

Модуль 2 - урок 2: Структур

Структуры

```
struct Person {  
    var name: String  
}
```

Имена типов пишутся с заглавной буквы

Имена свойств и методов пишутся с маленькой буквы

Структуры

Доступ к значениям свойств

```
struct Person {  
    var name: String  
}  
  
let person = Person(name: "Жасмин")  
print(person.name)
```

Жасмин

Структуры

Добавление функционала

```
struct Person {  
    var name: String  
  
    func sayHello() {  
        print("Всем привет! Меня зовут \(name)!")  
    }  
}  
  
let person = Person(name: "Жасмин")  
person.sayHello()
```

Всем привет! Меня зовут Жасмин!

Экземпляры

```
struct Shirt {  
    var size: String  
    var color: String  
}  
  
let myShirt = Shirt(size: "XL", color: "синий")  
  
let yourShirt = Shirt(size: "M", color: "красный")
```

```
struct Car {  
    var make: String  
    var year: Int  
    var color: String  
  
    func startEngine() {...}  
  
    func drive() {...}  
  
    func park() {...}  
  
    func steer(direction: Direction) {...}  
}
```

```
let firstCar = Car(make: "Хонда", year: 2010, color: "синий")  
let secondCar = Car(make: "Форд", year: 2013, color: "чёрный")
```

```
firstCar.startEngine()  
firstCar.drive()
```

Инициализаторы (конструкторы)

```
let string = String.init() // ""  
let integer = Int.init() // 0  
let bool = Bool.init() // false
```

Инициализаторы (конструкторы)

```
var string = String() // ""  
var integer = Int() // 0  
var bool = Bool() // false
```


Инициализаторы (конструкторы)

Значения по умолчанию

```
struct Odometer {  
    var count: Int = 0  
}  
  
let odometer = Odometer()  
print(odometer.count)
```

0

Инициализаторы (конструкторы)

Поэлементные (почленные) инициализаторы

```
let odometer = Odometer(count: 27000)  
print(odometer.count)
```

27000

Инициализаторы (конструкторы)

Поэлементные (почленные) инициализаторы

```
struct Person {  
    var name: String  
}
```

Инициализаторы (конструкторы)

Поэлементные (почленные) инициализаторы

```
struct Person {  
    var name: String  
  
    func sayHello() {  
        print("Всем привет!")  
    }  
}  
  
let person = Person(name: "Жасмин") // Поэлементный конструктор
```

```
struct Shirt {  
    let size: String  
    let color: String  
}
```

```
let myShirt = Shirt(size: "XL", color: "синий") // Поэлементный конструктор
```

```
struct Car {  
    let make: String  
    let year: Int  
    let color: String  
}
```

```
let firstCar = Car(make: "Хонда", year: 2010, color: "синий") // Поэлементный конструктор
```

Инициализаторы (конструкторы)

Собственные инициализаторы

```
struct Temperature {  
    var celsius: Double  
}  
  
let temperature = Temperature(celsius: 30.0)
```

```
let fahrenheitValue = 98.6  
let celsiusValue = (fahrenheitValue - 32) / 1.8  
  
let newTemperature = Temperature(celsius: celsiusValue)
```

```
struct Temperature {  
    var celsius: Double  
  
    init(celsius: Double) {  
        self.celsius = celsius  
    }  
  
    init(fahrenheit: Double) {  
        celsius = (fahrenheit - 32) / 1.8  
    }  
}  
  
let currentTemperature = Temperature(celsius: 18.5)  
let boiling = Temperature(fahrenheit: 212.0)  
  
print(currentTemperature.celsius)  
print(boiling.celsius)
```

18.5

100.0

Методы экземпляров

```
struct Size {  
    var width: Double  
    var height: Double  
  
    func area() -> Double {  
        return width * height  
    }  
}  
  
var someSize = Size(width: 10.0, height: 5.5)  
  
let area = someSize.area() // Присвоено значение 55.0
```


Меняющие (mutating) методы

```
struct Odometer {  
    var count: Int = 0 // Присваивает значение по умолчанию свойству 'count'.  
}
```

Какие могут понадобиться действия (методы)?

- Накрутить пробег
- Скрутить пробег
- Обнулить пробег

```
struct Odometer {  
    var count: Int = 0 // Присваивает значение по умолчанию свойству 'count'.  
  
    mutating func increment() {  
        count += 1  
    }  
  
    mutating func increment(by amount: Int) {  
        count += amount  
    }  
  
    mutating func reset() {  
        count = 0  
    }  
}
```

```
var odometer = Odometer() // odometer.count по умолчанию равен 0  
odometer.increment() // odometer.count увеличивается на 1  
odometer.increment(by: 15) // odometer.count увеличивается до 16  
odometer.reset() // odometer.count сбрасывается в 0
```

Вычисляемые свойства

```
struct Temperature {  
    let celsius: Double  
    let fahrenheit: Double  
    let kelvin: Double  
}  
  
let temperature = Temperature(celsius: 0, fahrenheit: 32, kelvin: 273.15)
```

```
struct Temperature {  
    var celsius: Double  
    var fahrenheit: Double  
    var kelvin: Double  
  
    init(celsius: Double) {  
        self.celsius = celsius  
        fahrenheit = celsius * 1.8 + 32  
        kelvin = celsius + 273.15  
    }  
  
    init(fahrenheit: Double) {  
        self.fahrenheit = fahrenheit  
        celsius = (fahrenheit - 32) / 1.8  
        kelvin = celsius + 273.15  
    }  
  
    init(kelvin: Double) {  
        self.kelvin = kelvin  
        celsius = kelvin - 273.15  
        fahrenheit = celsius * 1.8 + 32  
    }  
}
```

```
let currentTemperature = Temperature(celsius: 18.5)  
let boiling = Temperature(fahrenheit: 212.0)  
let freezing = Temperature(kelvin: 273.15)
```

```
struct Temperature {  
    var celsius: Double  
  
    var fahrenheit: Double {  
        return celsius * 1.8 + 32  
    }  
}  
  
let currentTemperature = Temperature(celsius: 0.0)  
print(currentTemperature.fahrenheit)
```

32.0

ЗАДАЧА Добавить поддержку шкалы Кельвина

Изменить структуру для чтения температуры в градусах Кельвина

```
struct Temperature {  
    let celsius: Double  
  
    var fahrenheit: Double {  
        return celsius * 1.8 + 32  
    }  
  
}
```

Подсказка: градусы Кельвина — это градусы Цельсия + 273.15

```
struct Temperature {  
    let celsius: Double  
  
    var fahrenheit: Double {  
        return celsius * 1.8 + 32  
    }  
  
    var kelvin: Double {  
        return celsius + 273.15  
    }  
}  
  
let currentTemperature = Temperature(celsius: 0.0)  
print(currentTemperature.kelvin)
```

273.15

Наблюдатели свойств

```
struct StepCounter {  
    var totalSteps: Int = 0 {  
        willSet {  
            print("totalSteps будет установлено в \(newValue)")  
        }  
        didSet {  
            if totalSteps > oldValue {  
                print("Добавлено шагов: \(totalSteps - oldValue)")  
            }  
        }  
    }  
}
```


Свойства и методы типа

```
struct Temperature {  
    static var boilingPoint = 100.0  
  
    static func convertedFromFahrenheit(_ temperatureInFahrenheit: Double) -> Double {  
        return(((temperatureInFahrenheit - 32) * 5) / 9)  
    }  
}  
  
let boilingPoint = Temperature.boilingPoint  
  
let currentTemperature = Temperature.convertedFromFahrenheit(99)  
  
let positiveNumber = abs(-4.14)
```

Копирование

```
var someSize = Size(width: 250, height: 1000)
var anotherSize = someSize

someSize.width = 500

print(someSize.width)
print(anotherSize.width)
```

500

250

self

```
struct Car {  
    var color: Color  
  
    var description: String {  
        return "Цвет этой машины – \(self.color)."  
    }  
}
```

self. Когда не требуется

Не требуется, когда свойство или метод доступны в текущем объекте

```
struct Car {  
    var color: Color  
  
    var description: String {  
        return "Цвет этой машины — \$(color)."  
    }  
}
```

self. Когда требуется

```
struct Temperature {  
    var celsius: Double  
  
    init(celsius: Double) {  
        self.celsius = celsius  
    }  
}
```

Модуль 2 - урок 4:

Классы и наследование

```
class Person {  
    let name: String  
  
    init(name: String) {  
        self.name = name  
    }  
  
    func sayHello() {  
        print("Всем привет!")  
    }  
}
```

```
let person = Person(name: "Жасмин")  
print(person.name)  
person.sayHello()
```

Jasmine

Hello there!

Наследование

Базовый класс	Vehicle
Суперкласс	Bicycle
Подкласс	TandemBicycle

Наследование

Определение базового класса

```
class Vehicle {  
    var currentSpeed = 0.0  
  
    var description: String {  
        return "движется со скоростью \$(currentSpeed) км/ч"  
    }  
  
    func makeNoise() {  
        // ничего не происходит – произвольное средство передвижения может не издавать звуков  
    }  
}  
  
let someVehicle = Vehicle()  
print("Средство передвижения: \$(someVehicle.description)")
```

Средство передвижения: движется со скоростью 0.0 км/ч

Наследование

Создание подкласса

```
class SomeSubclass: SomeSuperclass {  
    // далее идёт определение подкласса  
}
```

```
class Bicycle: Vehicle {  
    var hasBasket = false  
}
```

Наследование

Создание подкласса

```
class Tandem: Bicycle {  
    var currentNumberOfPassengers = 0  
}
```

Наследование

Создание подкласса

```
class Tandem: Bicycle {  
    var currentNumberOfPassengers = 0  
}  
  
let tandem = Tandem()  
tandem.hasBasket = true  
tandem.currentNumberOfPassengers = 2  
tandem.currentSpeed = 22.0  
print("Тандем: \(tandem.description)")
```

Тандем: движется со скоростью 22.0 км/ч

Наследование

Переопределение методов и свойств

```
class Train: Vehicle {  
    override func makeNoise() {  
        print("Ty-ty!")  
    }  
}
```

```
let train = Train()  
train.makeNoise()
```

Ty-ty!

Наследование

Переопределение методов и свойств

```
class Car: Vehicle {  
    var gear = 1  
    override var description: String {  
        return super.description + " на передаче \$(gear)"  
    }  
}
```

Наследование

Переопределение методов и свойств


```
class Car: Vehicle {  
    var gear = 1  
    override var description: String {  
        return super.description + " на передаче \$(gear)"  
    }  
}  
  
let car = Car()  
car.currentSpeed = 25.0  
car.gear = 3  
print("Автомобиль: \$(car.description)")
```

Автомобиль: движется со скоростью 25.0 км/ч на передаче 3

Наследование

Переопределение инициализатора

```
class Person {  
    let name: String  
  
    init(name: String) {  
        self.name = name  
    }  
}  
  
class Student: Person {  
    var favoriteSubject: String  
}
```

 Class 'Student' has no initializers

Наследование

Переопределение инициализатора

```
class Person {  
    let name: String  
  
    init(name: String) {  
        self.name = name  
    }  
}  
  
class Student: Person {  
    var favoriteSubject: String  
    init(name: String, favoriteSubject: String) {  
        self.favoriteSubject = favoriteSubject  
        super.init(name: name)  
    }  
}
```

Класс - это ссылочный тип

Когда вы создаёте экземпляр класса:

- Swift возвращает адрес этого экземпляра
- Возвращённый адрес присваивается переменной

Когда вы присваиваете адрес экземпляра разным переменным:

- Каждая переменная содержит тот же адрес
- Измените один экземпляр, и все переменные будут указывать на изменённый экземпляр

```
class Person {  
    let name: String  
    var age: Int  
  
    init(name: String, age: Int) {  
        self.name = name  
        self.age = age  
    }  
}  
  
var jack = Person(name: "Джек", age: 24)  
var myFriend = jack  
  
jack.age += 1  
  
print(jack.age)  
print(myFriend.age)
```

25

25

```
struct Person {  
    let name: String  
    var age: Int  
}
```

```
var jack = Person(name: "Джек", age: 24)  
var myFriend = jack
```

```
jack.age += 1
```

```
print(jack.age)  
print(myFriend.age)
```

25

24

Поэлементные (почленные) инициализаторы

- Swift не создаёт поэлементных инициализаторов для классов
- Мы сами создаём инициализаторы

Класс или структура?

- Начинайте новый тип со структур
- Используйте классы:
 - Когда вы работаете с библиотекой, использующей классы
 - Когда вы хотите обратиться к тому же экземпляру из разных мест
 - Когда вы хотите использовать наследование