



# Спринт 2, тема «Структуры»

## Яндекс Практикум

Мы сделали для вас памятку по теме «Структуры». 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.

## Структуры

 **Структура** — элемент, с помощью которого в Swift объединяют данные. Например, создадим структуру «Пользователь приложения»:

```
struct User {  
    let id: Int  
    let login: String  
    let name: String  
}
```

Фрагмент кода между фигурными скобками после названия структуры `struct User {` `<тело структуры> }` называют **телом структуры**.

Записи вида `let <имя_свойства>: <тип_свойства>` называют свойствами (от англ. "properties", читается «проперти» — свойства). Структура в коде выше содержит свойства: id, логин, имя. Свойство id имеет тип `Int`, а логин и имя — `String`.

Структура «Пользователь приложения» — шаблон для всех пользователей приложения. Создадим переменную типа `User` и наполним её данными:

```
let adminUser = User(  
    id: 1,  
    login: "admin@service.com",  
    name: "Admin"  
)
```

Свойства структуры могут быть константами или переменными:

- свойства-константы объявляют с помощью `let`, их значение нельзя изменить;
- свойства-переменные объявляют с помощью `var`, их менять можно.

Все свойства структуры `User` — константы, значит, после создания объекта их значения изменить нельзя.

Представим, что программа должна позволять пользователям изменять имя `name` после создания структуры. Попробуем изменить `name` для `adminUser` в примере выше:

```
var adminUser = User(  
    id: 1,  
    login: "admin@service.com",  
    name: "Admin"  
)  
  
adminUser.name = "Admin User"
```

Увы, получим ошибку!

Обратите внимание: `userAdmin` — это переменная, но её свойство `name` — константа, поэтому его менять нельзя.

Чтобы разрешить пользователям менять имя, нужно сделать свойство `name` переменной:

```
struct User {  
    let id: Int  
    let login: String  
    var name: String  
}
```

Теперь значение свойства `name` можно изменять:

```
adminUser.name = "Admin User"
```

У структуры часть свойств могут быть переменными, а другая часть — константами.

## Функции для структур

В теле структуры можно добавлять функции — их называют **методами** структуры.

```
struct User {
    let id: Int
    let login: String
    var name: String

    func printInfo() {
        print("Имя: \\(name)\\nЛогин: \\(login)\\n")
    }
}
```

Если функция меняет одно или более свойств структуры, её помечают ключевым словом `mutating`. Например, в коде ниже метод `changeName` меняет свойство `name`:

```
struct User {
    let id: Int
    let login: String
    var name: String

    mutating func changeName(_ newName: String) {
        name = newName
    }
}
```

## Инициализаторы

 **Инициализатор** — это псевдо-функция с названием `init`, которая:

- принимает 0 или более параметров;

- присваивает всем неопциональным свойствам структуры начальное значение.

Инициализатор называют «псевдо-функцией», потому что в момент вызова не используют ключевое слово `func`. В момент вызова вместо `init` иногда используют имя типа, например, `User`, как в примере ниже.

По-умолчанию Swift генерирует для каждой структуры **дефолтный инициализатор** (от англ. "default" — значение по умолчанию). Именно дефолтный инициализатор мы неявно используем, когда пишем:

```
var adminUser = User(  
    id: 1,  
    login: "admin@service.com",  
    name: "Admin"  
)
```

Код выше можно записать и так:

```
var adminUser = User.init(  
    id: 1,  
    login: "admin@service.com",  
    name: "Admin"  
)
```

Но такой способ записи менее предпочтителен, так как слово `init` избыточно.

Дефолтный инициализатор принимает на вход столько же аргументов, сколько у свойств у структуры. Имена и тип аргументов совпадают с именами и типами свойств.

Дефолтный инициализатор для `User` эквивалентен инициализатору `init` ниже:

```
struct User {  
    let id: Int  
    let login: String  
    var name: String  
  
    init(id: Int, login: String, name: String) {  
        self.id = id  
        self.login = login  
        self.name = name  
    }  
}
```

Иногда определяют свой инициализатор (или несколько). В таком случае Swift **не** генерирует дефолтный инициализатор, чтобы избежать потенциальных конфликтов в коде.

Определим свой инициализатор, который:

- получает только `id` и `login`,
- создаёт экземпляр структуры `User`,
- свойству `name` присваивает значение `login` с заглавной буквы:

```
struct User {
    let id: Int
    let login: String
    var name: String

    init(id: Int, login: String) {
        self.id = id
        self.login = login
        self.name = login.capitalized
    }
}

var adminUser = User(id: 1, login: "admin")
// Создаст пользователя с `id = 1`, `login = "admin"` и `name = "Admin"`.
```

## Уровни доступа

Для свойств и функций, определённых в теле структуры, задают разные уровни доступа (англ. "access levels" — уровни доступа):

- `public` — свойство или функция видны всегда и везде.
- `internal` — свойство или функция видны только внутри данного модуля компиляции. Этот тип свойства и функции внутри структуры имеют по умолчанию. Писать этот уровень доступа явным образом — избыточно, поэтому делать это не рекомендуют.
- `private` — свойство или функция доступны только внутри тела структуры.
- `fileprivate` — свойство или функция доступны только внутри данного swift-файла.

Добавим приватное свойство `password` в структуру и специальный метод для его задания извне:

```
struct User {  
    let id: Int  
    let login: String  
    private var password: String  
  
    mutating func changePassword(_ newPassword: String) {  
        password = newPassword  
    }  
  
    func isCorrectPassword(_ password: String) -> Bool {  
        return self.password == password  
    }  
}
```

В коде мы скрыли свойство `password` от объектов, определяемых вне тела структуры. Так мы защитим пароль от случайного изменения снаружи. Ещё мы добавили метод для проверки правильности пароля `isCorrectPassword`.

## Яндекс Практикум