Урок 4

Типы коллекций

Коллекции используются для хранения наборов значений. В Swift используется три типа коллекций - это Массивы, Словари и Множества. Массивы могут хранить в себе упорядоченные, не уникальные значения одного типа. Множества хранят в себе неупорядоченные, уникальные значения одного типа. Словари - это неупорядоченные коллекции, которые хранят в себе пары «ключ-значение». Тип ключа словаря может отличаться от типа значения. При этом сами ключи и значения должны быть одного типа.

Когда вы создаете какую либо коллекцию, то вы присваиваете её какой-то константе или переменной. В зависимости от этого коллекции могут быть изменяемыми или неизменными.

Массивы

Массивы – это коллекции, в которых могут храниться значения одного типа. Эти значения могут быть не уникальными, но всегда упорядоченными. Все значения в массиве хранятся по индексу в определенном порядке. Нумерация индексов начинается с нуля.

Значения внутри массива заключаются в квадратные скобки и отделены между собой запятыми.

Обратившись к массиву, можно вызвать логическое свойство **isEmpty**, которое позволяет узнать, является ли данный массив пустым или нет.

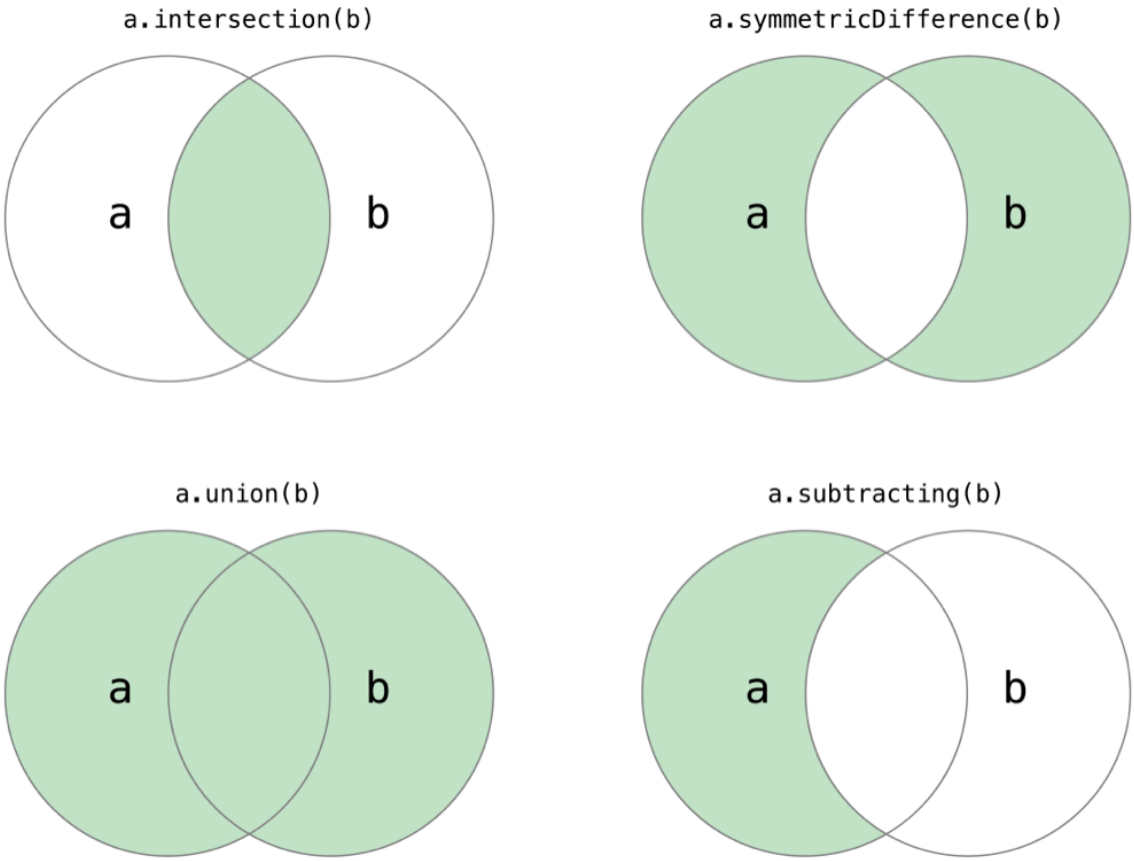
Множества

Отличием множества от массива является то, что множества хранят в себе уникальные значения одного типа в неупорядоченном виде. Вы можете использовать множества в том случае, если хотите быть уверены в том, что значения элементов внутри множества не повторяются. В отличии от массивов множества не имеют сокращенной формы записи, поэтому для того, что бы создать множество, а не массив, необходимо указать тип коллекции **Set** при создании коллекции.

Можно проверить наличие определенного элемента во множестве, используя метод `contains()`

Так как элементы внутри множества хранятся в неупорядоченном виде, то для того, что бы у вас была возможность делать итерации при переборе элементов внутри коллекции, необходимо сортировать данные внутри множества. Для этого нужно использовать метод `sorted()`, который возвращает вам элементы коллекции в виде отсортированного массива, используя оператор <. Т.е. это сортировка происходит от меньшего к большему

Кроме того, что элементы множества хранятся в неотсортированном виде, еще одной особенностью сетов, является уникальность всех значений. Эта особенность даёт нам дополнительные возможности для сравнения двух сетов между собой. Вы можете очень эффективно использовать базовые операции множеств, например, комбинирование двух множеств, определение общих значений двух множеств, определять содержат ли множества несколько, все или ни одного одинаковых значения.



Данная иллюстрация изображает два множества **a** и **b** в результате применения различных методов.

- Метод `intersection()` позволяет создать новое множество из общих значений двух входных множеств.
- Метод `symmetricDifference()` используется для создания нового множества из значений, которые не повторяются в двух входных множествах.
- Используйте метод `union()` для создания нового множества состоящего из всех значений обоих множеств.
- И метод `subtracting()` для создания множества со значениями не принадлежащих второму множеству из двух сравниваемых.

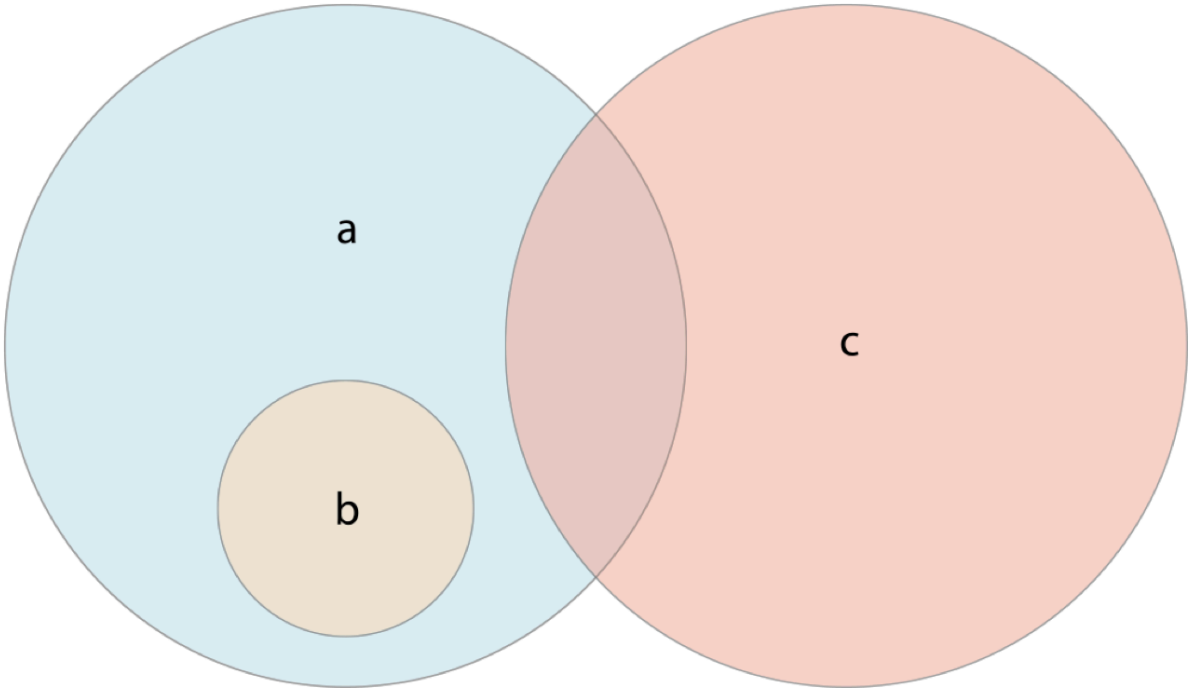


Иллюстрация выше отображает три множества **a**, **b** и **c**. Множество **a** является надмножеством множества **b**, так как содержит все его элементы, соответственно множество **b** является подмножеством множества **a**, опять таки потому, что все его элементы находятся в **a**. Множества **b** и **c** называются разделенными, так как у них нет общих элементов.

- Можно использовать оператор равенства (==) что бы определить все ли значения двух множеств одинаковы.
- Метод `isSubset(of:)` используется для определения все ли значения множества содержаться в указанном множестве.
- Метод `isSuperset(of:)`, используется чтобы определить содержит ли множество все значения указанного множества.
- Такие методы, как `isStrictSubset(of:)` или `isStrictSuperset(of:)` используется для определения является ли множество подмножеством или надмножеством, но не равным указанному сету.

- Для того, что бы определить есть ли у двух множеств общие значения используется метод `isDisjoint(with:)`

Словари

Словари - это такие коллекции, которые хранят в себе однотипные значения. Каждое значение связано с уникальным ключом, который выступает в качестве идентификатора этого значения внутри словаря. Ключи так же, как и значения должны быть одного типа, но при этом их тип может отличаться от типа значений. Зная ключ, можно извлечь значение из словаря. Так же как и во множествах, элементы словаря не имеют определенного порядка.

Поскольку есть вероятность запросить ключ для несуществующего значения, индекс словаря возвращает опциональное значение соответствующее типу значений словаря. Если словарь содержит значение для запрошенного ключа, индекс возвращает опциональное значение, содержащее существующее значение для этого ключа. В противном случае индекс возвращает nil:

В остальном работа со словарями такая же, как и с массивами.

Циклы

for-in циклы

Циклы относятся к операторам управления потоком. Они используются для многократного выполнения задач и делятся на **for in** циклы и **while** циклы

Цикл `for-in` используется для итерации по коллекциям элементов, таких как диапазоны чисел, элементы массива, элементы словаря и множеств, символов в строке и других последовательностей.

Циклы используются тогда, когда возникает необходимость повторить какой то кусок кода несколько раз. Каждый такой проход и называется итерацией. Количество итераций может зависеть от заданного вами диапазона, либо же пока не удовлетворится условие, например пока не найдется определенный элемент в массиве.

Синтаксис цикла `for in`:

```
for counter in lower…upper {
   some code
}
```

Цикл начинается с ключевого слова **for**, далее идет переменная **counter**, которая принимает значение из диапазона при каждой итерации. Количество повторов зависит от диапазона от **lower** до **upper**. К примеру если диапазон составляв от 1 до 5, то код заключенный между фигурными скобками повторится 5 раз. Так в нашем примере **counter** при каждой итерации будет меняться с **lower** до **upper**

While

Цикл **while** выполняет набор инструкций до тех пор, пока его условие не станет **false**. Этот вид циклов лучше всего использовать в тех случаях, когда количество итераций неизвестно. Swift предлагает два вида циклов **while**:

- while - вычисляет условие выполнения в начале каждой итерации цикла.
- repeat-while - вычисляет условие выполнения в конце каждой итерации цикла.

Так выглядит синтаксис **while** цикла:

```
while condition {
```

```
    some code
}
```

Расшифровать его можно так: Если наше условие истинно, выполняем инструкцию, находящуюся между двумя фигурными скобками. После выполнения инструкции снова проверяем условие. Если оно истинно, то еще раз выполняем код, внутри цикла и так до тех пора, пока наше условие не станет **false**.

Repeat-While

```
repeat {
    some code
} while condition
```

В отличии от предыдущего цикла этот цикл всегда будет выполнен, как минимум один раз, в не зависимости от того, истинно заданное условие или нет. Как видите в этом цикле первым идет ключевое слово **repeat**. В данном контексте его можно интерпретировать, как **выполнить**. Далее между фигурными скобками помещается код, который необходимо выполнить. После того, как инструкция выполнена, идет проверка на истинность условия. Если оно истинно, то цикл запускается снова и так до тех пор, пока условие не станет **false**. Из за того, что проверка условия происходит в конце, код будет выполнен, как минимум один раз, при первом запуске цикла.