



Спринт 2, тема «Опционалы»

Яндекс Практикум

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

Опционалы

 **Опционалы** (от англ. "optional" — необязательный) в языке Swift помогают обрабатывать ситуации, когда значение переменной отсутствует, то есть равно `nil`.

Опциональный тип позволяет явно указать, что значение может отсутствовать. Это облегчает обработку ошибок и уменьшает количество аварийных завершений программы. В Swift использование опционала обязательно при работе с переменными, значения которых могут отсутствовать.

Определение опционалов

Чтобы превратить тип данных в опционал, нужно поставить знак вопроса `?` после названия типа, например `String?`, `Int?`:

```
var temperature: Int?  
print(temperature)
```

Результат:

`nil`

Чтобы проверить, пустая ли переменная, её значение сравнивают с `nil`:

```
var temperature: Int?  
temperature = 23  
if temperature != nil {  
    print(temperature) // напечатает "Optional(23)"  
}
```

Как устроены опционалы

В Swift опционал определён с помощью перечислений:

```
enum Optional<Wrapped> {  
    case none  
    case some(Wrapped)  
}
```

Wrapped — это тип данных, который мы «кладём в коробку» **Optional**.

Опционал — это перечисление с двумя кейсами:

- `none` означает, что в коробке пусто, то есть значение отсутствует.
- `some` значит, что значение находится в коробке. В опционале значение является ассоциированным значением для кейса `some`.

Распаковка

Чтобы использовать значение без обёртки, нужно «достать» его из опционала. Этот процесс называется «распаковка» (от англ. unwrapping).

Есть три основных механизма распаковки:

- optional binding,
- force unwrapping,

- implicit unwrapping.

Рассмотрим каждый из них подробнее.

Механизм optional binding

Optional binding (дословный перевод с англ. — «связывание опционала») позволяет проверить, есть ли в опционале значение. Если да, то можно получить значение опционала во временную переменную или константу.

Синтаксис конструкции:

```
if let <временная переменная> = <опционал> {  
    <//действия для непустого опционала>  
} else {  
    <//действия для пустого опционала>  
}
```

Пример с данными со спутника:

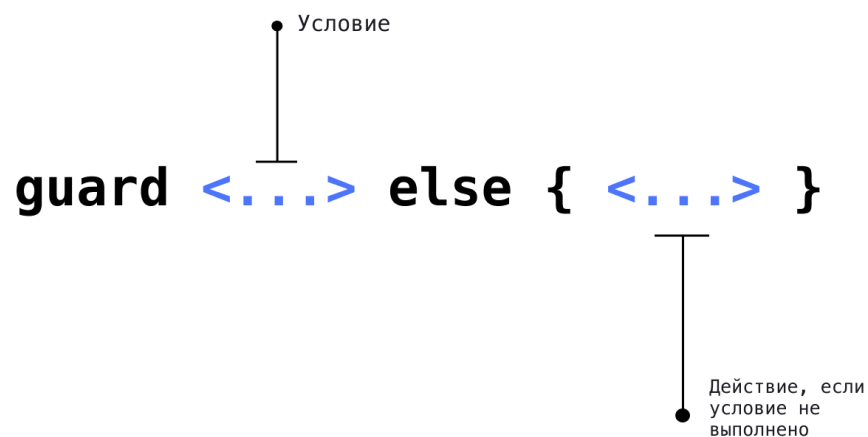
```
var message: String? = "Hello!"  
if let value = message { // проверим есть ли значение в `message` и если есть сохраним в `value`  
    print(value) // распечатает "Hello!"  
}
```

Такой вид распаковки — самый безопасный, мы рекомендуем использовать именно его.

Конструкция `guard`

Из-за того, что переменная `value` доступна только в области видимости `if`, нужно поместить всё использование этой переменной внутрь этой области видимости, это не всегда удобно.

Конструкция `guard` — это `if` наоборот. Вот её синтаксис:



Код, аналогичный варианту выше, с `guard` выглядит так:

```
guard let value = message else {  
    // Переменная `message` пуста. Её нельзя распаковать  
    return  
}
```

Если переменную `message` можно распаковать и сохранить распакованное значение в `value`, то блок кода после `else` не выполнится. Распакованное значение `value` можно использовать везде после конструкции `guard`.

Преимущества распаковки через `guard`:

- Заранее убеждаемся, что значение переменной с которой будем работать, не `nil`;
- Опционал распаковывается безопасно;
- Улучшаем читаемость кода, так как избавляемся от вложенности. Поясним: скобки `{ }` внутри других скобок `{ }` читаются сложнее, а в этой конструкции вся работа переменной находится в области видимости функции, а не только блока `guard`.

Обратите внимание: мы используем переменную `value` вне блока `guard`, так как блок `guard` выполняется, если условие не выполнилось (в отличие от `if`, который выполняется, если условие выполнилось).

Shadowing переменной

Чтобы избежать создания новых переменных при использовании конструкций `if let` и `guard let`, используют метод «затемнения» (от англ. "shadowing"):

```
if let message = message {
    ...
}
```

Новая константа `message` хранит распакованное значение. Поэтому внутри тела `if` можно обращаться к `message`, и это не будет опционалом.

Такой подход работает и с `guard`:

```
guard let message = message else {
    return
}
..
```

Для Swift последний `message`, который был объявлен, — это тот, что распакован. Поэтому когда мы обращаемся к `message`, значение берётся из уже распакованной новой переменной. Обратиться к старой переменной вне блока `guard` мы уже не сможем. Мы затемнили старую переменную и будто подставили вместо неё другую.

Nil-Coalescing оператор

Если значение переменной равно `nil`, то можно поставить значение по умолчанию. В этом поможет оператор `??`:

```
var message: String? = nil
print(message ?? "Сообщение не получено")
```

Эквивалентный код:

```
var message: String?
if let message = message {
    print(message)
} else {
    print("Сообщение не получено")
}
```

Небезопасная распаковка

В случаях, когда **точно известно**, что значение переменной не равно `nil`, иногда используют небезопасные механизмы распаковки.

Если значение переменной всё же окажется равным `nil`, то произойдёт внезапное зависание или закрытие программы. Это событие пользователи обычно воспринимают негативно.

Механизм force unwrapping

Force unwrapping (от англ. «принудительная распаковка») принудительно достаёт из опционала значение, не проверяя, пустое оно или нет.

Механизм реализуется с помощью восклицательного знака `!` после переменной-опционала:

```
var message: String?
message = "Hello, World"
print(message!)
```

В результате мы получим на экране: *"Hello, World"*, а не `Optional("Hello, World")`

Обратите внимание: если в переменной будет `nil`, то при попытке выполнить `force unwrapping` получится ошибка

Fatal error: Unexpectedly found nil while unwrapping an Optional value

И приложение упадёт.

Поэтому принудительную распаковку используют **тогда и только тогда**, когда точно известно, что в переменной есть значение.

Механизм `implicit unwrapping`

Механизм `implicit unwrapping` (англ. «неявная распаковка») подразумевает, что каждый раз когда мы обращаемся к помеченной специальным образом переменной, будет неявно происходить `force unwrap`. Неявно — потому, что явным образом в коде это не написано.

Чтобы реализовать этот механизм, нужно при создании переменной поставить после типа восклицательный знак `!`:

```
var temperature: Int!
```

Теперь мы если используем переменную, то Swift неявным образом подставит `force unwrapping`:

```
print(temperature) // выведет nil
temperature = 18
print(temperature) // выведет 18
```

Этот механизм упрощает код, но риск получить ошибку распаковки остаётся.

Цепочка опционалов

Создадим структуру, которая описывает сообщение, полученное с орбиты:

```
struct Message {
    let subject: String?
```

```
let text: String
let date: String?
}
```

Создадим один экземпляр сообщения:

```
let message = Message(subject: nil, text: "Hello", date: "01.01.2000")
```

Мы намеренно создали сообщение без темы.

Выведем длину темы сообщения:

```
print(message.subject?.count)
```

Здесь мы использовали цепочку опционалов и добавили вопросительный знак `?` после `subject`.

Так как `subject` является опциональным, то для доступа к его внутренним свойствам или функциям, нужно сначала убедиться, что он не `nil`. Если `subject` пустой, то у него нельзя посмотреть свойство `count`.

Цепочка для функций выглядит так же, как и для свойств:

```
print(message.subject?.uppercased())
```

Посмотрим, что получится, если всё сообщение тоже будет опционального типа:

```
let message: Message? = Message(subject: nil, text: "Hello", date: "01.01.2000")
print(message?.subject?.count)
```

Разберём по шагам, что происходит в коде:

1. Swift проверяет, есть ли значение в `message`. Если да, то распаковывает его. Если нет — цепочка прерывается, и результатом всего выражения будет `nil`.

2. Swift проверяет, есть ли значение в `message.subject`. Если да, то распаковывает его. Если нет — цепочка прерывается, и результатом всего выражения будет `nil`.
3. Swift берёт значение `message.subject.count` и делает его опциональным результатом всего выражения. Опциональным — потому, что цепочка могла в любой момент прерваться, что означало бы, что `count` пуст.

Это и есть цепочка опционалов. Если хотя бы одно звено имеет опциональный тип, то вся цепочка имеет опциональный тип. Ведь если оно имеет значение `nil`, то результатом всего выражения будет `nil`.

У звена в цепочке вызовов ставят вопросительный знак, когда оно имеет опциональный тип. Как в примере:

- `message` — опционал, значит, ставим после него `?`;
- `subject` — опционал, значит, ставим после него `?`.

Обратите внимание: не нужно ставить вопросительный знак в конце цепочки. `?` даёт доступ к внутренним свойствам или функциям, но у последнего звена они не нужны.

Например, если нужно вывести тему, то код будет такой:

```
print(message?.subject)
```

После `subject` нет `?`, так как доступ к её внутренним свойствам и функциям не нужен.

Небезопасные распаковки в цепочке

Обратите внимание: принудительные распаковки приводят к непредвиденным сбоям в работе приложения. Мы рекомендуем использовать конструкции `if let` и `guard let`.

Иногда принудительная распаковка полезна и используется в цепочке вызовов:

```
print(message!.subject)
```

Если `message` окажется пустым, получится ошибка *Unexpectedly found nil while unwrapping an Optional value*.

Яндекс Практикум