



Спринт 1, тема «Коллекции и циклы»

Яндекс Практикум

Мы сделали для вас шпаргалку по теме «Коллекции и циклы». Здесь вы найдёте короткое изложение пройденного в уроке материала и ключевые фрагменты кода. Используйте шпаргалку, чтобы быстро восстановить в памяти пройденный материал.

 Скачайте документ, чтобы обращаться к нему при необходимости, пока не доведёте навыки до автоматизма.

Коллекции

 **Коллекции** в Swift — это структуры данных, которые хранят набор элементов. Коллекции используют, чтобы управлять группами данных и обрабатывать эти группы.

В языке Swift есть три основных типа коллекций: массивы, словари и множества.

- **Массивы** — это упорядоченные коллекции значений. Они хранят несколько значений одного типа в одной коллекции. Например:

```
let array = [1, 2, 3, 4, 5]
```

- **Словари** — это коллекции пар «ключ-значение». Они хранят значения, которые связываются с ключами. Например:


```
let dictionary = ["key1": 1, "key2": 2, "key3": 3]
print(dictionary["key1"])
```

В коллекциях «ключ-значение» каждому уникальному ключу соответствует одно или несколько значений. Это позволяет искать значения по ключам, что быстрее и эффективнее, чем перебирать все элементы коллекции.

- **Множества** — это коллекции уникальных значений. Они хранят набор значений, в котором каждое значение уникально. Например:

```
let set = Set([1, 2, 3, 4, 5])
```

Массивы

 **Массив** — коллекция, которая упорядоченно хранит данные одного типа. Например, массив может содержать несколько элементов типа `Int`. Но в массиве нельзя хранить вперемешку элементы типа `Int` и `String`.

Чтобы объявить массив строк, используют конструкцию:

```
var arrayOfStrings: [String] = [ "Ivan", "Andrey", "Alex" ]
```

Есть и другой способ создать массив:

```
var arrayOfStrings = [String]()
arrayOfStrings.append("Ivan")
arrayOfStrings.append("Andrey")
arrayOfStrings.append("Alex")
```

Первый вариант предпочтительнее, потому что короче.

Если массив не пуст, то минимальный индекс равен 0, а максимальный — на 1 меньше количества элементов. Например, в массиве `arrayOfStrings` количество элементов равно 3, а индексы элементов — 0, 1, 2.

Чтобы получить значение по индексу 0 из массива `arrayOfStrings`, используют конструкцию: `arrayOfStrings[0]`.

Чтобы узнать количество элементов в массиве, используйте `count`:

```
arrayOfStrings.count // Распечатается число 3
```

Чтобы узнать, пустой ли массив, используют `isEmpty`:

```
arrayOfStrings.isEmpty // Распечатается false, если массив не пустой, и true – в проти-
воположном случае
```

Чтобы удалить из массива элемент по индексу, используйте синтаксис:

```
let removedElement = arrayOfStrings.remove(at: 1)
```

Словари

Для хранения пар ключ-значение используют коллекцию «словарь» (англ. "Dictionary" — словарь).

Для объявления словаря используют синтаксис:

```
var fruitsCount = [
    "банан": 5,
    "апельсин": 3,
    "манго": 10
]
```

Чтобы получить значения по заданному ключу, используют конструкцию:

```
fruitsCount["банан"] // Равно 5
```

Чтобы изменить значения по заданному ключу, используют конструкцию:

```
fruitsCount["банан"] = 4
```

Чтобы удалить значение для заданного ключа, используют:

```
fruitsCount.removeValue(forKey: "банан")
```

Множества

Для хранения множества уникальных данных используют тип данных «множество»:

```
var ourSet: Set<Int> = []
```

Множества, как массивы и словари, можно создавать заданным набором исходных значений:

```
var ourSet: Set = [1, 2, 3, 4, 5]
```

Чтобы добавить элемент в множество, используют `insert`:

```
ourSet.insert(1)
```

В множестве хранятся только уникальные элементы. Не получится добавить в множество элемент, который в нём уже есть.

Узнать количество элементов в множестве помогает `count`:

```
ourSet.count
```

Чтобы узнать есть ли заданный элемент в множестве, используют синтаксис:

```
if ourSet.contains(1) {  
    print("ourSet contains 1")  
} else {  
    print("ourSet does not contain 1s")  
}
```

Циклы

Цикл repeat-while

Чтобы выполнять действие, пока не наступит заданное условие, используют цикл repeat-while:

```
repeat {  
    <действие>  
} while (<условие>)
```

Код внутри фигурных скобок называют **телом цикла**. А условие после `while` — **условием цикла**.

Например:

```
var productTemperature = 20
repeat {
    productTemperature += 20
    print(productTemperature)
} while productTemperature < 60
```

Обратите внимание: оператор `repeat-while` сначала выполняет действие и только потом проверяет условие. Поэтому тело цикла `repeat-while` выполнится минимум один раз.

Цикл `while`

Если условие нужно проверять до выполнения тела цикла, используют оператор `while`:

```
while productTemperature < 80 {
    productTemperature += 20
    print(productTemperature)
}
```

Обратите внимание: оператор `while` сначала проверяет условие и, только если оно верно (результат его вычисления равен `true`), выполняет тело цикла. Поэтому если в начальный момент условие будет ложным (результат его вычисления равен `false`), то тело цикла не выполнится ни разу.

Цикл `for`

Если нужно выполнить действие заданное количество раз, используют цикл `for`:

```
for i in 1..10 {  
    print(i)  
}
```

Цикл из примера не зависит ни от каких условий и в любом случае выполнится ровно 10 раз. Он эквивалентен такому циклу `while`:

```
var i = 1  
while i <= 10 {  
    print(i)  
    i += 1  
}
```

Также цикл `for` используют для «обхода» коллекций, например, для массива:

```
let ourDigits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
  
for digit in ourDigits {  
    print(digit)  
}
```

Код из примера эквивалентен такому:

```
let ourDigits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
  
for i in 0..<ourDigits.count {  
    let digit = ourDigits[i]  
    print(digit)  
}
```

Пример обхода словаря:

```
var astronauts: [String: Int] = [  
    "Маша": 1,  
    "Игорь": 2,  
    "Антон": 1,  
    "Таня": 3  
]  
  
for (astronaut, mealCount) in astronauts {  
    print(astronaut, mealCount)  
}
```

Оператор `forEach`

Иногда для обхода коллекций вместо `for` используют оператор `forEach`:

```
ourDigits.forEach { digit in
    print(digit)
}
```

Оператор `forEach` менее предпочтителен, так как в нём не работают `break` и `continue` (см. ниже).

Ключевое слово `break`

Чтобы досрочно выйти из цикла, используют ключевое слово `break` (англ. "to break" — тормозить, остановить).

Обычно перед обращением к `break` проверяется заданное условие, которое называют **условием (досрочного) выхода**.

Например, фрагмент кода ниже добавляет в массив `alienDigits` значения из массива `ourDigits` до тех пор, пока очередное число меньше либо равно 2. Как только попадётся большее число, цикл прерывается:

```
let ourDigits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

var alienDigits = [Int]()
for digit in ourDigits {
    if digit > 2 {
        break
    }
    alienDigits.append(digit)
}
print(alienDigits)
```

Обратите внимание: `break` допустимо использовать только внутри тела цикла.

Ключевое слово `continue`

Иногда в теле цикла есть условие, от которого зависит, выполнять ли код дальше или пропустить всё, что ниже, и сразу перейти на следующий шаг

цикла. В таких случаях помогает ключевое слово `continue`.

Например фрагмент кода ниже копирует из строки `greetings` в строку `alienGreetings` все символы кроме пробела:

```
let greetings = "Привет! Мы пришли к вам с миром!"
var alienGreetings = ""
for character in greetings {
    if character == " " {
        continue
    }
    alienGreetings.append(character)
}

print(alienGreetings)
```

Яндекс Практикум