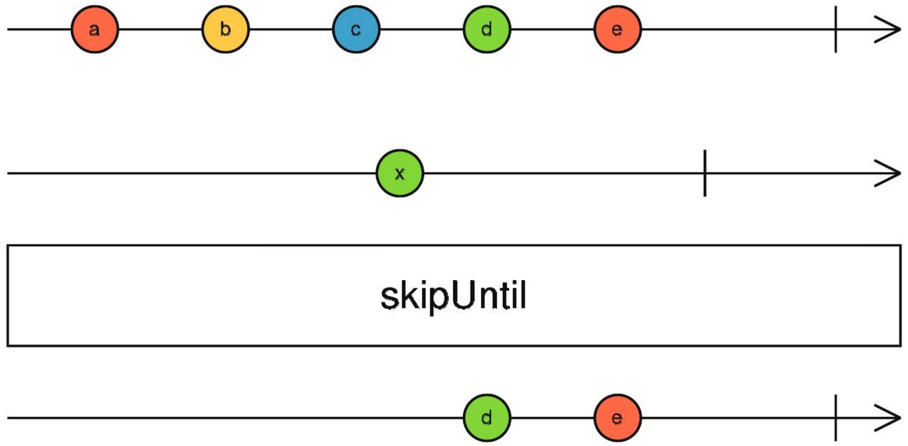
❖ Example:



```
example(of: "skipWhile") {
  let disposeBag = DisposeBag()

  Observable.of(2, 2, 3, 4, 4)
    .skipWhile { $0 % 2 == 0 }
    .subscribe(onNext: {
      print($0)
    })
    .disposed(by: disposeBag)
}
/*
--- Example of: skipWhile ---
3
4
4
*/
```

3.3. skipUntil()

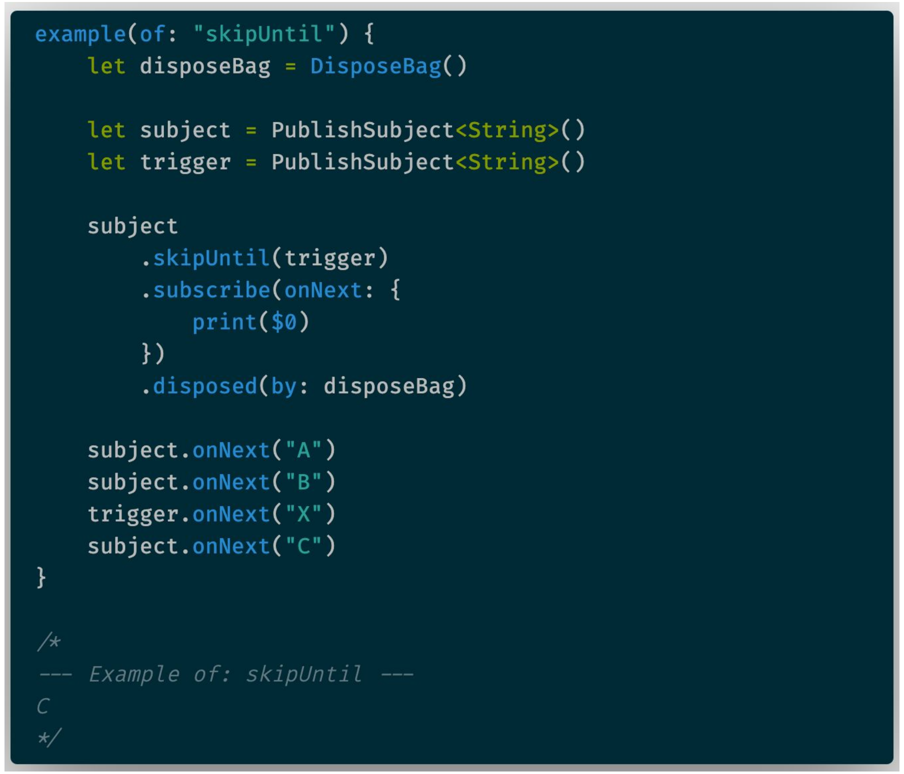
❖ This operator will keep skipping elements from the source observable (the one you are subscribing to) until some other trigger observable emits.



Sun*

3.3. skipUntil()

❖ Example:



```
example(of: "skipUntil") {
    let disposeBag = DisposeBag()

    let subject = PublishSubject<String>()
    let trigger = PublishSubject<String>()

    subject
        .skipUntil(trigger)
        .subscribe(onNext: {
            print($0)
        })
        .disposed(by: disposeBag)

    subject.onNext("A")
    subject.onNext("B")
    trigger.onNext("X")
    subject.onNext("C")
}

/*
--- Example of: skipUntil ---
C
*/
```

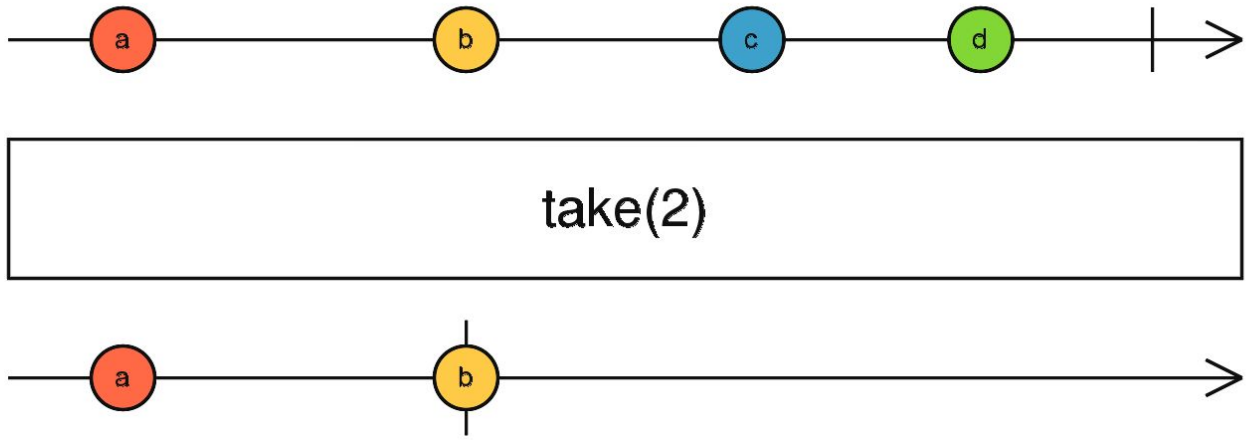
4. Taking operators

1. take()
2. takeWhile()
3. takeUntil()

Sun*

4.1. take()

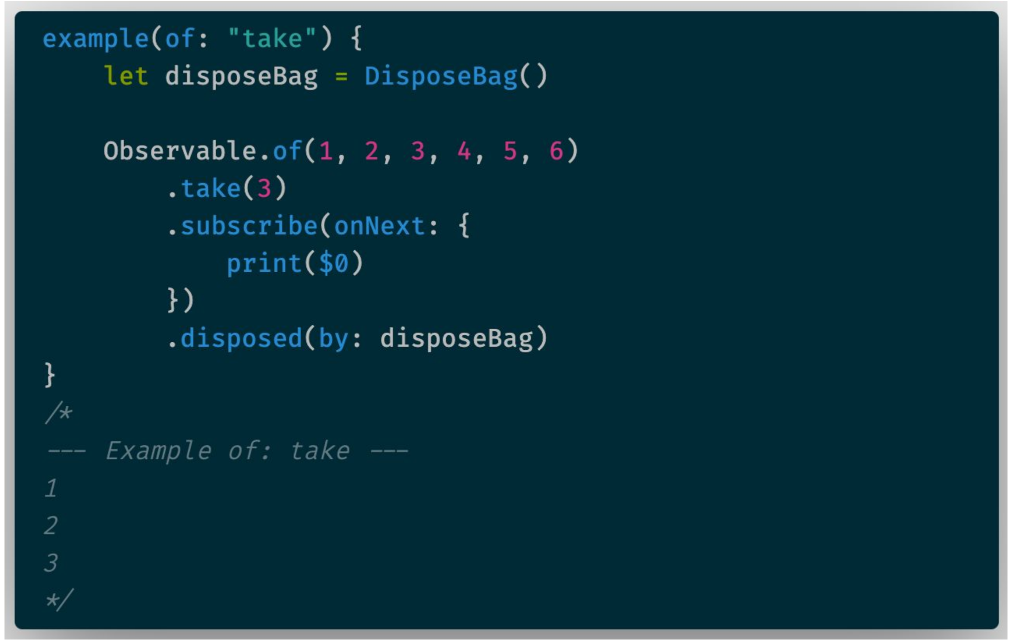
❖ This operator emits only the first of the number of elements you specified emitted by the source Observable.



Sun*

4.1. take()

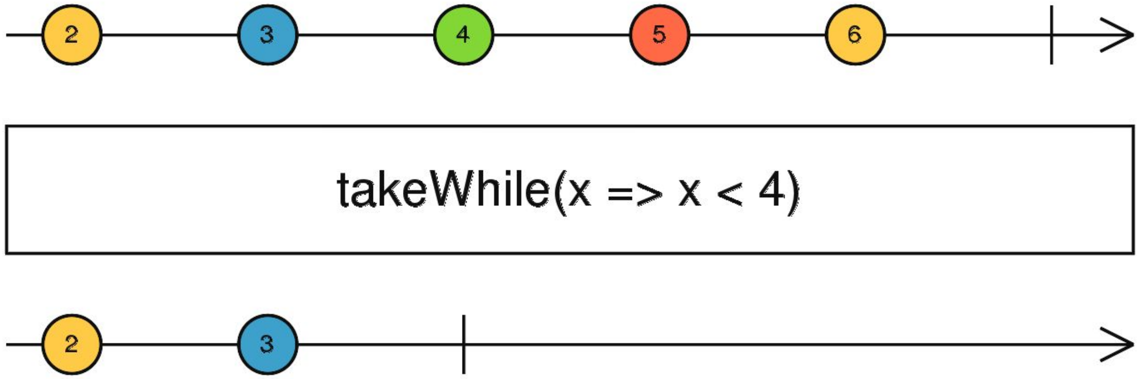
- Example:



```
example(of: "take") {
    let disposeBag = DisposeBag()

    Observable.of(1, 2, 3, 4, 5, 6)
        .take(3)
        .subscribe(onNext: {
            print($0)
        })
        .disposed(by: disposeBag)
}
/*
--- Example of: take ---
1
2
3
*/
```

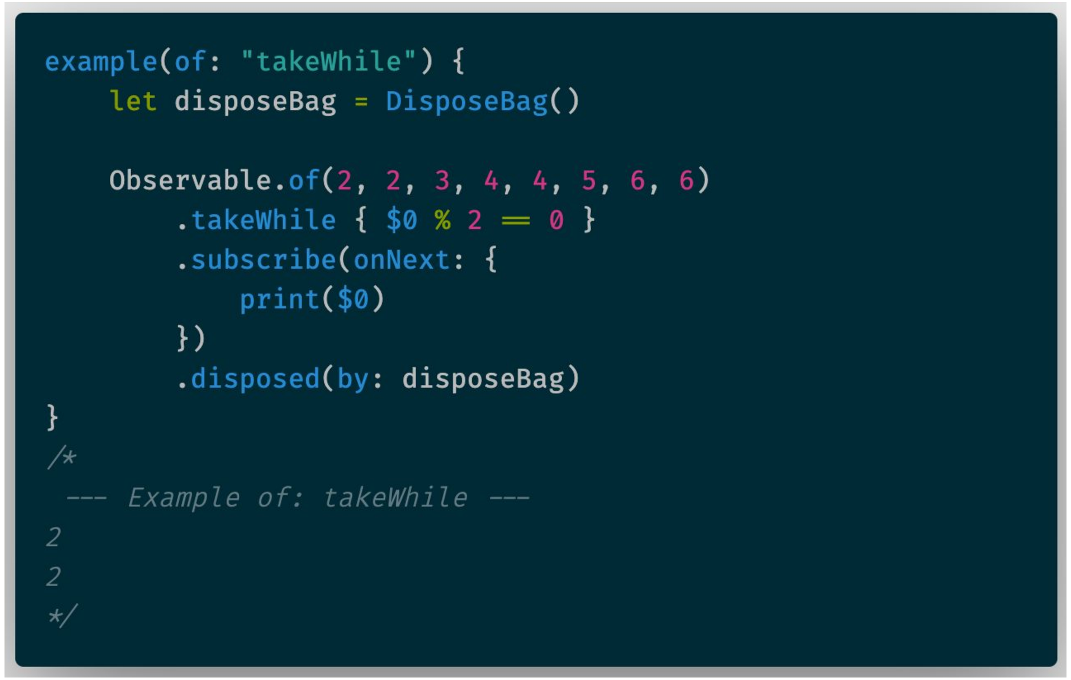
4.2. takeWhile()

❖ takeWhile works similarly to skipWhile, except you are taking instead of skipping.



4.2. takeWhile()

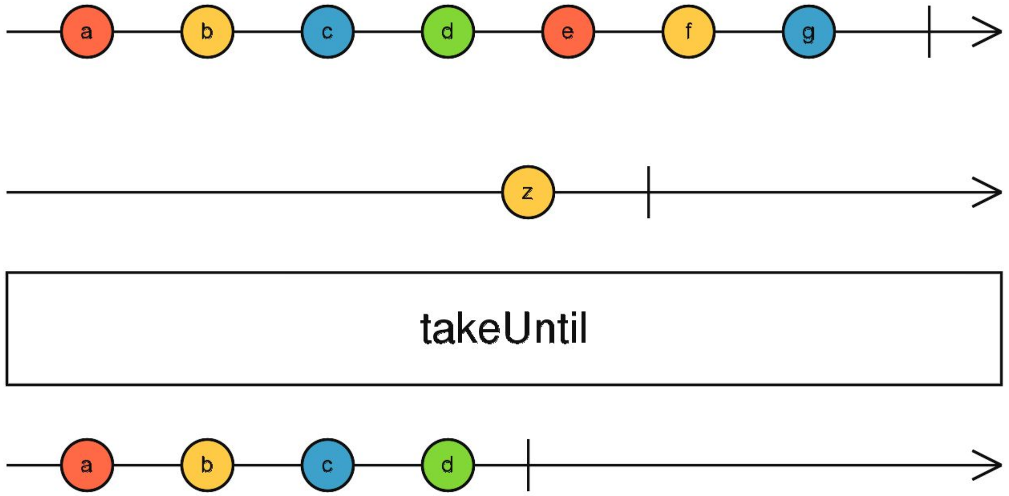
❖ Example:



```
example(of: "takeWhile") {
    let disposeBag = DisposeBag()

    Observable.of(2, 2, 3, 4, 4, 5, 6, 6)
        .takeWhile { $0 % 2 == 0 }
        .subscribe(onNext: {
            print($0)
        })
        .disposed(by: disposeBag)
}
/*
 --- Example of: takeWhile ---
2
2
*/
```

4.3. takeUntil()

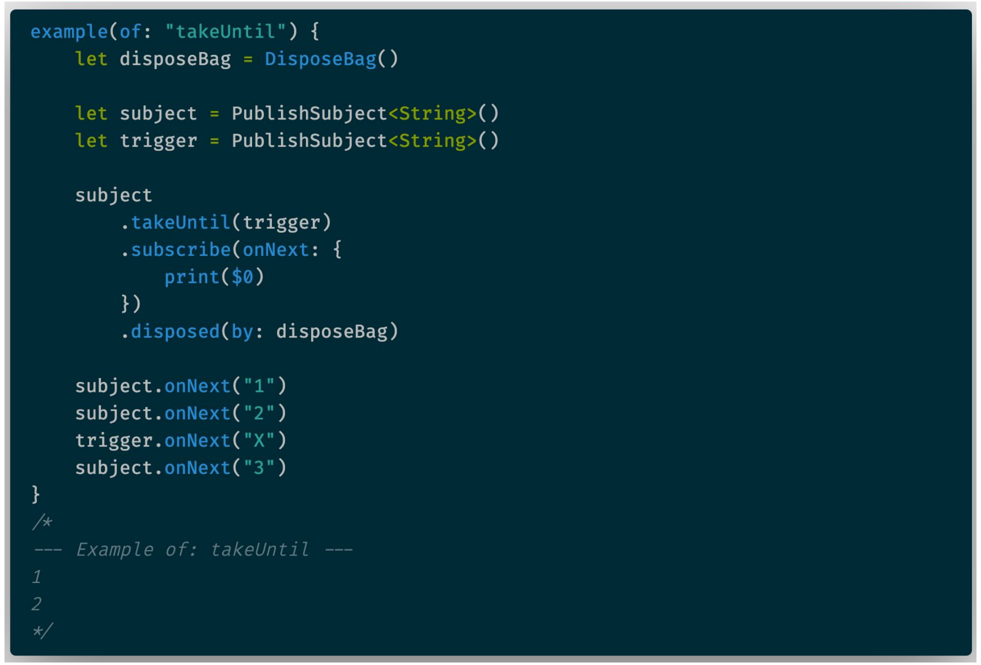
❖ Emits the values emitted by the source Observable until another Observable emits a value.



Sun*

4.3. takeUntil()

❖ Example:



```
example(of: "takeUntil") {
    let disposeBag = DisposeBag()

    let subject = PublishSubject<String>()
    let trigger = PublishSubject<String>()

    subject
        .takeUntil(trigger)
        .subscribe(onNext: {
            print($0)
        })
        .disposed(by: disposeBag)

    subject.onNext("1")
    subject.onNext("2")
    trigger.onNext("X")
    subject.onNext("3")
}
/*
--- Example of: takeUntil ---
1
2
*/
```

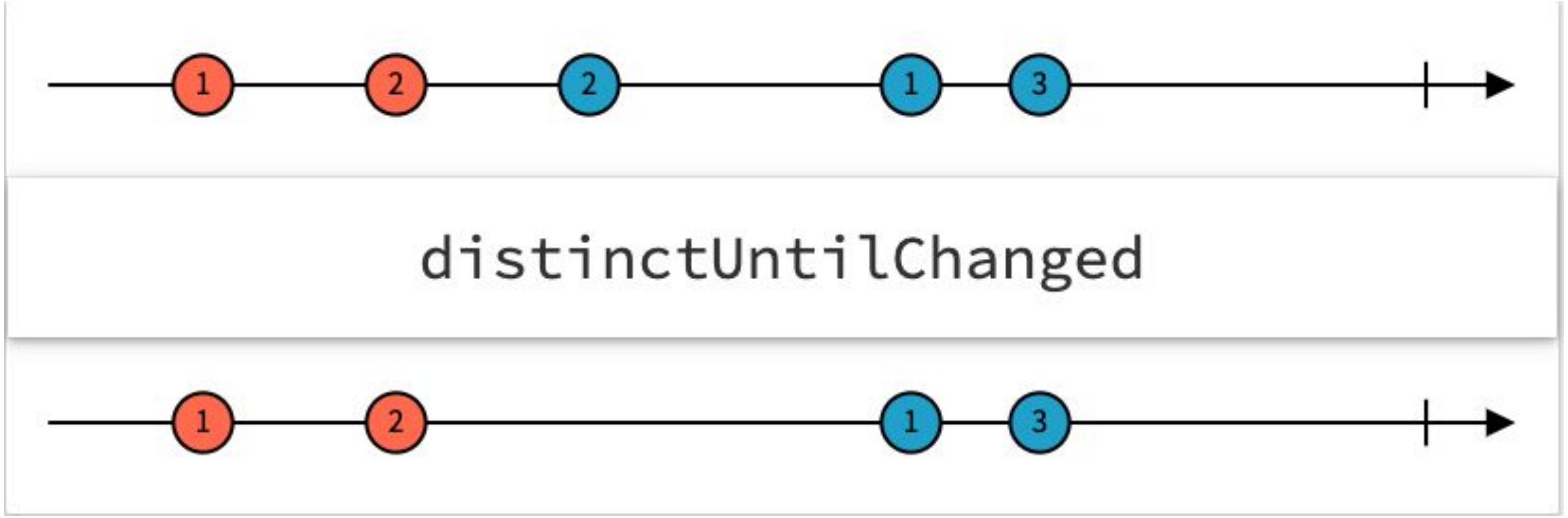
5. Distinct operators

1. distinctUntilChanged()

Sun*

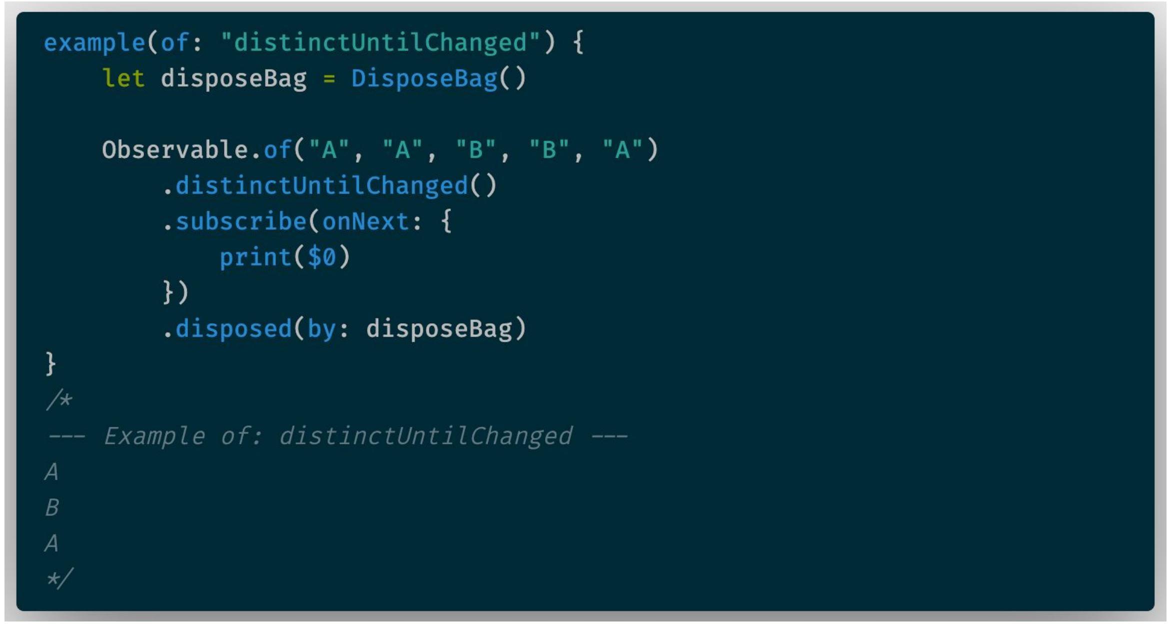
5.1. distinctUntilChanged()

- distinctUntilChanged() prevents duplicate contiguous items from getting through.



5.1. distinctUntilChanged()

- Example:



```
example(of: "distinctUntilChanged") {
    let disposeBag = DisposeBag()

    Observable.of("A", "A", "B", "B", "A")
        .distinctUntilChanged()
        .subscribe(onNext: {
            print($0)
        })
        .disposed(by: disposeBag)
}
/*
--- Example of: distinctUntilChanged ---
A
B
A
*/
```

Sun*

Question & Answer?

Sun*