


Спринт 1, тема «Функции»

Яндекс Практикум

Мы сделали для вас шпаргалку по теме «Функции». 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.

Функции

Функция — это *блок кода*, который выполняет конкретную задачу и может быть многократно вызван в рамках одной программы. Функции разбивают код на мелкие повторяемые части. Это упрощает его и оптимизирует.

Блок кода — это *набор инструкций, или команд*, которые выполняются вместе как единый элемент. Чтобы улучшить читаемость кода и использовать его повторно, команды вкладывают друг в друга или выделяют в отдельные функции.

Набор инструкций, или команды, — это конкретные действия, заданные в языке программирования, которые компьютер должен выполнить. Примеры команд: присвоить переменной значение, вычислить выражение, вызвать функцию. Когда запускают программу, компьютер последовательно выполняет команды.

Синтаксис объявления функции

В общем виде объявление функции в Swift выглядит так:

```
func functionName(parameters) -> Int {  
    // function body  
    return result  
}
```

Объявление функции содержит:

1. Ключевое слово `func` — обозначает начало объявления функции.
2. Название функции `functionName`, по которому мы сможем вызвать её в остальной программе.

Обратите внимание: название не должно начинаться с цифр или специальных символов, которые используют в синтаксисе Swift (например, название не должно начинаться с `.`, `,`, `/`, `(`, `)`, `[`, `]`, `{`, `}`, `$`).

3. Список параметров в круглых скобках `(parameters)` — это входные данные, которые будут доступны в теле функции.
4. Тип возвращаемого значения. В примере — `Int`.
5. Тело функции в фигурных скобках.
6. Возвращаемое значение `return result`. Ключевое слово `return` указывает на завершение работы функции и возвращает переменную `result` как результат. Тип этой переменной указан в п.4.

Синтаксис вызова функции

В общем виде вызов функции выглядит так:

```
let result = functionName(parameters)
```

В этом коде:

1. Объявляем константу, в которую сохранится возвращаемое значение функции: `let result`.
2. Присваиваем этой константе результат вызова функции с помощью оператора `=`.
3. Вызываем функцию с помощью её названия `functionName`.
4. За названием функции в круглых скобках указываем параметры для вызова `(parameters)`.

В результате мы вызовем функцию, и результат её выполнения сохранится в константу `result`. Swift будто заменит вызов `functionName(parameters)` на результат этой функции.

Пример

Пример функции:

```
import Foundation

// Объявление функции researchPlanet с параметрами
// - shipName типа String - название корабля;
// - planetName типа String - название планеты;
// Возвращающую
// - массив строк - [String] - список найденных видов жизни.
func researchPlanet(shipName: String, planetName: String) -> [String] {
    // тело функции
    return ... // возвращаемое значение
}

// Вызов функции с параметрами
// - shipName: «Тысячелетний сокол»;
// - planetName: «Ка-Пэкс».
let foundSpeciesResult = researchPlanet(
    shipName: «Тысячелетний сокол»,
    planetName: «Ка-Пэкс»
)

// В `foundSpeciesResult` будет находиться результат вызова функции.
print("Найдено \\(foundSpeciesResult.count) форм жизни")
```

Тип `Void`

Есть функции, которые ничего не возвращают как результат. Например, функция `print` выводит переданные значения на экран и завершается. В таком случае вместо типа возвращаемого значения функции используют ключевое слово `Void`:

```
func doSomething(...) -> Void {
    print("Hello world!")
    // ...
}

doSomething()
```

Если не указан тип значения, которое возвращает функция, то по умолчанию программа считает, что функция не возвращает ничего. Поэтому ключевое слово `Void` можно опускать.

```
func doSomething(...) {  
    print("Hello World!")  
    // ...  
}  
  
doSomething()
```

Внешнее и внутреннее название параметра

Параметры функции имеют внешнее и внутреннее название.

Внешним названием параметра называется название параметра, которое используется извне кода функции. В момент вызова функции используют именно внешнее название параметра.

Внутреннее название — это имя параметра внутри тела функции. Когда к параметру обращаются из тела функции, то используют именно его внутреннее название.

Внешнее и внутреннее название параметра могут совпадать или быть разными.

Если эти названия совпадают, то используют синтаксис:

```
func greetPerson(personName: String) {  
    print("Hello, \\\(personName)!")  
}  
  
greetPerson(personName: "Timofei")
```

Чтобы улучшить читаемость кода, иногда используют разные внешние и внутренние названия параметра. В этом случае синтаксис такой:

```
func greetPerson(withName personName: String) {  
    print("Hello, \\\(personName)!")  
}  
  
greetPerson(withName: "Timofei")
```

Параметр без внешнего названия

Иногда желательно вовсе скрыть внешнее название параметра, например, если оно избыточно по контексту. Тогда в объявлении функции в качестве внешнего названия параметра используют знак нижнего подчёркивания `_`:

```
func greetPerson(_ personName: String) {  
    print("Hello, \"\"(personName)!\")  
}  
  
greetPerson("Timofei")
```

Модификатор параметров `inout`

Обычно, все изменения, что происходят с значением параметра, переданным в функцию, остаётся в функции.

Если нужно изменить значение переданного параметра для всей программы, то в Swift используют ключевое слово `inout`:

```
func incrementNumber(_ number: inout Int) {  
    number += 1  
}  
  
var number: Int = 0  
  
print(number)  
incrementNumber(&number)  
print(number)
```

Уровни доступа функций

В Swift есть разные уровни доступа к функциям внутри и вовне модуля. Чтобы управлять уровнями доступа, используют специальные ключевые слова.

Пойдём от самого закрытого уровня доступа к самому открытому:

1. `private` — приватный уровень доступа. Функция доступна только в той сущности и в том файле, где она объявлена.

```
private func doSomething(...) { ... }
```

2. `fileprivate` — приватный уровень доступа. Функция доступна только в коде файла, где она объявлена.

```
fileprivate func doSomething(...) { ... }
```

3. **internal** — внутренний уровень доступа. Функция доступна во всём модуле проекта, где она объявлена. За пределами компиляционного модуля она уже недоступна.

```
internal func doSomething(...) { ... }
```

Обратите внимание: этот уровень доступа используется по умолчанию, то есть его можно не писать:

```
func doSomething(...) { ... }
```

4. **public** — публичный уровень доступа. Функция доступна во всём модуле, где она объявлена, а также в модулях, использующих этот модуль.

```
public func doSomething(...) { ... }
```

5. **open** — публичный уровень доступа. Функция доступна во всём модуле, где она объявлена, и в модулях, использующих этот модуль. Также эту функцию можно **переопределять** в модулях снаружи (подробнее о переопределении функций — в уроках и шпаргалке про классы). Это самый открытый уровень доступа.

```
open func doSomething(...) { ... }
```

Яндекс Практикум