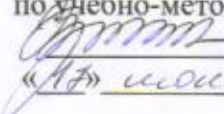


**ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ НОВОСИБИРСКОЙ ОБЛАСТИ
«Новосибирский электромеханический колледж»
(ГБПОУ НСО «НЭК»)**

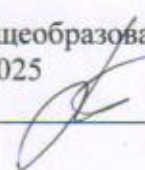
УТВЕРЖДАЮ

заместитель директора
по учебно-методической работе
 Е.О. Кравченко
«17» июля 2025 г.

**Методические указания
по организации самостоятельных работ**

В рамках изучения МДК и прохождения учебной практики

09.02.07 «Информационные системы и программирование»

Рассмотрено
на заседании кафедры общеобразовательных дисциплин
Протокол №10 от 16.06.2025
Заведующий кафедрой  Белимова В.Г.

Данные методические рекомендации помогут студентам выполнить самостоятельную работу в рамках изучения МДК и прохождения учебной практики

Согласовано:

Методист



Рахметова Т.Ю.

17.06. 2025 г.

Содержание

1. Введение.....	
1.1. Цель и задачи методических рекомендаций	
1.2. Область применения.....	
1.3. Нормативные ссылки.....	
2. Общие требования к выполнению и оформлению самостоятельных работ.....	
2.1. Виды самостоятельных работ.....	
2.2. Планирование и организация самостоятельной работы	
2.3. Методы и источники поиска информации	
2.4. Формы представления результатов самостоятельной работы.....	
3. Методические рекомендации по подготовке к практическим занятиям	
3.1. Изучение теоретического материала	
3.2. Подготовка к выполнению лабораторных и практических работ.....	
3.3. Работа с инструкциями и методическими материалами.....	
4. Методические рекомендации по подготовке к учебной практике (проектной деятельности).....	
4.1. Анализ технического задания и постановка задачи.....	
4.2. Поэтапное планирование проекта.....	
4.3. Самостоятельное освоение недостающих компетенций.....	
5. Технологическая карта (план разработки) проекта.....	
5.1. Назначение и структура технологической карты	
5.2. Пример заполнения технологической карты на основе сквозного проекта.....	
6. Критерии оценки самостоятельной работы	
6.1. Критерии оценки подготовки к занятиям.....	
6.2. Критерии оценки самостоятельной работы в рамках проекта.....	
7. Заключение.....	
7.1. Рекомендации по самоорганизации и тайм-менеджменту	
7.2. Часто встречающиеся трудности и пути их преодоления	

1. Введение

1.1. Цель и задачи методических рекомендаций

Настоящие методические рекомендации разработаны в дополнение к существующим указаниям по выполнению практических и лабораторных работ и направлены на систематизацию процесса самостоятельной учебной деятельности студентов. В условиях современного ИТ-образования именно самостоятельная работа становится ключевым звеном в формировании профессиональных компетенций, позволяя будущим специалистам не только осваивать учебный материал, но и развивать навыки самообучения, поиска информации, проектирования и решения нестандартных задач.

Основные цели рекомендаций:

- **Организационная:** создать четкую и понятную систему организации самостоятельной работы студентов на всех этапах обучения – от подготовки к отдельному занятию до выполнения комплексного учебного проекта (учебной практики).
- **Методическая:** предоставить студентам пошаговые алгоритмы и эффективные инструменты для самостоятельного освоения теоретического материала, поиска решений технических задач и реализации проектных этапов.
- **Мотивационная:** сформировать понимание важности самостоятельной работы как основы профессионального роста и успешного прохождения всех видов учебной деятельности, включая практики и итоговую аттестацию.
- **Контролирующая:** определить четкие критерии и формы оценки результатов самостоятельной работы, обеспечивая прозрачность и объективность оценивания.

Задачи рекомендаций:

1. Классифицировать виды самостоятельных работ в рамках учебного плана.
2. Описать эффективные методы планирования, поиска информации и организации рабочего времени.
3. Дать конкретные рекомендации по подготовке к практическим занятиям и лабораторным работам.
4. Предложить структурированный подход к самостоятельной подготовке в рамках учебной (проектной) практики.
5. Внедрить инструмент «Технологической карты (плана разработки) проекта», наглядно демонстрирующий логику поэтапного выполнения сложного задания (на примере сквозного проекта).
6. Установить единые критерии оценки самостоятельной деятельности.

1.2. Область применения

Данные рекомендации предназначены для студентов, обучающихся по специальности **09.02.07 «Информационные системы и программирование»**, и охватывают следующие виды самостоятельной работы:

- **Подготовка к аудиторным занятиям:** изучение лекционного материала, учебников, конспектов для успешного усвоения тем предстоящих практических и лабораторных работ по всем междисциплинарным курсам (МДК).
- **Подготовка к выполнению практических и лабораторных работ:** предварительный анализ задания, изучение инструкций, знакомство с программными инструментами и технологиями, которые будут использоваться на занятии.
- **Самостоятельное освоение компетенций в рамках учебной практики (УП):** выполнение индивидуальных или групповых проектов, требующих самостоятельного поиска информации, проектирования, кодирования, тестирования и документирования. Рекомендации фокусируются на ситуациях, когда студент

сталкивается с недостатком знаний или навыков для выполнения этапа проекта и должен самостоятельно спланировать, и реализовать процесс их освоения (реабилитационный/коррекционный план).

Рекомендации являются **обязательными к применению** при организации самостоятельной работы в рамках учебного процесса и должны использоваться совместно с «Методическими рекомендациями по выполнению практических и лабораторных работ».

1.3. Нормативные ссылки

При организации самостоятельной работы и оформлении ее результатов студентам следует руководствоваться следующими документами:

1. Федеральный государственный образовательный стандарт (ФГОС) по специальности 09.02.07 «Информационные системы и программирование».

2. Рабочие программы профессиональных модулей и междисциплинарных курсов (МДК).

3. «Методические рекомендации по выполнению практических и лабораторных работ» ГБПОУ НСО «Новосибирский электромеханический колледж».

4. Технические задания, календарные планы и иная проектная документация, выдаваемая в рамках учебной практики.

5. Общепринятые стандарты IT-индустрии в области управления проектами, документирования кода и обеспечения качества (Agile, Scrum, ГОСТ 34, PEP 8 для Python).

Следование данным рекомендациям позволит студентам перейти от пассивного восприятия информации к активному, осознанному и структурированному процессу самообразования, что является неотъемлемой чертой успешного IT-специалиста.

2. Общие требования к выполнению и оформлению самостоятельных работ

2.1. Виды самостоятельных работ

Самостоятельная работа студента (СРС) в рамках образовательного процесса по специальности «Информационные системы и программирование» подразделяется на следующие виды в зависимости от цели, содержания и формы контроля:

1. Самостоятельная работа по подготовке к аудиторным занятиям:

○ **Цель:** Актуализация и закрепление теоретических знаний для эффективной работы на практическом занятии.

○ **Содержание:** Изучение конспектов лекций, рекомендованных глав учебников, статей, видеоуроков; ознакомление с терминологией; предварительный анализ темы предстоящего занятия.

○ **Форма контроля:** Устный опрос в начале занятия, входной тест, проверка конспектов, активность в ходе обсуждения.

2. Самостоятельная работа по подготовке к выполнению практической/лабораторной работы:

○ **Цель:** Предварительное ознакомление с заданием, инструментами и технологиями для сокращения времени на освоение во время занятия.

○ **Содержание:** Детальное изучение методических указаний к работе; установка и первичная настройка необходимого программного обеспечения (IDE, СУБД, фреймворков); изучение синтаксиса или методов, которые будут применяться; продумывание алгоритма решения.

○ **Форма контроля:** Готовность к началу работы (установленное ПО), ответы на уточняющие вопросы преподавателя, наличие предварительного плана действий.

3. Самостоятельная работа по ликвидации пробелов в знаниях (реабилитационная):

○ **Цель:** Освоение компетенций, недостающих для выполнения текущего этапа учебной практики или проекта.

- **Содержание:** Работа по индивидуальному плану (например, «Плану освоения отсутствующих компетенций»). Включает самостоятельный поиск ответов на конкретные вопросы, изучение документации, просмотр tutorials, практическое выполнение тренировочных заданий.

- **Форма контроля:** Защита выполненного этапа плана (письменные ответы, демонстрация работающего кода, отчет по тестированию), успешное продолжение основного проекта.

4. Самостоятельная работа в рамках учебного проекта (проектная деятельность):

- **Цель:** Реализация отдельных модулей, этапов или функций сквозного проекта (например, «Цифровая поваренная книга», «Finance Manager», «Веб-приложение гостевой книги»).

- **Содержание:** Проектирование, кодирование, тестирование, отладка, документирование своей части работы. Самостоятельный выбор инструментов и алгоритмов в рамках ТЗ.

- **Форма контроля:** Проверка исходного кода, демонстрация работающего функционала, защита этапа проекта, итоговый отчет.

5. Самостоятельная работа по обобщению и систематизации знаний:

- **Цель:** Подготовка к дифференцированному зачету, экзамену или защите курсового проекта.

- **Содержание:** Повторение и структурирование материала модуля или курса, составление конспектов, схем, ментальных карт, решение комплексных задач.

- **Форма контроля:** Экзамен, зачет, защита курсовой работы.

2.2. Планирование и организация самостоятельной работы

Эффективная самостоятельная работа требует четкого планирования. Студентам рекомендуется:

1. **Анализ требований:** Внимательно изучить задание, ТЗ или план, выделить ключевые задачи и конечный результат.

2. **Декомпозиция:** Разбить крупную задачу на небольшие, последовательные и понятные шаги (подзадачи).

3. **Оценка времени:** Реалистично оценить время, необходимое для выполнения каждого шага. Заложить резерв на решение непредвиденных проблем (в среднем +30% к изначальной оценке).

4. **Составление плана:** Зафиксировать план в удобном формате (текстовый список, таблица в Excel/Google Sheets, карточки в Trello/YouTrack). В плане должны быть указаны: этап, конкретное действие, ожидаемый результат, срок выполнения, статус.

5. **Организация рабочего места и среды:** Обеспечить наличие необходимого ПО, доступ к интернету, литературе и другим ресурсам до начала работы.

6. **Соблюдение регулярности:** Выделять на самостоятельную работу постоянное время в своем расписании, избегая авралов и ночных бдений перед сдачей.

2.3. Методы и источники поиска информации

Ключевая компетенция IT-специалиста – умение быстро находить качественную информацию.

Основные источники информации:

1. **Официальная документация:** Первый и главный источник (документация Python, Qt, Microsoft Learn, MDN Web Docs). Отличается точностью и полнотой.

2. **Специализированные порталы и Q&A:** Stack Overflow, GitHub Issues, Хабр Q&A — для поиска решений конкретных ошибок и проблем.

3. **Учебная литература:** Учебники, рекомендованные в рабочих программах.

4. **Видеоуроки и курсы:** YouTube (каналы с качественным контентом), Stepik, Udemu — для визуального обучения и разбора сложных тем.

5. Коллегиальное обсуждение: Внутренние чаты группы, форумы колледжа (при наличии). Важно сначала попытаться решить проблему самостоятельно, а затем задавать четко сформулированный вопрос, приложив код и описание ошибки.

Методы поиска:

- Использование **ключевых слов** на английском языке (дает более релевантные результаты в IT-сфере).
- Поиск по **тексту ошибки** (копировать сообщение об ошибке из консоли в поисковую строку в кавычках).
- Использование **расширенного поиска** в Google/Yandex для фильтрации по дате, сайту.

2.4. Формы представления результатов самостоятельной работы

Форма представления результата зависит от вида работы и определяется преподавателем или ТЗ.

1. Письменный отчет/конспект: Для подготовки к занятию или изучения теории. Должен быть структурирован, содержать ключевые определения, схемы, примеры кода с комментариями.

2. Практический код: Файлы исходного кода (.py, .cpp, .php и т.д.), размещенные в системе контроля версий (Git) и сопровождаемые понятными коммитами.

3. Ответы на контрольные вопросы: Письменные развернутые ответы, демонстрирующие понимание темы, а не просто цитирование источника.

4. Заполненная «Технологическая карта» проекта: Таблица или документ, отражающий поэтапное выполнение проекта с указанием выполненных действий, проблем и результатов (см. Раздел 5).

5. Демонстрация работающей программы: Устная защита с запуском программы, показом интерфейса и объяснением логики работы.

6. Тестовая документация: Чек-листы, тест-кейсы, баг-репорты, составленные в рамках самостоятельной подготовки к тестированию.

Важно: Любой результат должен быть **оформлен аккуратно, подписан** (ФИО, группа) и **сдан в установленный срок**. Электронные файлы должны иметь понятные имена (например, Иванов_ИИ_ИСП-33_Отчет_Этап1.docx).

3. Методические рекомендации по подготовке к практическим занятиям

Систематическая и качественная подготовка к практическим занятиям является **ключевым фактором успешного освоения специальности** и формирования профессиональных навыков. Цель подготовки — не механическое выполнение инструкций, а **осознанное усвоение материала**, позволяющее самостоятельно решать профессиональные задачи. Данный раздел описывает алгоритм эффективной подготовки к занятиям по всем междисциплинарным курсам (МДК).

3.1. Изучение теоретического материала

Теоретическая база — фундамент для любой практической работы. Подготовка начинается с повторения и углубления теории.

План изучения теоретического материала:

1. Конспект лекции: Внимательно прочтите конспект предстоящего занятия. Выделите ключевые термины, определения, формулы, принципы. Если конспект отсутствует или недостаточно подробен, восполните пробелы, используя рекомендованную литературу.

2. Работа с учебником: Изучите соответствующие разделы учебника, указанные в рабочей программе или рекомендованные преподавателем. Не ограничивайтесь поверхностным чтением — делайте выписки, отмечайте сложные моменты для обсуждения на занятии.

3. **Специализированные ресурсы:** Для IT-дисциплин критически важно обращаться к **официальной документации** (Python Documentation, PHP Manual, Qt Docs). Используйте ее для уточнения синтаксиса функций, методов, принципов работы библиотек.

4. **Визуализация:** Для сложных тем (алгоритмы, архитектурные паттерны, взаимодействие процессов) создавайте схемы, блок-схемы, диаграммы классов или ментальные карты. Это помогает структурировать информацию и лучше понять взаимосвязи.

5. **Самоконтроль:** После изучения материала попробуйте ответить на контрольные вопросы (если они есть), сформулировать краткое резюме темы своими словами или объяснить материал воображаемому собеседнику.

3.2. Подготовка к выполнению лабораторных и практических работ

Цель этого этапа — не просто прочитав задание, а **сформировать готовый план действий**, чтобы на самом занятии сосредоточиться на реализации, а не на понимании базовых инструкций.

Пошаговый алгоритм подготовки к практической работе:

1. Предварительный анализ задания:

- Получите и внимательно прочтите методические указания к работе (файл задания).
- Выделите и выпишите **цель работы и перечень конкретных задач**.
- Определите **конечный результат**, который должен быть получен (работающая программа, заполненная таблица, отчет о тестировании).
- Оцените объем и сложность работы.

2. Анализ технических требований и инструментов:

- Выпишите все необходимые программные средства (IDE, компилятор, СУБД, фреймворки, библиотеки).
- **Установите и настройте все ПО заранее.** Проверьте его работоспособность на простом примере (например, запуск "Hello, World!").
- Если работа предполагает использование специфических библиотек (pytest, PyQt, PDO), установите их в виртуальное окружение и выполните минимальный тестовый скрипт.

3. Изучение новых технологий/синтаксиса:

- Если в работе встречаются незнакомые технологии, команды или методы, **изучите их до занятия**.
- Используйте официальную документацию, статьи, короткие видеоуроки (5- 10 минут).
- **Выполните микро-практикум:** Напишите небольшой отдельный скрипт, чтобы опробовать новую функцию, метод или запрос. Например, если в задании нужно использовать JOIN в SQL, напишите и выполните простой JOIN-запрос в отдельном файле.

4. Проектирование решения:

- Для работ по программированию продумайте архитектуру решения: какие функции/классы нужны, как они будут взаимодействовать.
- Для работ по тестированию заранее набросайте структуру тест-кейсов или чек-листа.
- Создайте **алгоритм выполнения** в виде последовательности шагов (псевдокод, комментарии в будущем коде).
- Определите **потенциально сложные места**, где могут возникнуть ошибки, и подготовьте вопросы преподавателю.

5. Подготовка рабочего пространства:

- Создайте папку для проекта с понятным именем.
- Инициализируйте Git-репозиторий (если это требуется или является лучшей практикой).
- Создайте необходимую структуру файлов (например, main.py, database.py, tests/).

Результатом подготовки должна стать готовность к немедленному началу практической части занятия, наличие четкого плана и предварительно настроенной рабочей среды.

3.3. Работа с инструкциями и методическими материалами

Умение работать с технической документацией — профессиональный навык.

Правила работы с методическими указаниями:

1. **Читайте внимательно и вдумчиво.** Не пропускайте разделы «Цель», «Теоретическая часть», «Контрольные вопросы» — они содержат ключевую информацию для понимания сути работы.

2. **Выделяйте ключевые действия.** При первом прочтении карандашом или маркером отмечайте конкретные задачи, которые нужно выполнить (например, «создать функцию...», «написать запрос...», «заполнить таблицу...»).

3. **Соотносите теоретическую часть с практической.** Понимайте, какая теоретическая концепция лежит в основе каждого практического шага. Это превращает механическое выполнение в осознанное обучение.

4. **Составляйте список вопросов.** Если какой-то пункт инструкции непонятен, формулируйте конкретный вопрос. На занятии вы сможете задать его преподавателю, экономя время.

5. **Используйте материалы как справочник во время работы.** Не стесняйтесь возвращаться к инструкции на каждом этапе для проверки правильности своих действий.

Эффективная подготовка экономит время на занятии, снижает стресс, повышает глубину понимания материала и, как следствие, — итоговую оценку за работу.

4. Методические рекомендации по подготовке к учебной практике (проектной деятельности)

Учебная практика (УП) является кульминацией учебного процесса, где студент применяет совокупность знаний и умений для решения комплексной профессиональной задачи — разработки программного продукта. Успех на практике на 80% зависит от качества **самостоятельной проектной деятельности**. Данные рекомендации помогут организовать этот процесс системно и эффективно.

4.1. Анализ технического задания и постановка задачи

Первый и критически важный этап – глубинное понимание того, **что необходимо создать**. Ошибки на этом этапе приводят к переделке работы и срыву сроков.

Алгоритм анализа ТЗ (Технического задания):

1. **Сквозное прочтение:** Внимательно прочитайте весь документ ТЗ от начала до конца, не вдаваясь в детали. Сформируйте общее представление о проекте: тип приложения (desktopное, веб, консольное), его основное назначение, целевая аудитория.

2. **Выделение ключевых требований (Mandatory Features):**

- Составьте список **обязательного функционала**, без которого проект не может быть принят (например, «Регистрация пользователей», «Добавление рецепта», «Формирование списка покупок»).

- Выделите **нефункциональные требования**: используемые технологии (Python 3.9+, PyQt6, SQLite), требования к интерфейсу, необходимость наличия документации.

3. **Детализация функционала:** По каждому пункту обязательного функционала задайте уточняющие вопросы (себе или руководителю):

- **Для данных:** Какие именно данные нужно хранить? Какие у них поля, типы, ограничения? (На основе этого будет проектироваться БД).

- **Для действий:** Как именно должно работать то или иное действие с точки зрения пользователя? (Например, "Добавить в список покупок" — отметить галочкой и нажать кнопку?).

- **Для логики:** Каковы правила обработки данных? (Например, при удалении рецепта нужно ли удалять связанные с ним ингредиенты из списка покупок?).

4. **Визуализация и уточнение:** Создайте эскиз (скетч) основных экранов приложения (главное окно, окно добавления, окно деталей). Это поможет выявить неочевидные элементы интерфейса и логики.

5. **Формулировка собственной постановки задачи:** Своими словами, максимально конкретно запишите, что вы должны сделать в рамках практики. Пример: «Разработать на Python с использованием PyQt6 и SQLite desktopное приложение «Цифровая поваренная книга», позволяющее пользователю создавать, просматривать, редактировать и удалять рецепты, а также формировать список покупок на основе выбранных ингредиентов».

4.2. Поэтапное планирование проекта

Разработка без плана — путь к хаосу. Используйте принцип декомпозиции (разбиения большой задачи на мелкие).

Рекомендуемая структура плана проекта (рабочий бэклог):

1. **Подготовительный этап (Environment Setup):**

- Настройка IDE и рабочего окружения.
- Установка и проверка необходимых библиотек.
- Создание структуры папок проекта (/src, /database, /tests, /docs).
- Инициализация Git-репозитория.

2. **Этап проектирования (Design):**

- Проектирование схемы базы данных (ER-диаграмма).

- Проектирование архитектуры приложения: выделение основных модулей (UI, бизнес-логика, слой данных).
- Создание макетов интерфейса (wireframes).
- 3. **Этап реализации базового функционала (Core Implementation):**
 - Реализация модуля работы с БД (подключение, модели, CRUD для основных сущностей).
 - Разработка базового UI главного окна.
 - Связывание UI и БД: реализация отображения данных, добавления новой записи.
- 4. **Этап реализации дополнительного функционала (Features Implementation):**
 - Реализация остальных CRUD-операций (редактирование, удаление).
 - Добавление специфической бизнес-логики (например, формирование списка покупок).
 - Реализация дополнительных окон и функций (поиск, фильтрация).
- 5. **Этап тестирования и отладки (Testing & Debugging):**
 - Модульное тестирование критических функций.
 - Интеграционное и ручное тестирование по сценариям.
 - Исправление выявленных ошибок (багов).
- 6. **Этап сборки и документирования (Deployment & Documentation):**
 - Сборка исполняемого файла (.exe).
 - Написание финальной документации (руководство пользователя, комментарии в коде).
 - Подготовка презентации и отчета по практике.

Используйте инструменты для управления: простую таблицу (Excel/Google Sheets), доску в Trello, задачи в GitHub Projects. По каждому пункту указывайте статус (To Do, In Progress, Done).

4.3. Самостоятельное освоение недостающих компетенций

В ходе практики вы неизбежно столкнетесь с задачами, для решения которых у вас недостаточно знаний. **Умение самостоятельно ликвидировать эти пробелы — ключевой результат практики.**

Алгоритм действий при выявлении пробела в знаниях:

1. **Конкретизация проблемы:** Четко сформулируйте, чего вы не знаете или не умеете. Не «я не могу сделать интерфейс», а «я не знаю, как в PyQt создать модальное окно с формой для ввода данных и обработать нажатие кнопки «Сохранить».
2. **Разработка плана освоения (Реабилитационный план):** Разбейте большую проблему на обучающие вопросы. Именно для этого служит «План освоения отсутствующих компетенций» (см. Приложения А, Б, В).
 - **Этап 1 (Поиск):** Сформулируйте конкретные вопросы по технологии (Как установить? Как создать? Как связать?).
 - **Этап 2 (Реализация):** Выполните практические шаги, опираясь на найденные ответы (Напишите код. Создайте окно. Подключите БД.).
 - **Этап 3 (Тестирование):** Проверьте, что освоенный навык работает корректно и интегрируется в основной проект.
3. **Целевой поиск информации:** Используйте методику из раздела 2.3. Ищите ответы на конкретные вопросы из вашего плана. Фиксируйте полезные источники (ссылки, названия видео).
4. **Практическое применение «здесь и сейчас»:** Не просто читайте, а сразу пробуйте. Создайте отдельный тестовый пилот-проект или новый файл в основном проекте, где будете отрабатывать новый навык.
5. **Интеграция в основной проект:** После успешной отработки на «песочнице», перенесите рабочий код или подход в ваш основной проект учебной практики.

6. Рефлексия и документирование: Зафиксируйте, какую проблему вы решили и каким образом. Это знание станет частью вашего профессионального опыта и может быть отражено в отчете.

Главное правило: Нельзя «застрять» с формулировкой «я этого не умею». Нужно перейти к действию: «для того чтобы это сделать, мне нужно последовательно изучить А, Б и В». Такой подход превращает учебную практику из непосильной задачи в управляемый процесс последовательного обучения и разработки.

5. Технологическая карта (план разработки) проекта

5.1. Назначение и структура технологической карты

Технологическая карта (ТК) — это динамичный инструмент управления индивидуальной или групповой проектной деятельностью, который визуализирует путь от постановки задачи до готового продукта. В отличие от календарного плана, ТК фокусируется на **технологических этапах, конкретных действиях, используемых ресурсах и результатах каждого шага**.

Назначение технологической карты:

1. **Структурирование процесса:** Разбивает крупный и сложный проект на последовательные, логически связанные и выполнимые этапы.
2. **Планирование и контроль:** Позволяет студенту и руководителю практики четко видеть текущий статус работы, объем выполненного и оставшиеся задачи.
3. **Фиксация компетенций:** Наглядно демонстрирует, какие именно навыки (поиск информации, проектирование, кодирование, тестирование) осваиваются и применяются на каждом этапе.
4. **Документирование процесса:** Служит основой для написания итогового отчета по практике, так как содержит хронологию и содержание всех работ.
5. **Рефлексия и самооценка:** Помогает студенту анализировать собственные трудности, успехи и эффективность выбранных методов работы.

Базовая структура технологической карты включает следующие столбцы:

№ этапа	Название этапа (блока работ)	Ключевые задачи / Действия	Используемые технологии, инструменты, ресурсы	Критерии выполнения / Результат этапа	Срок (план)	Срок (факт)	Статус (✓/ /△)	Примечания, возникшие проблемы
...	...	...	...	...	...	...	...	...

Пояснение к столбцам:

- **№ этапа:** Сквозная нумерация.
- **Название этапа:** Отражение фазы проекта (например, «Настройка среды», «Проектирование БД», «Реализация UI главного окна»).
- **Ключевые задачи:** Конкретные, проверяемые действия (например, «Установить Python 3.10», «Создать таблицу Recipes в SQLite», «Написать метод load_recipes()»).

- **Используемые ресурсы:** Ссылки на документацию, видеоуроки, статьи Stack Overflow, установленные библиотеки (pip install PyQt6).
- **Критерии выполнения:** Четкий индикатор завершения этапа (например, «Файл main.py запускается и отображает пустое окно», «Запрос SELECT * FROM Recipes возвращает тестовые данные», «Кнопка «Добавить» открывает новое модальное окно»).
- **Статус:** ✓ — Выполнено, — В работе, △ — Проблема/Приостановлено.

5.2. Пример заполнения технологической карты на основе сквозного проекта

Ниже приведен пример заполненной технологической карты для проекта «**Цифровая поваренная книга**» на начальных этапах. Карта построена по принципу трех ключевых фаз, соответствующих **Приложениям А, Б, В** (Поиск, Реализация, Тестирование).

№ этапа	Название этапа	Ключевые задачи / Действия	Используемые технологии, инструменты, ресурсы	Критерии выполнения / Результат этапа	Срок (план)	Срок (факт)	Статус	Примечания
А. ФАЗА 1: ПОИСК И ОСВОЕНИЕ (ПОДГОТОВКА)								
1.1	Настройка IDE и окружения	1. Скачать и установить VSCode. 2. Установить Python 3.10+. 3. Установить расширения Python, Pylance.	Официальные сайты VSCode, Python. Маркетплейс расширений VSCode.	VSCode запущен, в терминале команда python --version показывает нужную версию, расширения активны.	01.10	01.10	✓	—
1.2	Изучение GUI-фреймворка	1. Сравнить Tkinter и PyQt. 2. Принять решение в пользу PyQt6.	Статьи на Habr, официальная документация PyQt, видео	Выполнен выбор фреймворка. Написан и	02.10	03.10	✓	Выбрал PyQt из-за более современ

№ этапа	Название этапа	Ключевые задачи / Действия	Используемые технологии, инструменты, ресурсы	Критерии выполнения / Результат этапа	Срок (план)	Срок (факт)	Статус	Примечания
		3. Найти tutorial по созданию первого окна.	"PyQt6 Tutorial for Beginners".	запущен тестовый скрипт test_qt.py, который показывает пустое окно.				нового вида и богатых виджетов.
1.3	Проектирование структуры БД	1. Сформулировать вопросы по SQLite в Python. 2. Изучить синтаксис CREATE TABLE. 3. Нарисовать ER-диаграмму сущностей: Recipe, Ingredient, Recipe_Ingredients.	SQLite документация, модуль sqlite3 в Python, draw.io для диаграмм.	Готова ER-диаграмма в файле er_diagram.png. Написан скрипт create_db.py, создающий все таблицы.	03.10	04.10	✓	Возникла проблема со связью много-ко-многим, разобрался через JOIN.
Б. ФАЗА 2: РЕАЛИЗАЦИЯ ЯДРА ПРИЛОЖЕНИЯ								
2.1	Создание каркаса приложения	1. Создать структуру папок: views/, logic/, database/.	PyQt6 (QMainWindow, QListWidget, QVBoxLayout).	Приложение запускается. Отображается главное окно с	04.10	05.10	✓	Кнопка пока ничего не

№ этапа	Название этапа	Ключевые задачи / Действия	Используемые технологии, инструменты, ресурсы	Критерии выполнения / Результат этапа	Срок (план)	Срок (факт)	Статус	Примечания
	ния	2. Создать main.py с классом MainWindow. 3. Реализовать базовый UI: меню, кнопка "Добавить рецепт", QListWidget.		меню, пустым списком и рабочей кнопкой.				делает.
2.2	Реализация модуля работы с БД	1. Создать database/db_manager.py. 2. Реализовать класс DatabaseManager с методами connect(), add_recipe(), get_all_recipes(). 3. Настроить подключение к cookbook.db.	Модуль sqlite3, параметризованные запросы (?).	Класс Database Manager создан. Метод add_recipe("Тест", "...", 30) корректно добавляет запись в БД. Метод get_all_recipes() возвращает список.	05.10	06.10		Столкнулся с ошибкой "database is locked". Решил, убедившись, что соединение закрывается.
2.3	Связь UI и БД	1. Создать views/recipe	PyQt6 (QDialog, QLineEdit)	При нажатии кнопки	07.10	–		В процессе.

№ этапа	Название этапа	Ключевые задачи / Действия	Используемые технологии, инструменты, ресурсы	Критерии выполнения / Результат этапа	Срок (план)	Срок (факт)	Статус	Примечания
	(Добавление/Просмотр)	_form.py (окно добавления). 2. Привязать кнопку "Добавить рецепт" к открытию этого окна. 3. Реализовать функцию load_recipes() для заполнения списка.	dit), сигналы и слоты (.clicked.connect).	открывается форма ввода. После сохранения формы новый рецепт появляется в списке на главном окне.				Нужно передать данные из формы в db_manager.add_recipe().
В. ФАЗА 3: ТЕСТИРОВАНИЕ И СБОРКА								
3.1	Модульное тестирование БД	<i>Задачи будут детализованы после завершения фазы 2</i>	pytest, библиотека unittest.mock.	—	—	—		—
3.2	Сборка исполняемого файла	<i>Задачи будут детализованы после завершения фазы 2</i>	PyInstaller.	—	—	—		—

Как работать с технологической картой:

1. **Заполняйте карту по мере продвижения.** Она должна быть «живым» документом.
2. **Детализируйте задачи** настолько, чтобы одна задача соответствовала одному конкретному действию (от 30 минут до 2-3 часов работы).
3. **Обязательно фиксируйте ресурсы (столбец 4).** Это ваш персональный «чек-лист» полезных материалов.
4. **Честно отмечайте статусы и проблемы.** Столбец «Примечания» — важнейший инструмент рефлексии. Фиксируйте, как вы решили проблему.
5. **Сверяйтесь с картой ежедневно,** чтобы корректировать планы и не упускать из виду общую цель.

Технологическая карта является основным документом, сопровождающим **План освоения отсутствующих компетенций** (Приложения А-В) и наглядно показывает, как теоретический поиск и обучение преобразуются в практические шаги по созданию программного продукта.

6. Критерии оценки самостоятельной работы

Оценка самостоятельной работы студента (СРС) носит комплексный характер и направлена на проверку не только итогового результата, но и **процесса его достижения**, включая планирование, поиск информации, преодоление трудностей и рефлексия. Критерии разделены в зависимости от вида самостоятельной деятельности.

6.1. Критерии оценки подготовки к занятиям

Данные критерии применяются при оценке готовности студента к выполнению практической/лабораторной работы (проверка в начале занятия, оценка активности в процессе работы).

Критерий	Высокий уровень (5 баллов)	Достаточный уровень (4 балла)	Недостаточный уровень (3 балла)	Низкий уровень (2 балла)
Знание теоретической основы	Студент демонстрирует глубокое понимание теории, может объяснить ключевые концепции своими словами, привести примеры, ответить на уточняющие вопросы.	Студент знает основные определения и принципы, может пересказать материал, но испытывает затруднения при объяснении взаимосвязей или решении нестандартных задач.	Студент воспроизводит теорию фрагментарно, с ошибками, не может применить знания к конкретной ситуации в задании.	Теоретическая подготовка отсутствует. Студент не может ответить на базовые вопросы по теме.
Предварительный анализ задания	Студент детально изучил задание, выделил все ключевые задачи, подготовил четкий план действий, выявил потенциально сложные места и сформулировал вопросы.	Студент ознакомился с заданием, понимает общую цель, но план действий смутный или неполный. Вопросы носят общий характер.	Студент прочитал задание поверхностно, не может пояснить последовательность своих действий без подсказки преподавателя.	Задание не прочитано, студент не готов приступить к работе.
Готовность	Все необходимое ПО	Основное ПО	ПО установлено не	Рабочая среда не

Критерий	Высокий уровень (5 баллов)	Достаточный уровень (4 балла)	Недостаточный уровень (3 балла)	Низкий уровень (2 балла)
рабочей среды	установлено, настроено и проверено на работоспособность. Создана заготовка проекта/файловая структура.	установлено, но возможны незначительные недонастройки, не мешающие началу работы.	полностью или с критическими ошибками, требующими времени на исправление в ходе занятия.	подготовлена.
Активность и эффективность на занятии	Студент работает максимально самостоятельно, четко следует плану, грамотно использует конспекты и ресурсы, задает конкретные вопросы по делу.	Студент работает с периодическими консультациями, в целом справляется с заданием, но может допускать ошибки из-за невнимательности к деталям.	Студент нуждается в постоянной помощи, не может работать без пошаговых инструкций, часто отвлекается.	Студент пассивен, не предпринимает попыток выполнить задание самостоятельно.

Итоговая оценка выставляется как среднее арифметическое по всем критериям с учетом весомости (например, «Готовность среды» может быть обязательным допуском к работе).

6.2. Критерии оценки самостоятельной работы в рамках проекта

Данные критерии применяются при защите этапов учебной практики, проверке индивидуальных заданий и «Планов освоения компетенций».

Критерий	Высокий уровень (5 баллов)	Достаточный уровень (4 балла)	Недостаточный уровень (3 балла)	Низкий уровень (2 балла)
Качество планирования и	Работа четко спланирована с	План существует, но может быть	План формальный, задачи сформулированы размыто	Планирование не велось. Работа

Критерий	Высокий уровень (5 баллов)	Достаточный уровень (4 балла)	Недостаточный уровень (3 балла)	Низкий уровень (2 балла)
декомпозиции	использованием инструментов (ТК, бэклог). Задачи декомпозированы на мелкие, измеримые шаги. Сроки реалистичны.	недостаточно детализирован или содержать нереалистичные оценки времени. Логика этапов в целом сохранена.	(«сделать интерфейс»), декомпозиция отсутствует.	выполнялась хаотично.
Самостоятельность поиска и применения информации	Студент эффективно использует различные источники (документация, Stack Overflow, книги). Найденная информация критически осмыслена, адаптирована и корректно встроена в проект.	Студент находит информацию, в основном следует примерам, но может некритично копировать код без полного понимания. Адаптация под задачу требует усилий.	Поиск неэффективен (неверные запросы). Студент использует устаревшие или неподходящие источники. Код скопирован без понимания, что приводит к ошибкам.	Попытки самостоятельного поиска не наблюдаются. Студент ожидает готового решения.
Техническая реализация и качество кода	Код полностью функционален, соответствует ТЗ. Архитектура логична, код чистый, хорошо документирован (комментарии, docstrings), соответствует стандартам (PEP 8).	Код работает, но может содержать неоптимальные решения, избыточность или minor-баги. Комментарии и форматирование присутствуют, но не	Код работает с ошибками или неполностью. Архитектура запутана, код плохо читаем, отсутствует документация. Много закомментированного «мусора».	Код нерабочий или отсутствует.

Критерий	Высокий уровень (5 баллов)	Достаточный уровень (4 балла)	Недостаточный уровень (3 балла)	Низкий уровень (2 балла)
		езде.		
Тестирование и отладка	Реализованы методы самотестирования (модульные тесты, ручные сценарии). Выявленные ошибки грамотно исправлены. Процесс отладки задокументирован.	Функционал проверен вручную по основным сценариям. Ошибки исправляются после их обнаружения, но процесс может быть несистемным.	Тестирование поверхностное. Критические ошибки могут быть пропущены. Процесс отладки сводится к случайным изменениям кода.	Тестирование не проводилось.
Рефлексия и работа с ошибками	Студент анализирует возникающие проблемы, фиксирует их и пути решения в технологической карте или отчете. Делает выводы, чтобы избежать подобных ошибок в будущем.	Студент замечает проблемы, но анализ их причин поверхностен. Решения часто находятся с внешней помощью.	Проблемы констатируются, но их анализ и поиск решений отсутствуют («не работает, и не знаю почему»).	Отсутствует реакция на возникающие проблемы.
Своевременность и оформление результата	Работа сдана в установленный срок. Результат (код, отчет, ответы) оформлен аккуратно, в соответствии с требованиями. Файлы	Работа сдана с незначительной задержкой или есть мелкие замечания по оформлению.	Работа сдана с серьезным опозданием или оформлена неряшливо, что затрудняет проверку.	Работа не сдана.

Критерий	Высокий уровень (5 баллов)	Достаточный уровень (4 балла)	Недостаточный уровень (3 балла)	Низкий уровень (2 балла)
	корректно названы.			

Итоговая оценка за проектную работу является интегральной и выставляется с учетом всех критериев. Особый вес придается критериям «Самостоятельность поиска» и «Техническая реализация», так как они наиболее точно отражают сформированность профессиональных компетенций.

Примечание: Оценка «2» по любому из **обязательных критериев** (например, «Техническая реализация» для проекта или «Знание теоретической основы» для допуска к работе) может повлечь недопуск к защите этапа или необходимость переработки задания.

7. Заключение

Настоящие методические рекомендации представляют собой системный подход к организации наиболее важного компонента профессионального становления IT-специалиста — **самостоятельной учебной и проектной деятельности**. Их практическое применение позволяет трансформировать стихийный процесс самообучения в управляемый, результативный и рефлексивный, максимально приближая учебную среду к реальным условиям IT-индустрии, где инициатива, умение быстро осваивать новое и работать с документацией являются ключевыми компетенциями.

7.1. Рекомендации по самоорганизации и тайм-менеджменту

Успех в самостоятельной работе на 90% зависит от личной организации. Следующие практические советы помогут повысить ее эффективность:

1. **Принцип «Съешь лягушку»:** Начинайте учебный день или блок самостоятельной работы с самой сложной и неприятной задачи. Это снизит уровень стресса и повысит продуктивность в оставшееся время.

2. **Метод «Помодоро»:** Работайте интервалами по 25-30 минут (один «помидор») с последующим коротким перерывом (5 минут). После четырех интервалов — длинный перерыв (15-30 минут). Это помогает сохранять концентрацию и бороться с выгоранием.

3. **«Правило двух минут»:** Если задача требует меньше двух минут на выполнение (например, ответить на простое письмо, создать папку для проекта), сделайте ее немедленно. Это разгружает оперативную память и список дел.

4. **Используйте цифровые инструменты:** Заведите единый календарь (Google Calendar, Outlook) для всех дедлайнов. Используйте приложения для управления задачами (Todoist, Microsoft To Do) или простой текстовый файл для ведения ежедневного и еженедельного плана. Фиксируйте не только цели, но и потраченное время.

5. **Создайте «ритуал» начала работы:** Определенный порядок действий (налить чай, закрыть лишние вкладки, открыть план на день) помогает мозгу быстрее переключиться в рабочий режим.

6. **Планируйте отдых и физическую активность:** Программирование — интенсивная умственная работа. Включите в расписание перерывы, прогулки и спорт. Это не потеря времени, а инвестиция в качество работы и здоровье.

7.2. Часто встречающиеся трудности и пути их преодоления

В процессе самостоятельной работы студенты регулярно сталкиваются с типичными проблемами. Знание заранее, как на них реагировать, снижает панику и помогает действовать рационально.

Трудность	Возможная причина	Пути преодоления
Прокрастинация, не могу начать.	Страх сложной задачи, неясность с чего начать, перфекционизм.	1. Применить декомпозицию — разбить задачу на мельчайший первый шаг (например, «не делать проект», а «открыть IDE и создать файл main.py»). 2. Использовать таймер на 5 минут («поработаю всего 5 минут»). Часто этого хватает, чтобы включиться в процесс.
«Застрял» на ошибке, не могу найти решение.	Неверный поисковый запрос, недостаточное понимание	1. Максимально конкретизировать ошибку: скопировать полное сообщение из консоли, указать ОС, версии библиотек. 2. Искать на английском языке по тексту ошибки в кавычках.

Трудность	Возможная причина	Пути преодоления
	проблемы, попытка решить сложную задачу целиком.	3. Откатиться на последнюю рабочую версию (Git!) и двигаться небольшими шагами, проверяя каждый. 4. Объяснить проблему вслух (резиновая уточка) или письменно — часто решение приходит в процессе формулировки.
Потеря мотивации, ощущение рутины.	Отсутствие видимого прогресса, большие объемы монотонной работы.	1. Визуализировать прогресс: отмечать выполненные задачи в технологической карте, использовать прогресс-бары. 2. Чередовать виды деятельности (после писания кода — заняться дизайном или тестированием). 3. Найти «учебного напарника» для взаимной поддержки и обсуждения проблем. 4. Вспомнить и записать долгосрочную цель («Я учусь, чтобы стать разработчиком и создавать полезные продукты»).
Перегрузка информацией, «каша в голове».	Одновременное изучение нескольких сложных тем, отсутствие систематизации.	1. Остановиться и структурировать знания: составить конспект, ментальную карту или блок-схему. 2. Сфокусироваться на одной технологии/задаче до достижения понимания. 3. Применять новый навык на практике немедленно, даже в учебном мини-проекте. Практика закрепляет теорию.
Не укладываюсь в сроки.	Неверная оценка сложности, неучтенные проблемы, плохое планирование.	1. При планировании всегда закладывать временной буфер (+30-50%) на непредвиденные обстоятельства. 2. Регулярно (раз в день/неделю) сверяться с планом и корректировать его. 3. В случае срыва сроков — немедленно проинформировать руководителя (преподавателя) и совместно скорректировать план, а не скрывать проблему.

Следование изложенным в рекомендациях принципам, осознанное применение инструментов планирования (включая **Технологическую карту** и **Планы освоения компетенций**) и развитая навыковая рефлексия позволят не только успешно выполнить учебную программу, но и сформировать **профессиональный mindset**, необходимый для построения успешной карьеры в динамичной сфере информационных технологий. Помните: ценность представляет не только написанный код, но и **доказанная способность самостоятельно учиться, планировать и доводить проект до результата**.

Технологическая карта этап 3

Введение:

Цель плана — последовательно освоить недостающие навыки поиска информации, проектирования и разработки, необходимые для выполнения задания УП.

План состоит из трех этапов:

1. **Этап 1: Поиск и систематизация информации.** Формирование навыков самостоятельного поиска решений с использованием интернет-источников.
2. **Этап 2: Пошаговая реализация.** Выполнение практических действий на основе найденной информации.
3. **Этап 3: Тестирование и отладка.** Проверка работоспособности приложения и исправление ошибок.

Ниже представлен **Этап 1** плана. Вы должны последовательно найти ответы на каждый из 50 поставленных вопросов, используя поисковые системы (Google, Yandex) и специализированные ресурсы (Stack Overflow, официальная документация).

Этап 1: Поиск необходимой информации

А. Настройка рабочей среды и основы (1-8)

1. Как скачать и установить VSCodium (или Visual Studio Code) на Windows?
2. Как установить и настроить Python для работы в VSCodium?
3. Какие плагины (extensions) в VSCodium необходимы для комфортной разработки на Python (например, для подсветки синтаксиса, линтинга)?
4. Что такое "виртуальное окружение" (venv) в Python и как его создать и активировать в терминале VSCodium?
5. Как в VSCodium открыть папку проекта и создать в ней первые файлы (например, main.py)?
6. Какую библиотеку (фреймворк) Python лучше использовать для создания десктопного GUI-приложения: PyQt или Tkinter? (Сравнение и выбор).
7. Как установить выбранную библиотеку (PyQt или Tkinter) в свое виртуальное окружение с помощью команды pip install?
8. Как создать самое простое "пустое окно" приложения с помощью PyQt (или Tkinter)?

Б. Проектирование UI и подключение к БД (9-19)

9. Как в PyQt (или Tkinter) программно создать главное окно приложения с меню (File, Edit) и кнопкой "Добавить рецепт"?
10. Как программно создать макет окна (разметку) с помощью виджетов-компоновщиков (например, QVBoxLayout, QGridLayout в PyQt или pack, grid в Tkinter)?
11. Как программно добавить виджеты (например, QPushButton, QLabel, QLineEdit) в созданный макет окна?
12. Как создать иерархию (структуру) проекта на Python, разделяя файлы UI, логики и работы с базой данных (на основе ТЗ и ER-диаграммы)?
13. Какой встроенный модуль Python используется для работы с базами данных SQLite?
14. Нужно ли устанавливать модуль sqlite3 через pip, или он является частью стандартной библиотеки Python?
15. Как установить соединение с файлом базы данных SQLite (например, my_database.db) из Python?
16. Как выполнить простой SQL-запрос (например, SELECT 1) из Python, используя модуль sqlite3, чтобы проверить соединение с базой данных?

17. Что такое ORM (Object-Relational Mapper) и в чем ее преимущество перед "сырыми" SQL-запросами?
18. Что такое SQLAlchemy и как ее использовать для подключения к файлу базы данных SQLite?
19. Как определить таблицу базы данных (например, Recipes) в виде класса Python ("модели") с помощью SQLAlchemy ORM?

В. Реализация CRUD-операций (Создание, Чтение) (20-29)

20. Как в PyQt (или Tkinter) программно создать новое модальное окно (например, для добавления рецепта), которое открывается по нажатию кнопки?
21. Как в этом новом окне программно разместить поля ввода (QLineEdit / Entry) для названия рецепта, описания и времени приготовления?
22. Как получить текст из полей ввода (QLineEdit) в Python-переменные по нажатию кнопки "Сохранить"?
23. Как с помощью модуля sqlite3 выполнить SQL-запрос INSERT с использованием параметров (placeholder ?) для безопасного добавления нового рецепта?
24. Как на главном экране приложения программно разместить список (QListWidget / Listbox) для отображения всех рецептов?
25. Как с помощью модуля sqlite3 выполнить SQL-запрос SELECT, чтобы получить названия всех рецептов из таблицы Recipes?
26. Как заполнить QListWidget (или Listbox) данными, полученными из базы данных?
27. Как отследить, какой элемент списка (QListWidget) выбрал пользователь (событие "клик" или "выделение")?
28. Как по клику на рецепт в списке открыть новое окно "Детали рецепта" и передать в него ID этого рецепта?
29. Как в окне "Детали рецепта" выполнить запрос SELECT * FROM Recipes WHERE id = ? с помощью sqlite3, используя параметр для ID?

Г. Реализация CRUD (Обновление, Удаление) и связей (30-38)

30. Как в окне "Детали рецепта" отобразить список ингредиентов, связанных с этим рецептом (запрос SELECT с JOIN таблиц Recipe_Ingredients и Ingredients)?
31. Как программно добавить кнопку "Редактировать" в окне "Детали рецепта"?
32. Как по нажатию кнопки "Редактировать" сделать поля ввода доступными для изменения и загрузить в них текущие данные?
33. Как по нажатию кнопки "Сохранить изменения" выполнить SQL-запрос UPDATE (с параметрами) в sqlite3, чтобы обновить информацию о рецепте?
34. Как программно добавить кнопку "Удалить" в окне "Детали рецепта"?
35. Как по нажатию кнопки "Удалить" выполнить SQL-запрос DELETE (с параметрами) в sqlite3, чтобы удалить текущий рецепт?
36. Как показать пользователю диалоговое окно с подтверждением ("Вы уверены, что хотите удалить?") перед выполнением удаления?
37. Как реализовать поиск рецептов по названию (SQL-запрос SELECT с использованием WHERE ... LIKE ... в sqlite3)?
38. Как обновить список рецептов на главном экране после добавления, редактирования или удаления рецепта?

Д. Реализация "Списка покупок" (Основная бизнес-логика) (39-45)

39. Как в окне "Детали рецепта" программно рядом с каждым ингредиентом добавить CheckBox (флажок)?
40. Как программно добавить кнопку "Добавить ингредиенты в список покупок"?

- 41. Как по нажатию этой кнопки получить все ингредиенты, у которых CheckBox отмечен (установлена "галочка")?
- 42. Как для каждого отмеченного ингредиента выполнить SQL-запрос INSERT в таблицу Shopping_List (используя sqlite3 с параметрами)?
- 43. Как создать отдельное программное окно "Список покупок"?
- 44. Как в окне "Список покупок" отобразить все записи из таблицы Shopping_List (запрос SELECT через sqlite3)?
- 45. Как добавить кнопку "Очистить список покупок" (запрос DELETE FROM Shopping_List через sqlite3)?

Е. Тестирование, отладка и сборка (46-50)

- 46. Как использовать "отладчик" (debugger) в VSCodium для пошагового выполнения Python-кода и поиска ошибок?
- 47. Что такое try...except в Python и как его использовать для обработки ошибок (например, "файл базы данных не найден" или "рецепт не найден")?
- 48. Как показывать пользователю сообщения об ошибках с помощью MessageBox (QMessageBox в PyQt или messagebox в Tkinter)?
- 49. Что такое PyInstaller и как его использовать для преобразования Python-проекта (файлов .py) в один исполняемый .exe файл?
- 50. Какие проблемы могут возникнуть при сборке .exe файла PyInstaller (например, "не найден файл базы данных .db" или "не найдена иконка") и как их решить?

Технологическая карта этап 3

Введение:

На Этапе 1 вы выполнили поиск необходимой информации по 50 ключевым вопросам.

На данном этапе вы должны **практически реализовать** все шаги по созданию приложения, используя найденную на предыдущем этапе информацию. Каждое действие в этом списке является обязательным и должно быть выполнено **строго последовательно**. Ваша задача — реализовать функционал в соответствии с ранее разработанным ТЗ, макету программы, уделяя внимание чистоте кода, его документированию (docstrings) и разделению логики. В случае возникновения проблем с одним из пунктов - необходимо найти дополнительную информацию по вопросу, выполнить данный пункт самостоятельно.

Этап 2: Пошаговая реализация программы

А. Настройка рабочей среды и основы (1-8)

1. Установить VSCode (Visual Studio Code) на свой рабочий компьютер.
2. Установить актуальную версию Python (3.9+) и убедиться, что он интегрирован с VSCode.
3. Установить в VSCode плагины: "Python" (от Microsoft), "Pylance" и опционально "Black Formatter" (для автоматического форматирования).
4. Создать корневую папку проекта (например, "Cookbook"). Открыть эту папку в VSCode.
5. В терминале VSCode создать виртуальное окружение (python -m venv venv) и активировать его.
6. Создать в корне проекта главный исполняемый файл main.py.
7. Принять решение об использовании GUI-библиотеки (рекомендуется PyQt за счет гибкости и набора виджетов).
8. Установить выбранную библиотеку в виртуальное окружение (например, pip install PyQt6).
9. В файле main.py написать код, который создает и отображает простое пустое окно приложения (проверка работоспособности библиотеки).

Б. Проектирование UI и подключение к БД (9-19)

10. В main.py модифицировать код, программно добавив в окно верхнее меню (QMenuBar) и одну кнопку (QPushButton) "Добавить рецепт".
11. Создать в main.py или отдельном классе MainWindow метод для инициализации UI (например, initUI).
12. Программно создать макет (разметку) окна, используя QVBoxLayout и QGridLayout для размещения элементов.
13. Программно разместить в главном окне QListWidget (для списка рецептов) и кнопки.
14. Написать функцию, которая будет вызываться по нажатию кнопки "Добавить рецепт", используя механизм сигналов и слотов (.clicked.connect(...)).
15. Создать в проекте иерархию папок для разделения логики:
 - o views/ (для Python-классов, описывающих окна UI, например MainWindow, RecipeForm)
 - o logic/ (для хранения классов бизнес-логики)
 - o database/ (для модуля работы с БД)
16. Использовать встроенный модуль sqlite3 для работы с базой данных SQLite.
17. В папке database/ создать файл db_config.py и записать в него имя файла базы данных (например, DB_NAME = 'cookbook.db').

18. В папке database/ создать файл db_manager.py. Написать в нем функцию test_connection(), которая импортирует DB_NAME и выполняет тестовый запрос (SELECT 1) для проверки создания/подключения к файлу БД.

19. В db_manager.py создать класс DatabaseManager, который будет управлять всеми запросами к БД.

В. Реализация CRUD-операций (Создание, Чтение) (20-29)

20. Создать новый Python-файл/класс (например, RecipeFormWindow в папке views/) для модального окна (наследуя от QDialog).

21. Программно разместить на этой форме (RecipeFormWindow) поля: QLineEdit (Название), QTextEdit (Описание), QSpinBox (Время приготовления) и кнопки "Сохранить", "Отмена".

22. Реализовать в классе RecipeFormWindow логику получения данных из полей QLineEdit, QTextEdit и т.д. по нажатию кнопки "Сохранить".

23. В DatabaseManager реализовать метод add_recipe(name, description, time) с параметризованным запросом INSERT INTO Recipes ... VALUES (?, ?, ?).

24. Убедиться, что в главном окне программно создан QListWidget (список рецептов).

25. В DatabaseManager реализовать метод get_all_recipes(), выполняющий SELECT id, name FROM Recipes.

26. В главном окне реализовать функцию load_recipes(), которая вызывает get_all_recipes() и заполняет QListWidget полученными данными.

27. Подключить к QListWidget обработчик события itemClicked.

28. Создать новый Python-файл/класс (например, RecipeDetailWindow) для окна "Детали Рецепта".

29. В DatabaseManager реализовать метод get_recipe_details(recipe_id) с параметризованным запросом SELECT * FROM Recipes WHERE id = ?.

Г. Реализация CRUD (Обновление, Удаление) и связей (30-38)

30. В DatabaseManager реализовать метод get_ingredients(recipe_id) (используя JOIN таблиц Recipe_Ingredients и Ingredients). В окне RecipeDetailWindow отобразить этот список.

31. Программно добавить в RecipeDetailWindow кнопку "Редактировать".

32. Реализовать логику, которая по нажатию "Редактировать" делает поля (QLineEdit и др.) доступными для изменения и меняет текст кнопки на "Сохранить".

33. В DatabaseManager реализовать метод update_recipe(...) с параметризованным запросом UPDATE Recipes SET ... WHERE id = ?.

34. Программно добавить в RecipeDetailWindow кнопку "Удалить".

35. В DatabaseManager реализовать метод delete_recipe(recipe_id) с параметризованным запросом DELETE FROM Recipes WHERE id = ?.

36. При нажатии на кнопку "Удалить" сначала показывать диалоговое окно QMessageBox.question с вопросом "Вы уверены?".

37. В главном окне программно добавить QLineEdit для поиска. Реализовать в DatabaseManager метод search_recipes(term) (с WHERE name LIKE ?).

38. Реализовать обновление (load_recipes()) главного списка после добавления, редактирования или удаления рецепта.

Д. Реализация "Списка покупок" (Основная бизнес-логика) (39-45)

39. Модифицировать список ингредиентов в RecipeDetailWindow: программно добавить QCheckBox напротив каждого ингредиента.

40. Программно добавить в RecipeDetailWindow кнопку "Добавить в список покупок".

41. Написать функцию, которая по нажатию этой кнопки проходит по списку ингредиентов и собирает ID тех, у которых QCheckBox отмечен.
42. В DatabaseManager реализовать метод `add_to_shopping_list(ingredient_id, recipe_id)`, выполняющий `INSERT INTO Shopping_List ....`
43. Создать новый Python-файл/класс (например, `ShoppingListWindow`) для окна "Список покупок".
44. В DatabaseManager реализовать метод `get_shopping_list()` (с JOIN для получения названий). При открытии `ShoppingListWindow` вызывать этот метод и заполнять список покупок.
45. Программно добавить в `ShoppingListWindow` кнопку "Очистить список". В DatabaseManager реализовать метод `clear_shopping_list()` (`DELETE FROM Shopping_List`).

Е. Тестирование, отладка и сборка (46-50)

46. Установить точки останова (Breakpoints) в VSCodium на ключевых функциях (например, `add_recipe`) и запустить приложение в режиме отладки (F5) для пошаговой проверки.
47. Обернуть все вызовы к базе данных (в DatabaseManager) и критические операции (например, при подключении к БД) в блоки `try...except`.
48. В блоках `except` реализовать показ окон `QMessageBox.critical()` или `QMessageBox.warning()`, информирующих пользователя об ошибке (например, "Не удалось открыть файл базы данных" или "Ошибка SQL-запроса").
49. Установить PyInstaller (`pip install pyinstaller`). Выполнить в терминале команду для сборки проекта в один .exe файл.
50. Если сборка .exe не удалась, устранить ошибки. (Частая проблема: PyInstaller "не видит" файл базы данных .db. Решение: использовать флаг `--add-data` для включения файла БД в сборку).

Технологическая карта этап 3

Введение:

На Этапе 1 вы нашли необходимую информацию, а на Этапе 2 — реализовали функционал программы.

На данном этапе вы должны выступить в роли "Инженера по обеспечению качества (QA)". Ваша задача — **тщательно протестировать** приложение на всех уровнях (модульном, интеграционном, системном), выявить и задокументировать ошибки. Выполнение этого этапа критически важно для подтверждения работоспособности и соответствия вашего приложения исходным требованиям (ТЗ).

Этап 3: Пошаговое тестирование и обеспечение качества

А. Подготовка к тестированию (1-5)

1. Убедиться, что в виртуальном окружении установлена библиотека pytest (pip install pytest) для проведения модульного тестирования.
2. Создать в корне проекта папку tests/.
3. Изучить и подготовить к использованию отчет по покрытию кода: установить pip install coverage.
4. Подготовить "чистую" тестовую базу данных (убедиться, что файл .db пуст или заполнен только стандартными тестовыми данными).
5. Проверить, что в VSCode настроен отладчик (Debugger) для пошаговой проверки кода.

Б. Модульное тестирование (Unit Testing) (6-14)

(Цель: проверить корректность работы отдельных функций, в первую очередь, класса DatabaseManager из Этапа 2).

6. В папке tests/ создать файл test_db_manager.py.
7. Написать модульный тест (pytest) для функции test_connection() (Этап 2, Шаг 18), который проверяет успешность подключения к файлу БД SQLite.
8. Написать модульный тест для функции add_recipe() (Этап 2, Шаг 23), используя "mock" (имитацию) базы данных, чтобы проверить, что функция генерирует правильный SQL-запрос INSERT.
9. Написать модульный тест для функции get_all_recipes() (Этап 2, Шаг 25), проверяющий корректность SQL-запроса SELECT.
10. Написать модульный тест для функции get_recipe_details() (Этап 2, Шаг 29), проверяющий SQL-запрос SELECT ... WHERE id = ?.
11. Написать модульный тест для функции update_recipe() (Этап 2, Шаг 33), проверяющий SQL-запрос UPDATE.
12. Написать модульный тест для функции delete_recipe() (Этап 2, Шаг 35), проверяющий SQL-запрос DELETE.
13. Написать модульный тест для функции search_recipes() (Этап 2, Шаг 37), проверяющий SQL-запрос SELECT ... LIKE
14. Запустить все тесты (pytest) и убедиться, что они проходят.

В. Интеграционное тестирование (Integration Testing) (15-25)

(Цель: проверить взаимодействие UI, бизнес-логики и реальной базы данных) .

15. Запустить приложение.
16. (Create) Через UI-форму (Этап 2, Шаг 20) добавить новый рецепт: "Тестовый Рецепт 1".
17. (Verify) Открыть файл .db с помощью (например) DB Browser for SQLite и вручную проверить, что "Тестовый Рецепт 1" корректно сохранился в таблице Recipes.

18. (Read) Перезапустить приложение. Убедиться, что "Тестовый Рецепт 1" отображается в главном списке QListWidget (Этап 2, Шаг 26).
19. (Update) Открыть детали "Тестовый Рецепт 1", нажать "Редактировать" (Этап 2, Шаг 32) и изменить название на "Обновленный Рецепт 1".
20. (Verify) Проверить в DB Browser for SQLite, что название в таблице Recipes изменилось.
21. (Integration Logic) Добавить 3 ингредиента к "Обновленному Рецепту 1".
22. (Integration Logic) Отметить 2 из 3 ингредиентов и нажать "Добавить в список покупок" (Этап 2, Шаг 40).
23. (Verify) Проверить в DB Browser for SQLite, что в таблице Shopping_List появилось ровно 2 новые записи (Этап 2, Шаг 42).
24. (Delete) Через UI (Этап 2, Шаг 34) удалить "Обновленный Рецепт 1".
25. (Verify) Проверить в DB Browser for SQLite, что запись о рецепте удалена из таблицы Recipes.

Г. Системное тестирование (User Scenarios) (26-37)

(Цель: проверить приложение целиком, пройдя по основным пользовательским сценариям от начала до конца) .

26. *Сценарий 1: "Новый пользователь"*. Запустить приложение.
27. Убедиться, что главный список рецептов пуст (если БД чистая).
28. Добавить рецепт "Борщ" (описание, 60 минут).
29. Добавить к "Борщу" ингредиенты: "Свекла", "Картофель", "Капуста", "Мясо".
30. Закрыть приложение. Снова открыть. Убедиться, что "Борщ" остался в списке.
31. Открыть детали "Борща". Отметить "Свекла" и "Мясо". Нажать "Добавить в список покупок".
32. Открыть окно "Список покупок" (Этап 2, Шаг 43). Убедиться, что "Свекла" и "Мясо" в списке.
33. В окне "Список покупок" нажать "Очистить список" (Этап 2, Шаг 45). Убедиться, что список очистился.
34. *Сценарий 2: "Поиск и редактирование"*.
35. Добавить второй рецепт: "Омлет".
36. В главном окне в строке поиска (Этап 2, Шаг 37) ввести "Бор". Убедиться, что в списке остался только "Борщ".
37. Стереть поиск. Отредактировать "Омлет", переименовав его в "Омлет с сыром". Убедиться, что новое имя отобразилось в общем списке.

Д. Тестирование устойчивости и безопасности (38-45)

(Цель: проверить граничные условия и базовую безопасность).

38. (Exception) Найти файл базы данных .db и временно переименовать его (например, cookbook.db_backup).
39. Запустить приложение. Убедиться, что оно не "вылетает", а показывает пользователю окно с ошибкой "Не удалось открыть файл базы данных" (проверка try...except из Этапа 2, Шаг 48).
40. (Boundary) В форме добавления рецепта (Этап 2, Шаг 21) нажать "Сохранить", оставив поле "Название" пустым.
41. Убедиться, что приложение выдает ошибку "Название не может быть пустым" и не сохраняет рецепт.
42. (Boundary) В поле "Время приготовления" ввести отрицательное число (-10). Убедиться, что приложение не сохраняет значение или выдает ошибку.
43. (Security) Вернуть файлу базы данных .db оригинальное имя.

44. (SQL Injection Test) В поле поиска (Этап 2, Шаг 37) ввести SQL-инъекцию: ' OR 1=1.
45. Убедиться, что приложение *не* вывело все рецепты (если оно пустое, значит, параметризованные запросы [Этап 2, Шаг 23] работают корректно).

Е. Тестирование развертывания (Deployment Testing) (46-50)

(Цель: проверить работоспособность собранного файла .exe).

46. Запустить сборку проекта с помощью PyInstaller (Этап 2, Шаг 49).
47. Найти в папке dist/ готовый .exe файл.
48. Скопировать этот .exe файл (или папку со сборкой) на "чистый" компьютер или виртуальную машину, где не установлен Python и VSCodium.
49. Запустить .exe файл. Убедиться, что приложение запускается без ошибок.
50. Провести "дымовое тестирование" на чистой машине: проверить 2-3 ключевые функции (например, добавить 1 рецепт и добавить 1 ингредиент в список покупок), чтобы убедиться, что .exe файл работает автономно.