Спринт 1, тема «Переменные»

Яндекс Практикум

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

Переменные и константы

🔓**Переменная** — это *именованная область памяти*, адрес которой можно использовать для доступа к данным. С помощью переменных мы обращаемся к области памяти по простому и понятному имени.

«Объявить переменную» значит оповестить операционную систему о том, что вы собираетесь работать с данными в памяти.

Переменную, с которой мы хотим работать, «объявляют», чтобы система зарезервировала для неё конкретную область памяти.

Для объявления переменной в Swift используется ключевое слово `var`:

```
var price = 100
```

Чтобы изменить значение переменной `price` на `70`, используйте:

```
price = 70
```

Чтобы вывести значение переменной `price` на экран, используйте функцию `print`:

```
print(price)
```

🔒**Константы**, в отличие от переменных — это такие области памяти, значение которых задаётся лишь раз. В дальнейшем оно не может быть изменено.

Для объявления констант используется ключевое слово `let`:

```
let name = "Ivan"
```

Изменить значение константы после инициализации — нельзя:

```
name = "Vanya" // Swift не разрешит выполнить данную инструкцию
```

Типы данных

Тип данных — это характеристика, которая определяет, какие значения переменная может хранить и какие операции с ней можно выполнять.

В языке Swift **пять** базовых типов данных.

- Числовые — в свою очередь, делятся на:
  - `Int` — целочисленные (например, `100`);
  - `Float` и `Double` — числа с плавающей точкой (например, `3.1415926535`);
- Строковые — `String` (например, `"Ivan"`);
- Логические (булевые) — `Bool` (может принимать только `true` или `false` значения).

Эти типы называются базовыми, так как с их помощью можно создавать более сложные конструкции — структуры и классы.

`Int`

Одна переменная типа `Int` может хранить одно целое число, которое не больше 2147483647 и не меньше -2147483648.

Объявление переменной типа `Int` выглядит так:

```
var price: Int
price = 100
```

`Float` и `Double`

Числами с плавающей точкой называют десятичные дроби (например, 0.5 или 419.32). Часто на письме целую и дробную части разделяют запятой, но в программировании такой разделитель — точка.

Объявление переменных типа `Float` и `Double` выглядит так:

```
var price: Float
var timer: Double
price = 100.0
timer = 3.5
```

Если значение задаётся в момент объявления переменной с использованием константы, то тип переменной определяется исходя из типа константы:

```
var num = 3.5
print(type(of: num)) // Выведет на экран тип переменной num "Double"
```

`String`

Переменные этого типа позволяют хранить строки и всё, что помещается в обычную строку — например, ваше имя, адрес или название любимой книги.

Объявление переменной типа `String` выглядит так:

```
var name: String
name = "Ivan"
```

`Bool`

Это логический тип данных, который может принимать всего два значения: «истина» (`true`) или «ложь» (`false`).

Объявление переменной типа `Bool` выглядит так:

```
var hasProduct: Bool
hasProduct = true
```

Другие типы данных

- `UInt` — тот же `Int`, только он не умеет хранить отрицательные числа. Приставка ‘U’ (от английского слова “Unsigned” — «беззнаковый») означает, что тип может хранить только положительные значения. `UInt` может вместить в себя в два раза больше положительных чисел, чем `Int`, так как ему не нужно выделять значения для отрицательных чисел.
- `Int8` / `Int16` / `Int32` / `Int64` — тот же `Int`, но с меньшими или большими границами. Стандартный тип `Int` в большинстве случаев будет равен `Int32`.
- `UInt8` / `UInt16` / `UInt32` / `UInt64` — полагаем, вы сами догадались, что они умеют. (Если нет, подсказка: приставка `U` означает, что этот тип не может хранить отрицательные числа).

Границы этих типов можно проверить так:

```
print(UInt.min)
print(UInt.max)
```

Арифметические операции

*Любая операция содержит *операнд* и *оператор*, который необходимо применить к операндам. Операнд — это, то над чем производят операцию. Например, в операции сложения `2 + 3`, числа `2` и `3` являются операндами, а `+` является оператором, который нужно применить к этим операндам.

Бинарные арифметические операторы

В Swift, как и в любом языке программирования, есть стандартные математические операции:

- Сложение `+`

```
let a = 3
let b = 2
let c = a + b // в результате константе `c` будет присвоено значение `5`
```

- Вычитание `-`:

```
let a = 3
let b = 2
let c = a - b // в результате константе `c` будет присвоено значение `1`
```

- Умножение `*`:

```
let a = 3
let b = 2
let c = a * b // в результате константе `c` будет присвоено значение `6`
```

- Деление `/`:

```
let a = 4
let b = 2
let c = a / b // в результате константе `c` будет присвоено значение `2`
```

- Остаток от деления целых чисел `%` (его также называют «делением по модулю»):

```
let a = 3
let b = 2
let c = a % b // в результате константе `c` будет присвоено значение `1`, так как посл
е деления `3` на `2` остаток будет `1`
```

Приоритет арифметических операций, по сути, тот же, что вы изучали ещё в школе:

1. Скобки — наивысший приоритет;

2. Затем идут умножение и деление;

3. Остаток от деления;

4. Сложение и вычитание.

Унарный минус

Идея унарного минуса в том, чтобы взять некоторое значение и изменить его знак. То есть превратить положительное число в отрицательное, а отрицательное число — в положительное. Для этого нужно поставить знак минус `-` перед ним. Это равносильно умножению на `-1`:

```
let a = -3
let b = 3*(-1)
print(a == b)
```

Дробные числа

К типам `Float` и `Double` применяются те же арифметические операции, что и для целых чисел `Int`, за исключением остатка от деления.

> Обратите внимание: для дробных чисел операция остатка от деления % не предусмотрена.

Строки

Строки `String` можно складывать между собой. Но это не математическое сложение, а просто «склейка» двух строк или **конкатенация**. Она используется, когда нужно объединить несколько строк в одну. Это удобно делать, используя только знак `+`:

```
let greeting: String = "Hello, " + "Ivan"
```

Значением greeting будет строка `Hello, Ivan`.

Операторы

Операторы присвоения

Вы уже использовали оператор присвоения `=`, чтобы задать значение переменной:

```
var a: Int = 3
```

Здесь мы создали переменную `a` и присвоили ей значение `3`. Теперь переменная `a` хранит значение `3`. Это значит, что каждый раз, обращаясь к переменной `a`, мы берём то значение, которое сохранили в ней.

Сложное присвоение

Выражение `a = a + 5` сначала сложит `a + 5`, затем сохранит результат сложения обратно в `a`.

В Swift есть более удобная запись для таких операций: `a += 5`.

Она даёт то же самое, что и в предыдущем примере:

1. Берёт текущее значение из переменной `a`;

2. Прибавляет к нему 5;

3. Сохраняет результат обратно в `a`.

Это более лаконичный способ для записи сложения, без потери читаемости кода.

Для всех арифметических операций есть подобная запись:

- `+=` сложить и присвоить
- `-=` вычесть и присвоить
- `*=` умножить и присвоить
- `/=` делить и присвоить
- `%=` найти остаток от деления и присвоить

Например:

```
var a = 10

a += 10
> 20

a -= 10
> 10

a *= 3
> 30

a /= 5
> 6
```

Операторы сравнения

В Swift есть следующие операторы сравнения:

- Знак больше `>`
- Знак меньше `<`
- Знак равенства `==`

> Обратите внимание: знак равенства состоит из двух знаков равно `==`. Так нужно, потому что с помощью одного знака `=` мы присваиваем значение переменной.

Тернарный оператор

Тернарный оператор — это специальное сокращение, которое позволяет выполнить одно из двух выражений, в зависимости от истинности утверждения:

```
let a = 1.99
let b = 2.99
let minAB = a < b ? a : b
> 1.99
```

Операторы промежутков

В Swift есть специальные операторы для определения промежутка. Их три:

1. Открытый промежуток — это промежуток от `a` до `b`, включая `b`. Описывается так: `a...b`.

Например: `1000...5000` — диапазон цен от 1000 до 5000 включительно.

2. Полуоткрытый промежуток — от `a` до `b`, не включая `b`. Описывается так: `a..<b` (запись отличается от предыдущей лишь знаком `<`).

Например: `1..<5` — диапазон от 1 до 5, не включая 5 (то есть 1, 2, 3, 4).

3. Односторонний промежуток — тот, у которого не задан конец `1...` или начало `...10`.

Например: — `...5000` — диапазон до 5000 включительно.

> Обратите внимание: мы не задаём нижний предел, а лишь говорим о максимальном значении. — `1000...` — диапазон от

> 1000. И тут мы не задаём верхний предел, а только говорим о начальном значении.

Открытый промежуток можно сделать не включающим последнее значение, используя знак `<` - `..<5000` — диапазон до 5000, не включая 5000. Диапазоны — отличая способность языка Swift!

Приведение типов

Нельзя просто так взять и сложить друг с другом переменные типов `Int` и `Double`.

Если ваше арифметическое выражение содержит и целые, и дробные значения, без приведения типов не обойтись.

Следующий пример приведёт к ошибке:

```
let a: Int = 3
let b: Double = 3.5
print(a + b) // Ошибка. Нельзя складывать Int и Double
```

Выше вы получите ошибку: *Binary operator ‘+’ cannot be applied to operands of type ‘Int’ and ‘Double’*

Это можно перевести как > «Бинарный оператор ‘+’ не может быть применён к операндам разных типов ‘Int’ и ‘Double’».

Так происходит, потому что Swift — безопасный язык, который пытается защитить нас от ошибок.

Чтобы исправить эту ситуацию, используют так называемое **приведение типов**. Оно «меняет» тип переменной, пока идёт вычисление. Пример выше можно было бы написать так:

```
let a: Int = 3
let b: Double = 3.5
print(Double(a) + b)
```

Мы обернули переменную `a` в тип `Double` — ошибки больше нет. Теперь на время вычисления суммы переменная `a` имеет тип `Double`, и её можно складывать с другой переменной типа `Double`.

Яндекс Практикум