**Введение в язык Dart**

Dart – объектно-ориентированный язык программирования с сильной статической типизацией и поддержкой обобщенного программирования.

Он не поддерживает множественное наследование, то есть родителем производного класса может выступать только один базовый класс.

В тоже самое время, как и в языке программирования Java или C#, класс может реализовывать множество интерфейсов. По своему синтаксису Dart очень похож на семейство языков C (Си) – (C++, C#, Java, Kotlin и т. д.).

Dart – молодой язык программирования, который был впервые анонсирован корпорацией Google 10 октября 2011 года.

Первая версия языка увидела свет в ноябре 2013 года, далее отметилась версия 2.12 (март 2021), начиная с которой Dart поддерживает null-safety (нулевую безопасность, null-безопасность). В ее основе лежат следующие принципы проектирования − По умолчанию не допускает значения NULL. Если не указывается явно, что переменная может иметь значение NULL. Dart будет выдавать ошибки на этапе компиляции при присваивании такой переменной значения NULL. − Использование null-safety позволяет оптимизировать компилятор. Уменьшается не только количество ошибок, связанных с

присваиванием NULL, а также объем скомпилированного приложения

и повышается скорость его выполнения.

Таким образом, сейчас Dart – оптимизированный для клиентской части язык программирования, позволяющий вести разработку быстрых приложений на любой платформе. При этом он дает возможность использовать динамический тип в сочетании с проверками во время выполнения. Это особенно полезно при быстром прототипировании.

Dart на сегодняшний день является разработка графических приложений с помощью кроссплатформенного фреймворка Flutter. В частности, Flutter позволяет создавать клиентские веб-приложения, мобильные приложения под Android и iOS и десктопные приложения под Windows, MacOS и Linux. Таким образом, мы можем написать на языке Dart программу с общим кодом и скомпилировать эту программу сразу под все поддерживаемые платформы - Android, iOS, Windows, Linux, MacOS, веб-браузер. Но естественно фреймворком Flutter применение языка Dart не ограничивается.

Его синтаксис похож на синтаксис других си-подобных языков. При этом Dart - объектно-ориентированный язык. Все значения, которые используются в программе на Dart, представляют объекты.

Стоит отметить, что если Dart применяется для работы с фреймворком Flutter, то отдельно Dart можно не устанавливать, поскольку установочный пакет Flutter уже по умолчанию содержит инструменты для языка Dart. Однако поскольку Flutter - это все таки частный вариант использования Dart, поэтому рассмотрим отдельную установку Dart SDK.

**Асинхронность**: встроенная поддержка асинхронных операций через Future и Stream.

**Поддержка кроссплатформенности**: один код для Android, iOS, веба и настольных приложений.

**Области применения**

* **Мобильные приложения**: разработка через Flutter.
* **Веб-приложения**: компиляция в JavaScript для работы в браузере.
* **Серверные приложения**: использование Dart на сервере.
* **Работа с командной строкой**: скрипты и утилиты.

Основные элементы синтаксиса Dart

Переменные и типы данных

Объявление переменных

var name = 'Dart'; // Тип определяется автоматически int age = 25;

Основные типы данных:

* **int** — целые числа.
* **double** — числа с плавающей точкой.
* **String** — строки.
* **bool** — логические значения.

Операторы и выражения

Арифметические операторы: +, -, \*, /, %.

Логические операторы: &&, ||, !.

Операторы сравнения: ==, !=, <, >, <=, >=

|  |  |  |
| --- | --- | --- |
| **Раздел** | **Описание** | **Пример/Примечание** |
| **Что такое Dart?** | Dart — это язык программирования, созданный Google. Он используется для разработки мобильных, серверных и веб-приложений. | Dart активно используется для разработки приложений на платформе **Flutter.** |
| |  |  |  | | --- | --- | --- | | **Среда разработки** |  |  | | Для работы с Dart необходимы SDK и текстовый редактор (например, Visual Studio Code или IntelliJ IDEA). | SDK содержит компилятор Dart и другие инструменты |
| **Структура программы** | Программа на Dart состоит из основного файла с расширением .dart. | Основной файл программы может содержать функцию main(). |
| **Первая программа** | Программа начинается с функции main(), которая является точкой входа. | dart\nvoid main() {\n print('Hello, Dart!');\n}\n |
| **Переменные** | Переменные в Dart могут быть объявлены с использованием ключевых слов var, final, или const. | var name = 'Dart';, final age = 30;, const pi = 3.14; |
| **Типы данных** | Dart имеет несколько встроенных типов данных: int, double, String, bool, List, Map, и другие. | dart\nint age = 30;\ndouble price = 19.99;\nString name = 'Dart';\n |
| **Функции** | Функции в Dart объявляются с использованием ключевого слова void, если не возвращают значение, или с указанием типа возвращаемого значения. | dart\nint add(int a, int b) {\n return a + b;\n}\n |
| **Комментарии** | Комментарии в Dart могут быть однострочными // и многострочными /\* ... \*/. | // Это однострочный комментарий /\* Это\n многострочный\n комментарий \*/ |
| **Импорт библиотек** | Для использования дополнительных функций и классов можно импортировать библиотеки с помощью import. | dart\nimport 'dart:math';\n |
| **Управляющие структуры** | Dart поддерживает управляющие структуры, такие как if, else, for, while, do-while, и switch-case. | dart\nif (x > 10) {\n print('Большое число');\n}\n |
| **Обработка ошибок** | Для обработки ошибок используются блоки try-catch-finally. | dart\ntry {\n int result = 10 ~/ 0;\n} catch (e) {\n print('Ошибка: $e');\n}\n |
| **Комментарии документации** | Для создания документации можно использовать специальный синтаксис комментариев ///. | /// Это комментарий для генерации документации |
| **Библиотеки и пакеты** | Dart имеет встроенные и сторонние библиотеки и пакеты, которые можно использовать через pub.dev. | import 'package:flutter/material.dart'; |

"Программа начинается с функции main(), которая является точкой входа" означает следующее:

* **Функция main()** — это особая функция в языке программирования Dart (и во многих других языках), с которой начинается выполнение программы. Когда вы запускаете программу, первым делом выполняется именно эта функция.
* **Точка входа** — это технический термин, который обозначает место в программе, с которого начинается выполнение. В данном контексте это функция main(). Без этой функции программа просто не знает, с чего начать выполнение.

**Пример:**

dart

void main() {

print('Hello, Dart!');

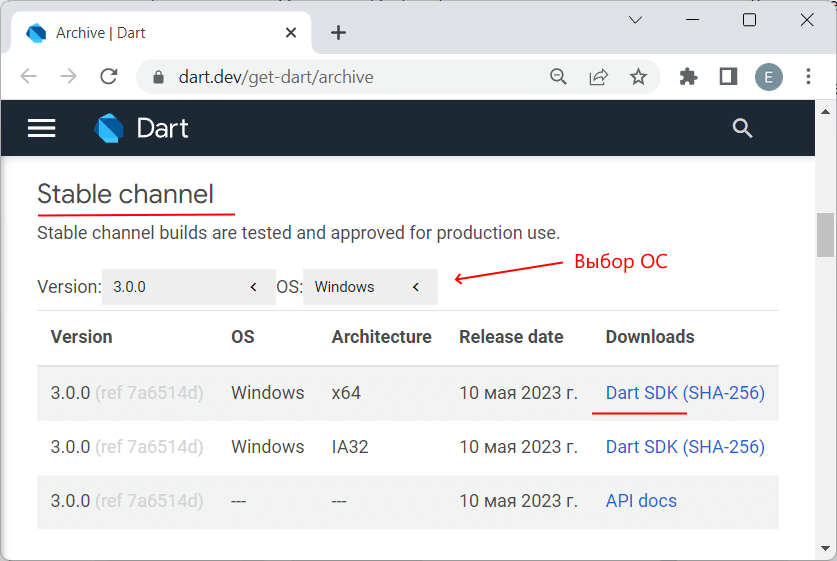
}

В этом примере:

* Когда программа запускается, сначала выполняется код, который находится внутри функции main().
* В данном случае, main() выводит на экран текст "Hello, Dart!".

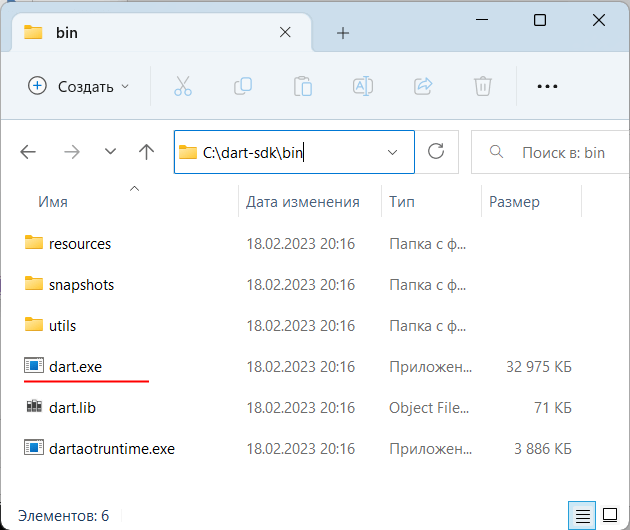
Таким образом, функция main() определяет порядок, в котором запускаются и выполняются действия в программе. Она играет ключевую роль, позволяя компилятору понять, что и как нужно делать при запуске программы.

**Установка Dart на Windows**

Для работы с Dart необходимо установить Dart SDK. Для этого нужно загрузить zip-архив с SDK с адреса <https://dart.dev/get-dart/archive> и распаковать его на жестком диске.

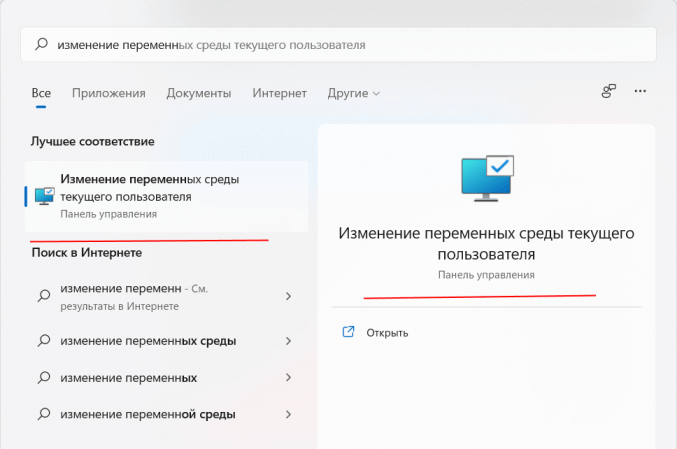
На странице загрузки есть пакеты для Windows, Linux, MacOS. В данном же случае мы будем использовать стабильную сборку, которая рекомендуется к использованию в разработки, поэтому при загрузке выбираем нужный пакет в секции "Stable channel".

Допустим, в моем случае Dart SDK распакован по пути C:\dart-sdk. Если мы перейдем в распакованном SDK к папке bin, то можем увидеть там такую программу как dart.exe:

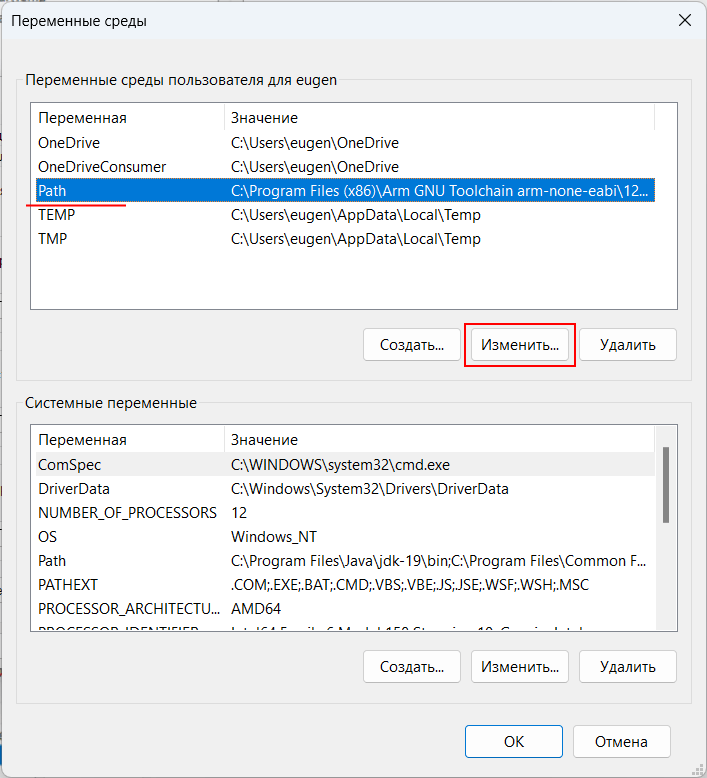


С помощью данной утилиты собственно и будут запускаться скрипты на языке Dart.

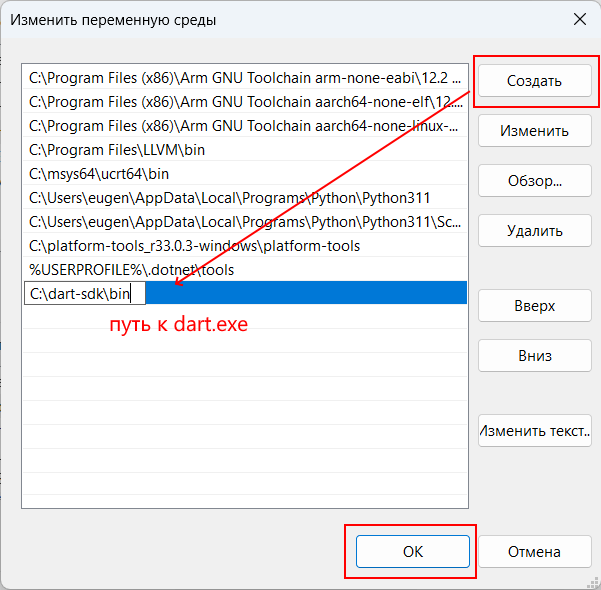
Чтобы каждый раз при запуске программы не надо было вводить полный путь до утилиты dart.exe, можно добавить путь к утилите в переменные среды. Для перехода к окну изменения переменных среды можно ввести в окно поиска "изменение переменных" и выбрать появившийся пункт "Изменение переменных среды текущего пользователя":



В открывшемся окне выберем пункт Path и нажмем на кнопку Изменить:



Далее нажмем на кнопку Создать и добавим путь к dart sdk:



**4. Настройка проекта**

1. **Создание нового проекта:**
   * Откройте терминал в VS Code (`Ctrl+``) и выполните команду

dart create my\_project

Эта команда создаст новый проект с базовой структурой. Папка my\_project будет содержать исходные файлы, необходимые для работы.

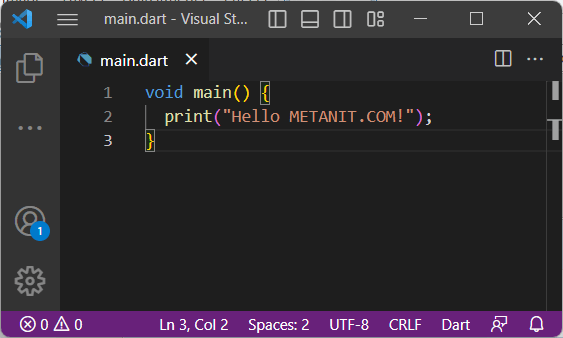
Первая программа

Напишем первую простейшую программу. Для этого определим на жестком диске каталог для файлов. Например, в моем случае пусть это будет каталог C:\dart. Создадим в каталоге новый текстовый файл, который будет называться main.dart (файлы с кодом Dart обычно имеют расширение \*.dart). Откроем данный файл в любом текстовом редакторе и определим в этом файле следующий простейший код:

void main() {

print("Hello METANIT.COM!");

}



Любое приложение на языке Dart должно иметь функцию, которая называется main. Эта функция имеет тип void и не принимает никаких параметров, поэтому после названия функции идут пустые скобки.

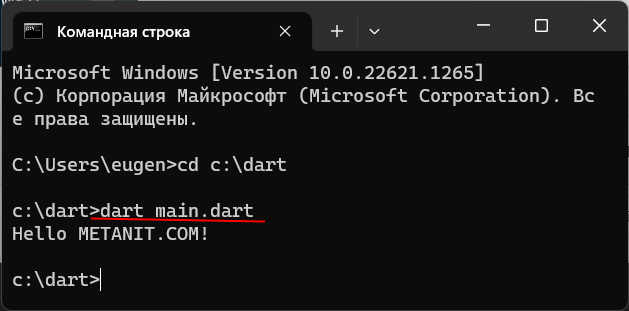
Тело функции помещается в фигурные скобки. В частности, в данном случае в функции main вызывается другая функция - print(), которая выводит на консоль некоторую строку. В данном случае это строка "Hello METANIT.COM!"

Теперь запустим данную программу. Для этого откроем командную строку/терминал и перейдем к папке с файлом main.dart с помощью команды cd.

Затем компиляции и запуска программы введем в командную строку следующую команду:

dart main.dart

То есть в Dart SDK вызывается программа dart (dart.exe), которой передается наш скрипт main.dart. И после это будет выполняться наша программа, которая выведет на консоль строку:

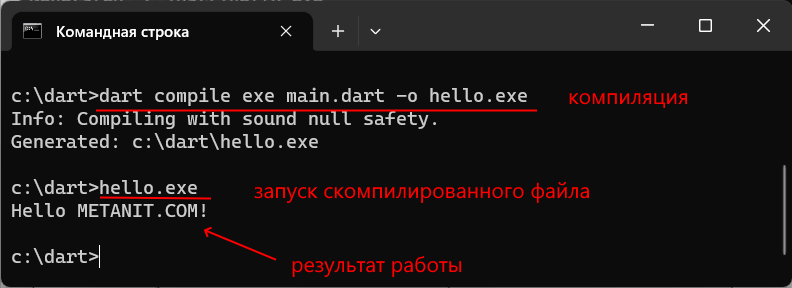


Компиляция программы

С помощью утилиты dart.exe мы можем запустить программу, однако как мы можем создать некий исполняемый файл, чтобы в любое время его можно было запускать (без обращения к dart.exe) и переносить на другой компьютер с той же операционной системой? Для этого утилите dart.exe наряду с исходным файлом программы надо передать команду compile exe. А после флага -o можно указать путь и название файла, который будет скомпилирован. Например: dart compile exe main.dart -o hello.exe

В данном случае будет компилировать файл hello.exe.

Так, скомпилируем и запустим файл программы:



**Установка расширения Dart для VS Code**

1. **Откройте Visual Studio Code.**
2. **Перейдите в раздел Extensions:**
   * Нажмите на значок Extensions на боковой панели или используйте комбинацию клавиш Ctrl+Shift+X.
3. **Установите расширение Dart:**
   * В строке поиска введите "Dart".
   * Выберите расширение "Dart" от Dart-Code и нажмите "Install".

**Шаг 4: Создание нового проекта**

1. **Откройте терминал в Visual Studio Code:**
   * Используйте меню "Terminal" -> "New Terminal" или комбинацию клавиш `Ctrl+``.
2. **Создайте новый проект:**
   * В терминале выполните команду:

bash

dart create first\_dart\_project

* + Эта команда создаст папку first\_dart\_project с базовой структурой проекта Dart.

1. **Откройте созданный проект:**
   * Перейдите в меню "File" -> "Open Folder..." и выберите папку first\_dart\_project.

**Шаг 5: Написание первой программы**

1. **Откройте файл main.dart:**
   * Найдите файл main.dart в папке lib (или bin, в зависимости от шаблона) и откройте его.
2. **Напишите код программы:**
   * Замените содержимое файла main.dart следующим кодом:

dart

// Функция main() - точка входа в программу.

void main() {

// Объявление переменной типа String

String name = 'Студент';

// Вывод сообщения на экран

print('Привет, $name!');

// Объявление переменной типа int

int age = 20;

// Вывод сообщения с использованием переменной

print('Ваш возраст: $age лет');

// Арифметическая операция

int nextYearAge = age + 1;

print('В следующем году вам будет $nextYearAge лет');

// Условное выражение

if (age >= 18) {

print('Вы совершеннолетний.');

} else {

print('Вы несовершеннолетний.');

}

}

1. **Сохраните файл:**
   * Нажмите Ctrl+S для сохранения изменений.

**Шаг 6: Запуск программы**

1. **Запустите программу:**
   * Откройте терминал в Visual Studio Code.
   * Выполните команду:

bash

dart run lib/main.dart

* + Если ваш файл находится в папке bin, команда будет:

bash

dart run bin/main.dart

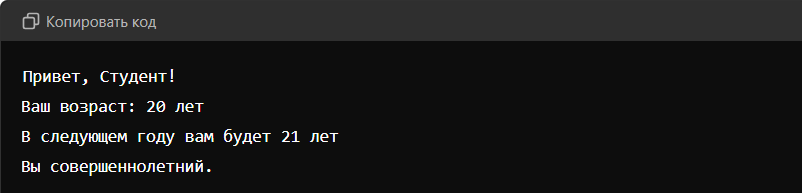
1. **Просмотрите результат:**
   * В терминале отобразится вывод программы:

Привет, Студент!

Ваш возраст: 20 лет

В следующем году вам будет 21 лет

Вы совершеннолетний.



**Объяснение программы**

* **Функция main()** — это основная функция, с которой начинается выполнение программы.
* **Переменная name** хранит строку 'Студент', которая используется для вывода приветствия.
* **Переменная age** хранит возраст пользователя, и с её помощью демонстрируются арифметические операции и условные выражения.
* **Функция print()** выводит информацию на экран.

**Правила именования**

При написании кода на Dart лучше придерживаться следующих

рекомендаций при объявлении переменных, функций, классов и их

методов:

1. При объявлении переменных, функций и методов классов

используется верблюжий стиль, а само название начинается с

маленькой буквы (lowerCamelCase). Для логического разделения слов

в объявляемой переменной необходимо использовать символ в

верхнем регистре: myCatName. Имя же объявляемого класса начинается

с большой буквы (UpperCamelCase): DailySchedule;

2. Нельзя использовать в начале объявляемого имени числовые

значения;

3. Регистр символов имеет значение. Так, например, var CHECK = 10; и

var check = 10; две совершенно разные переменные;

4. Не используйте в качестве имен переменных ключевые слова Dart;

5. Если имя переменной, функции и т.д. начинается с символа «\_», то она

является приватной (для импортирующего код модуля).

**Встроенные типы данных Dart**

Перед знакомством с встроенными типами данных давайте обратимся

к документации. В ней говорится, что все помещаемые в переменную

значения являются объектами, которые в свою очередь представляют собой

экземпляр класса. Такая концепция очень похожа на ту, что применяется в

языке программирования Python, в котором все является объектом. Таким

образом, даже числа, строки, функции и null – это объекты.

Существуют следующие встроенные типы данных:

− Числа (int, double);

− Строки (String);

− Логические значения (bool);

− Списки (List);

− Записи (Record);

− Множества (Set);

− Таблицы (Map);

− Руны (Rune);

− Символы (Symbol);

− Значение null (Null).

Но, прежде чем начнем знакомиться поближе с встроенными типами

данных и их объявлением, нам необходимо поговорить про комментарии.

Они в Dart делятся на 2 типа: однострочные и многострочные. В первом

случае используется «//», после чего идет комментарий, который не

переносится на следующую строку:

// комментарий

var a = 10; // еще один комментарий

Если комментарий будет занимать более 2-х строк, то его каждую

строку необходимо начинать либо с «//», либо использовать многострочный

формат записи комментария:

/\*

Сверх

длинный комментарий

\*/

var a = 10;

Комментарии можно использовать не только для того, чтобы

комментировать происходящее в коде. Например, посредством

комментариев можно исключить выполнение определенной строки или

блока кода (то есть закомментировать их). Но сильно этим увлекаться не

стоит, так как такой подход засоряет чистоту вашей кодовой базы, из-за

чего в последующем обязательно возникнут трудности у новых людей в

команде.

Строки же документации, которые позволяют использовать

инструмент dartdoc для автоматической генерации документации вашего

проекта начинаются с «///». Считается плохим тоном в тех частях, где

пояснения должны попасть в документацию использовать простые

комментарии, так как в этом случае они будут пропущены. Более подробно

с тем, как принято документировать код в проектах, разрабатываемых с

использованием Dart, можно ознакомиться в руководстве по

документированию проектов, расположенному на официальном сайте [4].

В данной книге комментарии используются для демонстрации того,

какой результат будет на выходе программы, что вернет та или иная строка

кода или дополнительного пояснения.

**Числа (int, double)**

В Dart всего два числовых типа данных: целочисленные (int) и

вещественные, т. е. с плавающей точкой (double).

Целочисленные значения типа int, в зависимости от платформы,

могут занимать в памяти не более 64 бит. В виртуальной машине Dart числа

типа int могут принимать значения в диапазоне от -263 до 263 - 1, а при

переводе кода в JavaScript используется диапазон значений, который

характерен для этого языка программирования: от -253 до 253 - 1

Числа с плавающей точкой типа double занимают в памяти 64 бита и

реализованы в соответствии со стандартом IEEE 754.

Теперь давайте рассмотрим, как можно объявлять переменные

числовых типов данных:

int a = 5;

int hex = 0xDEAFF; // 912127

var b = 10; // int

double c = 30.5;

var d = 1.1;

var exponents = 1.42e5; // 142000.0

Ключевое слово var перед именем переменной означает, что

компилятор Dart сам выведет тип объявляемой переменной в зависимости

от того, что разработчик напишет в правой части объявления после символа

«=».

Как и в других языках программирования со статической типизацией,

если мы объявили целочисленную переменную, то компилятор не даст нам

записать в эту переменную значение вещественного типа:

int a = 5;

a = 3.5; // error: A value of type 'double' can't be

//assigned to a variable of type 'int'.

var b = 2;

b = 3.5; // error

При этом вещественным переменным мы можем присваивать

целочисленные значения:

double a = 3.5;

a = 5;

var b = 2.2;

b = 3;

Типы int и double являются подтипами типа num, в котором в свою

очередь определены такие операции с числами, как: \*, /, + и -. Если мы

объявим переменную типа num и сначала присвоим ей целочисленное

значение, а после вещественное, это не будет считаться ошибкой, так как

работа с этими числами осуществляется через экземпляр базового класса

типа num:

num a = 3;

a = 5.3;

Особо внимательно нужно подходить к сравнению чисел, т.к., не ровен

час,

наткнуться на неприятные моменты при сравнении двух

вещественных значений из-за округления. Поэтому их, рекомендуют

сравнивать посредством >=, <= или метода compareTo, который возвращает:

− отрицательное число, если значение, с которым происходит

сравнение – больше;

− ноль, если они равны;

− положительное число, если меньше.

print(4.compareTo(5)); // -1

print(5.compareTo(4)); // 1

print(4.compareTo(4)); // 0

Ниже приведен ряд операций как над целочисленными переменными,

так и вещественными:

// Берем значение по модулю

print(-5.abs()); // 5

// Округление до большего и меньшего ближайшего целого

print(5.5.round()); // 6

print(5.5.floor()); // 5

// Число четное или нет?

print(5.isEven); // false

print(6.isEven); // true

// Число нечетное или нет?

print(5.isOdd); // true

print(6.isOdd); // false

// Число отрицательное или нет?

print((-5).isNegative); // true

print(6.isNegative); // false

// Представление числа в заданной системе счисления

print(15.toRadixString(2)); // 1111 - двоичная

print(15.toRadixString(8)); // 17 - восьмеричная

print(15.toRadixString(16)); // f - шестнадцатеричная

// Минимальное количество битов, необходимое

// для хранения целого числа

print(5.bitLength); // 3

print(22.bitLength); // 5

// Расчет наибольшего общего делителя

print(30.gcd(12)); // 6

print(4.gcd(2)); // 2

**Строки (String)**

Строки в Dart представляют собой последовательность символов в

кодировке UTF-16. Для их объявления (создания) могут использоваться как

одинарные, так и двойные кавычки:

String s1 = 'Мама мыла раму';

var s2 = "Мама мыла две рамы";

var s3 = '''Многострочная

строка''';

Для обращения к конкретному элементу строки по его индексу можно

использовать квадратные скобки:

print(s2[0]); // М, т.к. индексация начинается с нуля

Так как строки – неизменяемый тип данных (Immutable), то запись

вида: s2[0] = 'П' приведет к ошибке. В виду этого на основе одного

объекта должен быть создан другой, где в процессе создания производятся

необходимые изменения:

var s4 = 'П' + s2.substring(1); // Пама мыла две рамы

В этом случае использовалась операция конкатенация (операция

сложения между двумя строками). Из строки s2 были взяты все символы,

кроме первого, посредством метода substring. Он используется, когда

необходимо вырезать подстроку определенной длины. Для этого в метод

substring необходимо передать индекс первого и последнего элемента, на

основе которых сформируется новая строка. Например:

var s3 = 'П' + s2.substring(1, 9); // Пама мыла

Узнать длину строки можно, обратившись к атрибуту переменной

length:

print(s2.length); // 18

Для перевода всех символов в верхний или нижний регистр

используются следующие методы:

print(s2.toUpperCase()); // МАМА МЫЛА ДВЕ РАМЫ

print(s2.toLowerCase()); // мама мыла две рамы

Когда вызываете такие методы у строк, необходимо помнить, что они

не влияют на оригинальный объект, а возвращают преобразованное

значение, которое необходимо присвоить другой переменной для

последующей работы с ним. Более наглядно это объяснит следующий

пример:

var s2 = "Мама мыла две рамы";

s2.toUpperCase();

print(s2); // Мама мыла две рамы

var s3 = s2.toUpperCase();

print(s3); // МАМА МЫЛА ДВЕ РАМЫ

Теперь рассмотрим ситуацию, которая будет встречаться довольно

часто при написании кода – перевод числа в строку и наоборот:

// String -> int

var myInt = int.parse('34'); // строка в число

// String -> double

var myDouble = double.parse('11.45');

// int -> String

String s1 = 14.toString();

String s2 = myInt.toString();

// double -> String

String s3 = 3.14159.toStringAsFixed(2); // 2 числа после точки

String s4 = myDouble.toString();

Помимо конкатенации существует операция объединения, создающая

новую строку на основе заданной, где она дублируется указываемое

количество раз:

var s2 = "Oo";

print(s2\*4); // OoOoOoOo

Для посимвольного сравнения строк используйте следующий подход:

var s1 = 'Oo', s2 = 'Oo';

print(s2 == s1); // true - строки равны

print(s2 == 'oO'); // false

Когда же стоит задача лексикографически (в алфавитном порядке)

сравнить одну строку с другой, тут на помощь придет метод compareTo,

который вернет:

− отрицательное целое число, если текущая строка меньше строки, с

которой она сравнивается;

− положительное целое число, если текущая строка больше строки, с

которой она сравнивается;

− ноль, если строки равны.

var s1 = 'Мама', s2 = 'Папа';

print(s1.compareTo(s2)); // -1

print(s2.compareTo(s1)); // 1

print(s1.compareTo('Мама')); // 0

Для проверки того, входит ли символ или подстрока в строку, тип

String предоставляет метод contains:

var s1 = 'Вот те на!';

print(s1.contains('е')); // true

print(s1.contains('на')); // true

print(s1.contains('-\_-')); // false

Еще этот метод предоставляет возможность задать позицию, от

которой в основной строке будет осуществляться поиск (по умолчанию –

ноль):

print(s1.contains('В')); // true

print(s1.contains('В',0)); // true

print(s1.contains('В',1)); // false

Стоит только задать отрицательную позицию или больше длины

строки и программа экстренно завершится со следующей ошибкой:

print(s1.contains('В', 11)); // RangeError: Invalid value:

// Not in inclusive range 0..10

Иногда бывает необходимо найти номер индекса с первым или

последним вхождением символа или подстроки в строку:

var s1 = 'Мама мыла две рамы';

print(s1.indexOf('м')); // 2

print(s1.lastIndexOf('м')); // 16

print(s1.indexOf('м', 6)); // 16

print(s1.indexOf ('М', 6));

// -1, т.к. символ не найден (6 -> длина строки)

print(s1.lastIndexOf ('М', 6));

// 0, т.к. поиск начинается с конца (6 -> 0)

// Для обоих методов можно задавать индекс,

// с которого начнется поиск в строке

Строки могут содержать лишние пробелы (в начале, конце или сразу в

обоих местах), чтобы от них избавиться воспользуйтесь следующими

методами:

var s1 =' Ma ma ';

print(s1.trimLeft()); //Ma ma

print(s1.trim()); //Ma ma

print(s1.trimRight()); // Ma ma

Еще одна частая задача – замена одного символа или подстроки в

строке на другие:

var s1 = "Мама мыла рамы";

print(s1.replaceAll('мы', 'ру')); // Мама рула рару

print(s1.replaceAll('м', 'н')); // Мана ныла раны

print(s1.replaceFirst('м', 'М')); // МаМа мыла рамы

Чтобы разбить строку на несколько частей, воспользуйтесь методом

split:

var s1 = "Мама мыла рамы";

print(s1.split(' ')); // [Мама, мыла, рамы]

print(s1.split('л')); // [Мама мы, а рамы]

print(s1.split('мыла')); // [Мама , рамы]

В результате работы метода split вернется список подстрок,

количество которых будет зависеть от выбранного разделителя.

Для проверки же, осуществляется работа с пустой строкой или нет,

воспользуйтесь следующими свойствами экземпляра типа данных:

var myStr = 'Hello, world!';

print(myStr.isEmpty); // false

print(myStr.isNotEmpty); // true

myStr = '';

print(myStr.isEmpty); // true

print(myStr.isNotEmpty); // false

Логические значения (bool)

Переменные типа bool могут принимать только 2 значения: true и

false. Их объявление производится следующим образом:

bool a = false;

var b = true;

Вопросы:

1. Для замены какого языка программирования разрабатывался Dart?

2. Какие 2 платформы используются в компиляторе Dart? Для чего они

используются и какие между ними различия? 61

3. Какие ключевые особенности у языка программирования Dart?

4. Какие встроенные типы данных предоставляет Dart?

5. Какому числовому типу данных можно присваивать как

целочисленные, так и вещественные значения?

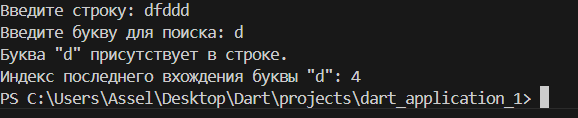
**Задания по строкам**

1. Напишите приложение, где пользователь вводит строку и букву,

наличие которой предстоит проверить в введенной строке. Выведите в

терминал полученный результат в терминал, а также индекс последнего

вхождения буквы в строку.



2. Напишите приложение, где пользователь вводит строку и букву.

Выведите в терминал длину строки, также индекс первого и последнего

вхождения буквы в строку.