Chapter 10: Sets

Section 10.1: Declaring Sets

Sets are unordered collections of unique values. Unique values must be of the same type.

```
var colors = Set<String>()
```

You can declare a set with values by using the array literal syntax.

```
var favoriteColors: Set<String> = ["Red", "Blue", "Green", "Blue"]
// {"Blue", "Green", "Red"}
```

Section 10.2: Performing operations on sets

Common values from both sets:

You can use the `intersect(_:)` method to create a new set containing all the values common to both sets.

```
let favoriteColors: Set = ["Red", "Blue", "Green"]
let newColors: Set = ["Purple", "Orange", "Green"]

let intersect = favoriteColors.intersect(newColors) // a AND b
// intersect = {"Green"}
```

All values from each set:

You can use the `union(_:)` method to create a new set containing all the unique values from each set.

```
let union = favoriteColors.union(newColors) // a OR b
// union = {"Red", "Purple", "Green", "Orange", "Blue"}
```

Notice how the value "Green" only appears once in the new set.

Values that don't exist in both sets:

You can use the `exclusiveOr(_:)` method to create a new set containing the unique values from either but not both sets.

```
let exclusiveOr = favoriteColors.exclusiveOr(newColors) // a XOR b
// exclusiveOr = {"Red", "Purple", "Orange", "Blue"}
```

Notice how the value "Green" doesn't appear in the new set, since it was in both sets.

Values that are not in a set:

You can use the `subtract(_:)` method to create a new set containing values that aren't in a specific set.

```
let subtract = favoriteColors.subtract(newColors) // a - (a AND b)
// subtract = {"Blue", "Red"}
```

Notice how the value "Green" doesn't appear in the new set, since it was also in the second set.

Section 10.3: CountedSet

Version = 3.0

Swift 3 introduces the `CountedSet` class (it's the Swift version of the `NSCountedSet` Objective-C class).

CountedSet, as suggested by the name, keeps track of how many times a value is present.

```
let countedSet = CountedSet()
countedSet.add(1)
countedSet.add(1)
countedSet.add(1)
countedSet.add(2)

countedSet.count(for: 1) // 3
countedSet.count(for: 2) // 1
```

Section 10.4: Modifying values in a set

```
var favoriteColors: Set = ["Red", "Blue", "Green"]
//favoriteColors = {"Blue", "Green", "Red"}
```

You can use the `insert(_:)` method to add a new item into a set.

```
favoriteColors.insert("Orange")
//favoriteColors = {"Red", "Green", "Orange", "Blue"}
```

You can use the `remove(_:)` method to remove an item from a set. It returns optional containing value that was removed or nil if value was not in the set.

```
let removedColor = favoriteColors.remove("Red")
//favoriteColors = {"Green", "Orange", "Blue"}
// removedColor = Optional("Red")

let anotherRemovedColor = favoriteColors.remove("Black")
// anotherRemovedColor = nil
```

Section 10.5: Checking whether a set contains a value

```
var favoriteColors: Set = ["Red", "Blue", "Green"]
//favoriteColors = {"Blue", "Green", "Red"}
```

You can use the `contains(_:)` method to check whether a set contains a value. It will return true if the set contains that value.

```
if favoriteColors.contains("Blue") {
    print("Who doesn't like blue!")
}
// Prints "Who doesn't like blue!"
```

Section 10.6: Adding values of my own type to a Set

In order to define a `Set` of your own type you need to conform your type to `Hashable`

```
struct Starship: Hashable {
```

```
    let name: String
    var hashValue: Int { return name.hashValue }
}

func ==(left:Starship, right: Starship) -> Bool {
    return left.name == right.name
}
```

Now you can create a `Set` of `Starship(s)`

```
let ships : Set<Starship> = [Starship(name:"Enterprise D"), Starship(name:"Voyager"),
Starship(name:"Defiant") ]
```