Министерство образования Тульской области

Государственное профессиональное образовательное учреждение Тульской области «Донской политехнический колледж»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ДЛЯ ВЫПОЛНЕНИЯ ПРАКТИЧЕСКИХ РАБОТ

по междисциплинарному курсу

«МДК.01.03 Разработка мобильных приложений»

по теме: «Разработка приложения с использованием future, async/await, streams»

для обучающихся по программе подготовки специалистов среднего звена

по специальности 09.02.07 Информационные системы и программирование,

квалификация «Программист»

Автор:

С.М. Гвоздев, преподаватель ГПОУ ТО «ДПК»

2024 г.

Лист согласования:

Автор разработки:

Гвоздев Сергей Михайлович, преподаватель ГПОУ ТО «ДПК»

Рецензенты:

Евтехова О.А., заместитель директора по учебной и научно-методической работе ГПОУ ТО «ДПК»

Панченко Т.А., заместитель директора по организации образовательного процесса ГПОУ ТО «ДПК».

Филатова Е.А., старший методист ГПОУ ТО «ДПК».

Методические рекомендации предназначены для студентов 3 курса, обучающихся по специальности 09.02.07 Информационные системы и программирование, квалификация «Программист». Конкретные примеры данного пособия окажут практическую помощь при выполнении практических заданий по дисциплине «МДК.01.03 Разработка мобильных приложений»: «Разработка приложения с использованием future, async/await, streams» с использованием онлайн-редактора кода Dartpad.

СОГЛАСОВАНО

на заседании предметной (цикловой) комиссии
дисциплин профессионального цикла отделения
«Информационная безопасность и администрирование»
Протокол № 2

от «03» октября 2024 г.

Председатель ПЦК Гвоздев С.М.

СОДЕРЖАНИЕ

Введение 4
Задание № 1: Простой Future 6
Задание № 2: Использование async/await 6
Задание № 3: Обработка ошибок в Future 7
Задание № 4: Использование async/await с обработкой ошибок 7
Задание № 5: Простой Stream 8
Задание № 6: Использование async/await с Stream 8
Задание № 7: Обработка ошибок в Stream 9
Задание № 8: Использование async/await с обработкой ошибок в Stream 10
Задание № 9: Создание Future из Stream 11
Задание № 10: Использование async/await с Future из Stream 11
Задание № 11: Создание Stream из Future 12
Задание № 12: Использование async/await с Stream из Future 12
Задание № 13: Объединение двух Future 13
Задание № 14: Использование async/await с объединением Future 13
Задание № 15: Объединение двух Stream. 14
Задание № 16: Использование async/await с объединением Stream 14
Задание № 17: Преобразование Stream с использованием map 15
Задание № 18: Использование async/await с преобразованием Stream 15
Задание № 19: Фильтрация Stream с использованием where 16
Задание № 20: Использование async/await с фильтрацией Stream 16
Индивидуальные задания для закрепления материала 17

Введение

Методические рекомендации составлены в соответствии с рабочей программой ПМ.01 «Разработка модулей программного обеспечения для компьютерных систем» специальности 09.02.07 Информационные системы и программирование, квалификация «Программист».

В ходе освоения профессионального модуля обучающийся должен:

знать:

- основные этапы разработки программного обеспечения;
- основные принципы технологии структурного и объектно-ориентированного программирования;

уметь:

- разрабатывать приложения на языке Dart с использованием future, async/await, streams.

В процессе обучения по «МДК.01.03 Разработка мобильных приложений» у студентов формируется комплексное понимание и практические навыки, необходимые для создания функциональных и удобных мобильных приложений для различных платформ, таких как iOS и Android. Это достигается через изучение теоретических основ, включая архитектуру мобильных приложений, принципы работы мобильных операционных систем, основные компоненты приложений, такие как пользовательский интерфейс и бизнес-логика, а также различные паттерны проектирования.

На практике студенты осваивают инструменты и технологии разработки, включая языки программирования, интегрированные среды разработки (IDE), системы управления версиями и другие полезные инструменты. Они учатся проектировать интуитивно понятный и привлекательный пользовательский интерфейс, оптимизировать производительность приложений, обеспечивать безопасность данных и взаимодействие с различными сервисами и API.

Кроме того, студенты получают навыки тестирования и отладки мобильных приложений, а также знакомятся с процессом публикации приложений в магазинах приложений и монетизации. В целом, цель дисциплины – подготовить студентов к успешной карьере в области разработки мобильных приложений, обеспечив их глубоким пониманием технологий и практическими навыками, необходимыми для создания качественных и востребованных продуктов.

Задание № 1: Простой Future

Создайте *Future*, который через 2 секунды возвращает строку «Привет, Future!». Выведите эту строку в консоль.

Описание алгоритма решения:

1. Создайте *Future* с помощью конструктора *Future.delayed*.
2. Передайте ему длительность задержки (2 секунды) и анонимную функцию, которая возвращает строку «Привет, Future!».
3. Используйте *then* для вывода строки в консоль после завершения *Future*.

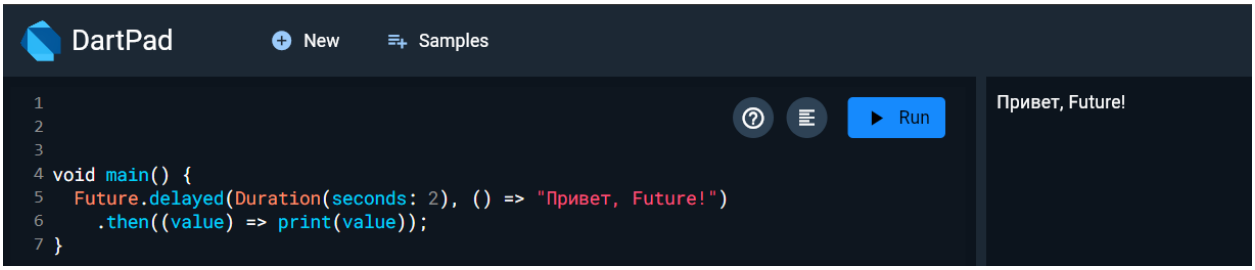

Рисунок №1 – Результат решения задания №1.

Задание № 2: Использование async/await

Перепишите предыдущее задание с использованием *async/await*.

Описание алгоритма решения:

1. Определите асинхронную функцию с помощью ключевого слова *async*.
2. Внутри функции используйте *await* для ожидания завершения *Future*.
3. Выведите результат *Future* в консоль.

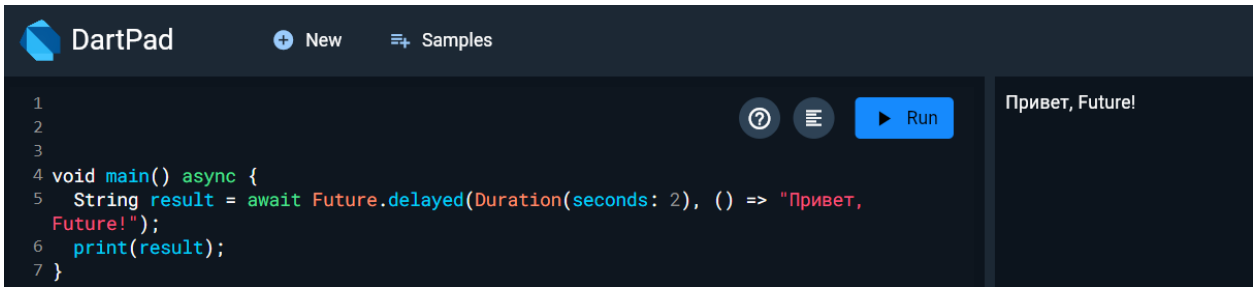

Рисунок №2 – Результат решения задания №2.

Задание № 3: Обработка ошибок в Future

Создайте *Future*, который через 2 секунды генерирует исключение. Обработайте это исключение и выведите сообщение об ошибке в консоль.

Описание алгоритма решения:

1. Создайте *Future* с помощью конструктора *Future.delayed*.
2. Передайте ему длительность задержки (*2 секунды*) и анонимную функцию, которая генерирует исключение.
3. Используйте *catchError* для обработки исключения и вывода сообщения об ошибке в консоль.

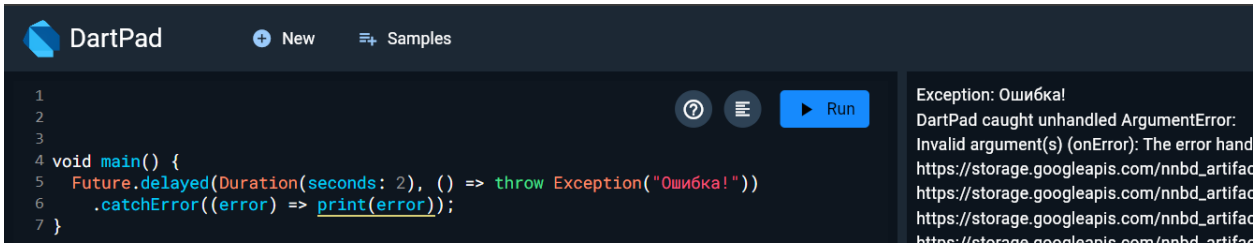

Рисунок №3 – Результат решения задания №3.

Задание № 4: Использование async/await с обработкой ошибок

Перепишите предыдущее задание с использованием *async/await* и блока *try-catch*.

Описание алгоритма решения:

1. Определите асинхронную функцию с помощью ключевого слова *async*.
2. Внутри функции используйте *await* для ожидания завершения Future.
3. Используйте блок *try-catch* для обработки исключения и вывода сообщения об ошибке в консоль.

Рисунок №4 – Результат решения задания №4.

Задание № 5: Простой Stream

Создайте *Stream*, который каждую секунду генерирует число от 1 до 5. Выведите каждое число в консоль.

Описание алгоритма решения:

1. Создайте Stream с помощью конструктора *Stream.periodic*.
2. Передайте ему длительность интервала (1 секунда) и анонимную функцию, которая генерирует число.
3. Используйте *take* для ограничения количества элементов в потоке.
4. Используйте *listen* для вывода каждого элемента в консоль.

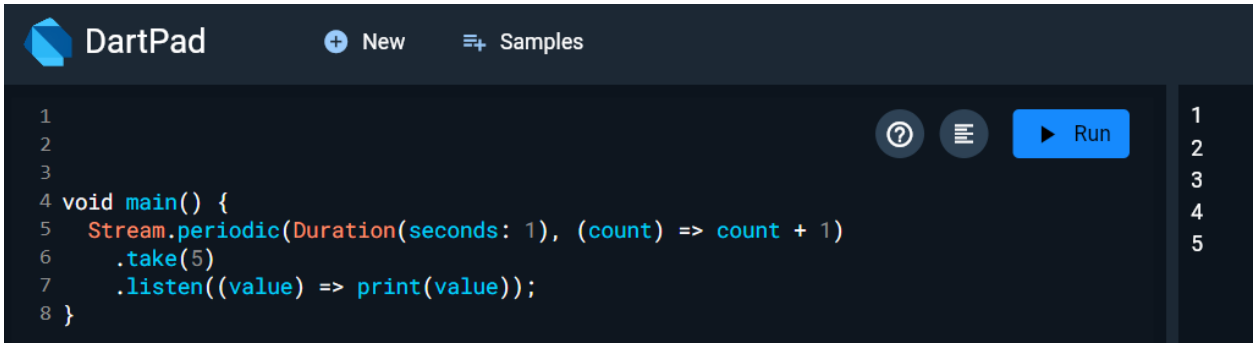

Рисунок №5 – Результат решения задания №5.

Задание № 6: Использование async/await с Stream

Перепишите предыдущее задание с использованием *async/await* и цикла *for-in*.

Описание алгоритма решения:

1. Определите асинхронную функцию с помощью ключевого слова *async*.
2. Внутри функции используйте *await for* для итерации по элементам Stream.
3. Выведите каждый элемент в консоль.

Рисунок №6 – Результат решения задания №6.

Задание № 7: Обработка ошибок в Stream

Создайте *Stream*, который каждую секунду генерирует число от 1 до 5, но на третьей итерации генерирует исключение. Обработайте это исключение и выведите сообщение об ошибке в консоль.

Описание алгоритма решения:

1. Создайте *Stream* с помощью конструктора *Stream.periodic*.
2. Передайте ему длительность интервала (1 секунда) и анонимную функцию, которая генерирует число или исключение.
3. Используйте *take* для ограничения количества элементов в потоке.
4. Используйте *listen* для вывода каждого элемента в консоль и обработки исключения.

Рисунок №7 – Результат решения задания №7.

Задание № 8: Использование async/await с обработкой ошибок в Stream

Перепишите предыдущее задание с использованием *async/await* и блока *try-catch*.

Описание алгоритма решения:

1. Определите асинхронную функцию с помощью ключевого слова *async*.
2. Внутри функции используйте *await for* для итерации по элементам *Stream*.
3. Используйте блок *try-catch* для обработки исключения и вывода сообщения об ошибке в консоль.

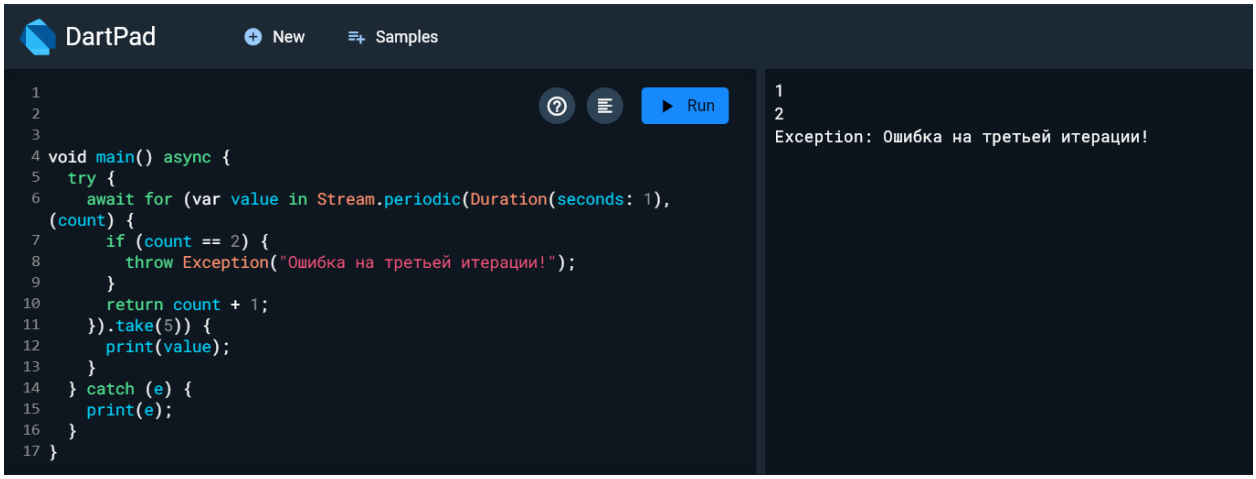

Рисунок №8 – Результат решения задания №8.

Задание № 9: Создание Future из Stream

Создайте *Stream*, который каждую секунду генерирует число от 1 до 5. Преобразуйте этот *Stream* в *Future*, который завершается, когда *Stream* завершается. Выведите сообщение «Stream завершен» в консоль.

Описание алгоритма решения:

1. Создайте *Stream* с помощью конструктора *Stream.periodic*.
2. Передайте ему длительность интервала (1 секунда) и анонимную функцию, которая генерирует число.
3. Используйте *take* для ограничения количества элементов в потоке.
4. Используйте *listen* для вывода каждого элемента в консоль.
5. Используйте *asFuture* для преобразования *Stream в Future*.
6. Используйте *then* для вывода сообщения «Stream завершен» в консоль после завершения *Stream*.

Рисунок №9 – Результат решения задания №9.

Задание № 10: Использование async/await с Future из Stream

Перепишите предыдущее задание с использованием *async/await*.

Описание алгоритма решения:

1. Определите асинхронную функцию с помощью ключевого слова *async*.
2. Внутри функции используйте *await for* для итерации по элементам *Stream*.
3. Выведите каждый элемент в консоль.
4. Используйте *await* для ожидания завершения *Stream*.
5. Выведите сообщение «Stream завершен» в консоль.

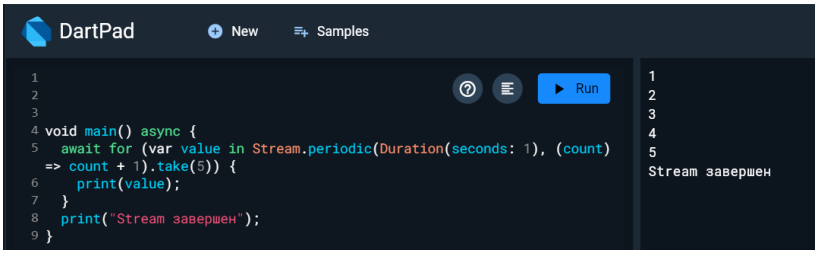

Рисунок №10 – Результат решения задания №10.

Задание № 11: Создание Stream из Future

Создайте Future, который через 2 секунды возвращает строку «Привет, Future!». Преобразуйте этот Future в Stream и выведите результат в консоль.

Описание алгоритма решения:

1. Создайте Future с помощью конструктора *Future.delayed*.
2. Передайте ему длительность задержки (2 секунды) и анонимную функцию, которая возвращает строку «Привет, Future!».
3. Используйте *asStream* для преобразования Future в Stream.
4. Используйте *listen* для вывода результата в консоль.

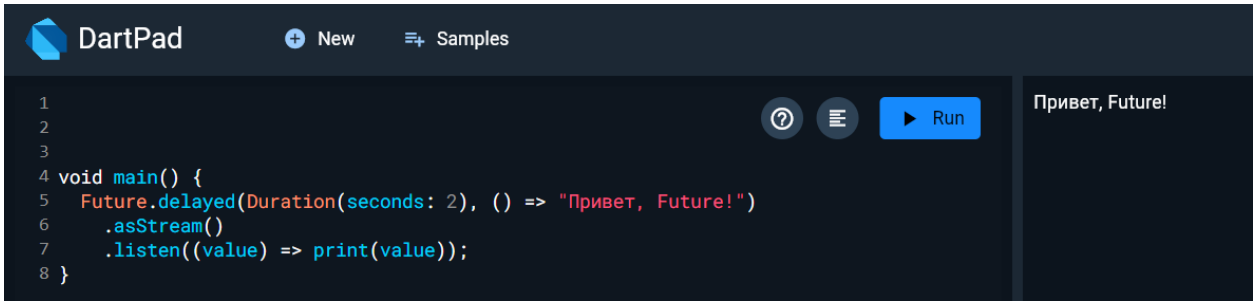

Рисунок №11 – Результат решения задания №11.

Задание № 12: Использование async/await с Stream из Future

Перепишите предыдущее задание с использованием *async/await*.

Описание алгоритма решения:

1. Определите асинхронную функцию с помощью ключевого слова *async*.
2. Внутри функции используйте *await for* для итерации по элементам *Stream*.
3. Выведите результат в консоль

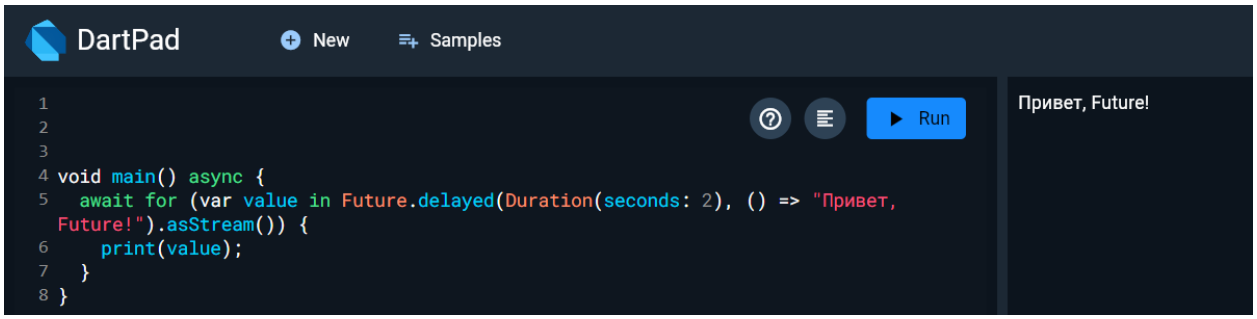

Рисунок №12 – Результат решения задания №12.

Задание № 13: Объединение двух Future

Создайте два Future: первый возвращает строку «Привет» через 2 секунды, а второй возвращает строку «Future!» через 3 секунды. Объедините результаты этих Future в одну строку и выведите её в консоль.

Описание алгоритма решения:

1. Создайте два *Future* с помощью конструктора *Future.delayed*.
2. Используйте *Future.wait* для ожидания завершения обоих *Future*.
3. Объедините результаты в одну строку и выведите её в консоль.

Рисунок №13 – Результат решения задания №13.

Задание № 14: Использование async/await с объединением Future

Перепишите предыдущее задание с использованием *async/awai*.

Описание алгоритма решения:

1. Определите асинхронную функцию с помощью ключевого слова *async*.
2. Внутри функции используйте *await* для ожидания завершения обоих *Future*.
3. Объедините результаты в одну строку и выведите её в консоль.

Рисунок №14 – Результат решения задания №14.

Задание № 15: Объединение двух Stream.

Создайте два *Stream*: первый генерирует числа от 1 до 3 с интервалом в 1 секунду, а *второй* генерирует числа от 4 до 6 с интервалом в 1 секунду. Объедините эти *Stream* в один и выведите все числа в консоль.

Описание алгоритма решения:

1. Создайте два *Stream* с помощью конструктора *Stream.periodic*.
2. Используйте *take* для ограничения количества элементов в каждом *Stream*.
3. Используйте *Stream.merge* для объединения двух Stream.
4. Используйте *listen* для вывода всех элементов.

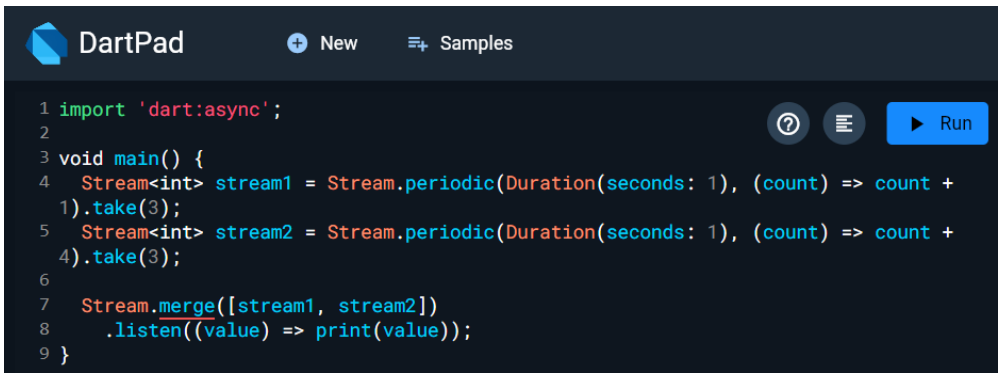

Рисунок №15 – Результат решения задания №15.

Задание № 16: Использование async/await с объединением Stream

Перепишите предыдущее задание с использованием *async/await*.

Описание алгоритма решения:

1. Определите асинхронную функцию с помощью ключевого слова *async*.
2. Внутри функции используйте *await for* для итерации по элементам объединенного *Stream*.

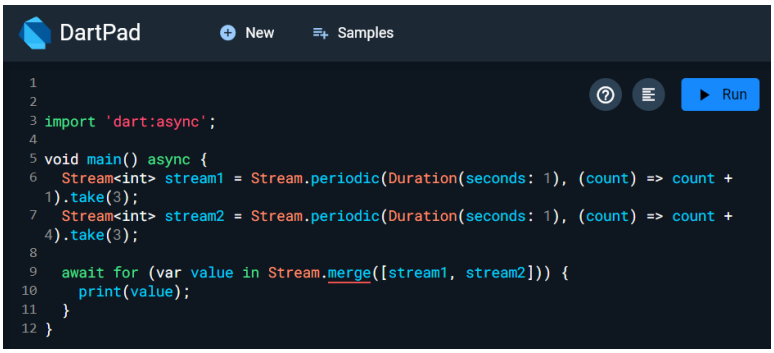

Рисунок №16 – Результат решения задания №16.

Задание № 17: Преобразование Stream с использованием map

Создайте *Stream*, который каждую секунду генерирует число от 1 до 5. Преобразуйте каждое число в строку, добавив к нему слово «число». Выведите результат в консоль.

Описание алгоритма решения:

1. Создайте *Stream* с помощью конструктора *Stream.periodic*.
2. Передайте ему длительность интервала (1 секунда) и анонимную функцию, которая генерирует число.
3. Используйте *take* для ограничения количества элементов в потоке.
4. Используйте *map* для преобразования каждого числа в строку.
5. Используйте *listen* для вывода результата в консоль.

Рисунок №17 – Результат решения задания №17.

Задание № 18: Использование async/await с преобразованием Stream

Перепишите предыдущее задание с использованием *async/await*.

Описание алгоритма решения:

1. Определите асинхронную функцию с помощью ключевого слова *async*.
2. Внутри функции используйте *await for* для итерации по элементам *Stream*.
3. Преобразуйте каждый элемент в строку.
4. Выведите результат в консоль

Рисунок №18 – Результат решения задания №18.

Задание № 19: Фильтрация Stream с использованием where

Создайте *Stream*, который каждую секунду генерирует число от 1 до 10. Отфильтруйте этот *Stream*, чтобы он содержал только четные числа. Выведите результат в консоль.

Описание алгоритма решения:

1. Создайте Stream с помощью конструктора *Stream.periodic*.
2. Передайте ему длительность интервала (1 секунда) и анонимную функцию, которая генерирует число.
3. Используйте *take* для ограничения количества элементов в потоке.
4. Используйте *where* для фильтрации только четных чисел.
5. Используйте *listen* для вывода результата в консоль

Рисунок №19 – Результат решения задания №19.

Задание № 20: Использование async/await с фильтрацией Stream

Перепишите предыдущее задание с использованием *async/await*.

Описание алгоритма решения:

1. Определите асинхронную функцию с помощью ключевого слова *async*.
2. Внутри функции используйте *await for* для итерации по элементам *Stream*.
3. Отфильтруйте только четные числа.
4. Выведите результат в консоль

Рисунок №20 – Результат решения задания №20.

Индивидуальные задания для закрепления материала

Базовые задания:

1. **Асинхронное чтение файла**: Напишите программу, которая асинхронно читает содержимое текстового файла и выводит его на экран. Используйте Future и async/await.
2. **Асинхронная загрузка изображения**: Создайте приложение, которое асинхронно загружает изображение по URL и отображает его на экране. Используйте Future и async/await.
3. **Асинхронная сортировка списка**: Напишите программу, которая асинхронно сортирует список чисел и выводит отсортированный список на экран. Используйте Future и async/await

Средний уровень:

1. **Асинхронная загрузка данных с API**: Напишите программу, которая асинхронно загружает данные с API (например, JSONPlaceholder) и выводит их на экран. Используйте Future и async/await.
2. **Асинхронное выполнение нескольких задач**: Напишите программу, которая асинхронно выполняет несколько задач (например, загрузка данных с разных API) и выводит результаты на экран. Используйте Future.wait.
3. **Асинхронная обработка потока данных**: Напишите программу, которая асинхронно обрабатывает поток данных (например, поток чисел) и выводит их на экран. Используйте Stream и async/await.

Продвинутый уровень:

1. **Асинхронная загрузка и обработка изображений**: Напишите программу, которая асинхронно загружает несколько изображений по URL, обрабатывает их (например, изменяет размер) и отображает на экране. Используйте Future, async/await и Canvas.
2. **Асинхронная загрузка и отображение данных в таблице**: Напишите программу, которая асинхронно загружает данные с API и отображает их в таблице на экране. Используйте Future, async/await и TableElement.
3. **Асинхронная загрузка и отображение данных в виде списка**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде списка на экране. Используйте Future, async/await и UListElement.
4. **Асинхронная загрузка и отображение данных в виде карточек**: Напишите программу, которая асинхронно загружает

данные с API и отображает их в виде карточек на экране. Используйте Future, async/await и DivElement.

5. **Асинхронная загрузка и отображение данных в виде диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде диаграммы на экране. Используйте Future, async/await и Canvas.
6. **Асинхронная загрузка и отображение данных в виде графика**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде графика на экране. Используйте Future, async/await и Canvas.
7. **Асинхронная загрузка и отображение данных в виде круговой диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде круговой диаграммы на экране. Используйте Future, async/await и Canvas.
8. **Асинхронная загрузка и отображение данных в виде столбчатой диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде столбчатой диаграммы на экране. Используйте Future, async/await и Canvas.
9. **Асинхронная загрузка и отображение данных в виде линейной диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде линейной диаграммы на экране. Используйте Future, async/await и Canvas.
10. **Асинхронная загрузка и отображение данных в виде точечной диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде точечной диаграммы на экране. Используйте Future, async/await и Canvas.
11. **Асинхронная загрузка и отображение данных в виде круговой диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде круговой диаграммы на экране. Используйте Future, async/await и Canvas.
12. **Асинхронная загрузка и отображение данных в виде столбчатой диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде столбчатой диаграммы на экране. Используйте Future, async/await и Canvas.
13. **Асинхронная загрузка и отображение данных в виде линейной диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде линейной диаграммы на экране. Используйте Future, async/await и Canvas.

14. **Асинхронная загрузка и отображение данных в виде точечной диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде точечной диаграммы на экране. Используйте Future, async/await и Canvas.
15. **Асинхронная загрузка и отображение данных в виде круговой диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде круговой диаграммы на экране. Используйте Future, async/await и Canvas.
16. **Асинхронная загрузка и отображение данных в виде столбчатой диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде столбчатой диаграммы на экране. Используйте Future, async/await и Canvas.
17. **Асинхронная загрузка и отображение данных в виде линейной диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде линейной диаграммы на экране. Используйте Future, async/await и Canvas.
18. **Асинхронная загрузка и отображение данных в виде точечной диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде точечной диаграммы на экране. Используйте Future, async/await и Canvas.
19. **Асинхронная загрузка и отображение данных в виде круговой диаграммы**: Напишите программу, которая асинхронно загружает данные с API и отображает их в виде круговой диаграммы на экране. Используйте Future, async/await и Canvas.

Дополнительные рекомендации:

1. Используйте комментарии для пояснения работы приложения.
2. Оформляйте код в соответствии с общепринятыми стандартами.
3. Тестируйте программу на различных наборах данных.
4. Представить результат работы в рукописном виде в рабочей тетради и в электронном формате с использованием Git-репозитория с предоставление QR-кода на репозиторий.

Критерии оценивания:

«Отлично» - выполнены все тренировочные задания и девять заданий из блока «продвинутый уровень».

«Хорошо» - выполнены все тренировочные задания и три задания из блока «средний уровень».

«Удовлетворительно» - выполнены все тренировочные задания и три задания из блока «базовый уровень».