Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Архитектура вычислительных систем

К защите допустить:

И.О. заведующего кафедрой
информатики
_____ С.И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ГЕНЕРАТОР ОТЧЁТОВ (НАКЛАДНЫХ, ЗАЯВЛЕНИЙ И Т.Д.), НАПРИМЕР, ИСПОЛЬЗУЮЩИЙ ЛАТЕХ.

БГУИР КП 1-40 04 01 014 ПЗ

Студент Н.В. Легоньков
Руководитель Н.Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире эффективное управление данными и создание профессионально оформленных документов являются неотъемлемой частью работы многих организаций и индивидуальных пользователей. С постоянным ростом объемов информации возникает необходимость в автоматизации процесса создания отчётов, накладных, заявлений и других документов. В этом контексте широкое распространение получают инструменты, позволяющие генерировать документы автоматически с помощью программного обеспечения.

Одним из наиболее мощных инструментов для создания профессиональных документов является LaTeX — система верстки текстов, позволяющая автоматизировать процесс оформления и создания сложных документов. В данном курсовом проекте мы рассмотрим использование LaTeX в качестве генератора отчётов, накладных, заявлений и других документов. Будет изучено как создавать шаблоны для различных типов документов, так и автоматизировать процесс их заполнения данными из различных источников.

Цель данного проекта — рассмотреть принципы работы генератора отчётов на базе LaTeX, его преимущества и ограничения, а также разработать и протестировать конкретный пример использования данного подхода для создания документов определенного типа. В ходе работы будут рассмотрены основные концепции LaTeX, методы создания шаблонов, а также интеграция с внешними источниками данных для автоматической генерации документов.

В рамках данного курсового проекта будет разработана программа на языке программирования Python, предназначенная для генерации файлов из предварительно созданных шаблонов в LaTeX. Python был выбран в качестве основного инструмента разработки благодаря своей гибкости, мощным библиотекам для обработки данных и легкости интеграции с различными источниками данных.

С использованием Python будет реализован механизм, позволяющий заполнять шаблоны LaTeX данными из внешних источников, а также другие источники. Это позволит автоматизировать процесс создания документов, сократить время, затрачиваемое на их подготовку, и уменьшить вероятность ошибок вручную заполняемых документов.

Такой подход объединяет мощь LaTeX в создании качественной верстки и удобство работы с данными в Python. Предполагается, что данная комбинация инструментов позволит создать эффективный генератор отчётов,

накладных, заявлений и других документов, который сможет удовлетворить потребности как отдельных пользователей, так и организаций различного масштаба.

Предполагается, что данное исследование и практические разработки окажутся полезными как для широкого круга пользователей LaTeX, так и для специалистов, сталкивающихся с необходимостью автоматизации процесса создания документации в их профессиональной деятельности.

1 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1 Основные понятия

В традиционном понимании платформа – это комплекс аппаратных и программных средств, на котором функционирует программное обеспечение пользователя ЭВМ. Основа аппаратной платформы (*hardware*-платформы) – процессор. Тип процессора определяет архитектуру аппаратных средств – аппаратную платформу, т. е. тип и характеристики компьютера.

Понятия «программная платформа» (*software*-платформа), или «программное обеспечение» вошли в жизнь с развитием компьютерной индустрии. Без программного обеспечения компьютер – всего лишь электронное устройство, которое не управляется, и потому не может приносить пользы. В зависимости от функций, выполняемых программным обеспечением, его можно разделить на две большие группы: системное и прикладное программное обеспечение.

Системное программное обеспечение – это «программная оболочка» аппаратных средств, предназначенная для отделения остальных программ от непосредственного взаимодействия с оборудованием и организации процесса обработки информации в компьютере. К системному программному обеспечению относятся такие типы программ, как операционные системы, различные сервисные средства, функционально дополняющие возможности операционных систем, инструментальные средства (системы управления базами данных, программирования, оболочки экспертных систем).

Прикладное программное обеспечение предназначено для решения определенных задач пользователя.

Операционная система (ОС) – это комплекс взаимосвязанных системных программ, которые загружаются при включении компьютера и постоянно находятся в памяти компьютера. Они производят диалог с пользователем, осуществляют управление компьютером, его ресурсами (оперативной памятью, местом на дисках и т.д.), запускают другие (прикладные) программы на выполнение. Операционная система обеспечивает пользователю и прикладным программам удобный способ общения (интерфейс) с устройствами компьютера.

Основная причина необходимости операционной системы состоит в том, что элементарные операции для работы с устройствами компьютера и управления ресурсами компьютера – это операции очень низкого уровня, поэтому действия, которые необходимы пользователю и прикладным

программам, состоят из нескольких сотен или тысяч таких элементарных операций.

Операционная система скрывает от пользователя все эти сложные и ненужные подробности и предоставляет ему удобный интерфейс для работы. Она выполняет также различные вспомогательные действия, например, копирование или печать файлов. Операционная система осуществляет загрузку в оперативную память всех программ, передает им управление в начале их работы, выполняет различные действия по запросу выполняемых программ и освобождает занимаемую программами оперативную память при их завершении.

В функции операционной системы входит:

1 Осуществление диалога с пользователем.

2 Ввод-вывод и управление данными.

3 Планирование и организация процесса обработки программ.

4 Распределение ресурсов (оперативной памяти и кэша, процессора, внешних устройств).

5 Запуск программ на выполнение.

6 Всевозможные вспомогательные операции обслуживания.

7 Передача информации между различными внутренними устройствами.

8 Программная поддержка работы периферийных устройств (дисплея, клавиатуры, дисковых накопителей, принтера и др.) [1].

1.2 Структура и архитектура платформы

В качестве языка программирования был выбран *Python*.

Python – это высокоуровневый язык программирования общего назначения, который используется в том числе и для разработки веб-приложений. Язык ориентирован на повышение производительности разработчика и читаемости кода.

Python поддерживает несколько парадигм программирования: структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. В языке присутствет динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных. Программный код на *Python* организовывается в функции и классы, которые могут объединяться в модули, а они в свою очередь могут быть объединены в пакеты. *Python* обычно

используется как интерпретируемый, но может быть скомпилирован в байт-код *Java* и в *MSIL* (в рамках платформы *.NET*). *Python* был выбран, потому что в сферах анализа данных и машинного обучения *Python* сейчас, несомненно, вне конкуренции [2].

В качестве среды разработки была выбрана *Visual Studio Code*.

Visual Studio Code (VS Code) – это редактор кода для разных языков программирования. Он относительно немного весит, гибкий и удобный. В нем можно писать, форматировать и редактировать код на разных языках.

VS Code позволяет легко писать, форматировать и редактировать код на разных языках. С его помощью можно быстро создать проект и структуру файлов в нем, он подсвечивает синтаксис кода и помогает автоматически править ошибки. В нем есть возможности для отладки и запуска кода на некоторых языках.

Редактор легко расширяется, поэтому к перечисленным функциям можно добавить новые – достаточно просто скачать нужное дополнение из официального каталога. Дополнения тоже распространяются бесплатно [3].

1.3 Обоснование выбора платформы

Выбор Python в качестве основной платформы для разработки программы генерации документов из шаблонов LaTeX основан на ряде весомых факторов, которые дополняют и расширяют возможности LaTeX, обеспечивая более гибкий, эффективный и универсальный подход к созданию документов.

Первоначально, LaTeX является стандартом для профессиональной верстки текста и документов, благодаря своей мощной системе компоновки и возможностям создания сложных шаблонов. Однако, интеграция LaTeX с Python позволяет значительно расширить функциональность и автоматизировать процесс создания документов.

Python, как один из самых популярных и гибких языков программирования, предоставляет обширные инструменты для работы с данными. Это важно для генерации документов, так как программа должна иметь возможность получать данные из различных источников и включать их в создаваемые отчеты, накладные или заявления. Благодаря библиотекам Pandas, numpy и другим, Python обеспечивает удобные средства для обработки, анализа и преобразования данных в формат, который может быть легко встроен в LaTeX-шаблоны.

Интеграция Python с LaTeX через библиотеки, такие как pylatex, предоставляет возможности для создания и редактирования LaTeX-документов из Python. Это обеспечивает гибкость в создании и настройке документов, а также контроль над форматированием и структурой. Кроме того, использование Python позволяет легко масштабировать программу и добавлять новые функции при необходимости.

Кроссплатформенность Python гарантирует, что разработанная программа будет работать на различных операционных системах, что делает ее доступной и удобной для использования широкому кругу пользователей. Таким образом, выбор Python в качестве основной платформы для разработки программы генерации документов из шаблонов LaTeX обусловлен его мощными возможностями, гибкостью, простотой в использовании и широким спектром инструментов, что обеспечивает создание эффективного и универсального генератора документов.

1.4 История TeX и LaTeX

Дональд Кнут опубликовал первую версию системы обработки печатных документов, известную ныне как TEX. Многие специалисты безоговорочно относят её к одному из выдающихся достижений XX столетия, приравнивая к созданию печатного станка Гутенбергом (Gutenberg, Iohann). TEX предвосхитил идеи, получившие признание на рубеже третьего тысячелетия. Система команд TEX, по сути, была первым языком разметки гипертекстов, к которым принадлежит широко известный ныне HTML (Hyper Text Markup Language) — язык разметки документов для интернета. Исполняемая программа tex, выполняющая преобразование размеченного текста в документ, пригодный для высококачественной печати, была чуть ли не первой из программ, которые сейчас принято называть парсерами.

TEX общепризнанно считается наиболее качественной системой подготовки печатных публикаций. Как сказано в словаре компьютерных терминов, TEX определяет стандарт, к которому пытаются приблизиться другие настольные издательские системы.

Следующий шаг сделал Лесли Лампорт (Lamport, Leslie). В начале восьмидесятых годов XX века он разработал систему подготовки печатных документов LATEX, основанную на форматирующих средствах TEX’a. LATEX позволил пользователю сконцентрировать свои усилия на содержании и структуре текста, не заботясь о деталях его оформления. Как и профессор Кнут, Л. Лампорт опередил своё время. Идея отделения содержания от формы,

реализованная в системе LATEX, нашла своё продолжение в XML — расширяемом языке разметки (eXtensible Markup Language), появившемся в конце девяностых годов XX века. Простая замена стиля документа1 в системе LATEX, как и замена стиля XSL (eXtensible Style Language), «надеваемого» на разметку XML, способна одинаково радикально изменить внешний вид документа.

LATEX реализован в виде формата, то есть надстройки над компактной системой базовых команд, встроенных в исполняемую программу tex. Надстройка, созданная самим Кнутом, называется «формат Plain TEX», или просто TEX. Формат TEX входит составной частью в формат LATEX.

LATEX содержит удобные средства генерации алфавитного указателя, списков литературы, рисунков и таблиц, развитые средства импортирования графики, обеспечивает автоматическую нумерацию формул, ссылок и других подобных объектов в сочетании с эффективным механизмом перекрёстного цитирования. Подлинного совершенства TEX и LATEX достигли в форматировании математических формул. Ни одна другая издательская система не сумела достичь тех же вершин в этой области издательского ремесла. Поэтому LATEX особенно популярен в научных кругах. За два десятилетия после изобретений Д. Кнута и Л. Лампорта появились прекрасные текстовые процессоры, но TEX и LATEX сохраняют ранее завоёванные позиции. Причина очевидна: уникальное качество подготовки печатной продукции, помноженное на полную совместимость версий TEX'а и LATEX'а для различных компьютеров.

В конце восьмидесятых годов TEX и LATEX достигли России. Был разработан алгоритм автоматического переноса русских слов. Кириллические шрифты разрабатывались в разных местах: в Вашингтонском университете, в Институте высоких энергий в Протвино, в издательстве «Мир» [4].

1.5 PDFLaTeX

При видимой стабильности языка разметки LATEX все последние годы происходила незримая работа по совершенствованию исполняемых модулей системы LATEX. Малозаметные улучшения, столь многочисленные, что их просто невозможно здесь перечислить, переросли в новое качество в начале XXI века. Русские пользователи LATEX'а могут вести отсчёт новой эры с ноября 2001 года, когда достоянием общественности стали шрифты cm-super, разработанные Владимиром Воловичем. Их появление фактически сделало ненужным традиционный сценарий компиляции исходного текста с разметкой

LATEX в dvi-файл, так как теперь есть все необходимое для прямого преобразования размеченного текста в pdfфайл. Если для преобразования исходного текста с разметкой LATEX в формат DVI (DeVice Independent), который был разработан Д. Кнутом специально для системы TEX, нужно использовать программу latex, то для преобразования того же текста в формат PDF (Portable Document Format), который в настоящее время доминирует в электронном документообороте, нужно использовать программу pdflatex. Строго говоря, и раньше документы LATEX можно было преобразовать в формат PDF, но при этом либо происходило ухудшение качества изображения, либо в исходный текст необходимо было вносить определённые изменения (например, подменять шрифты и рисунки), либо выполнять преобразование в 2–3 шага. Теперь в этом нет необходимости [4].

1.6 Логическое проектирование

В текстовых процессорах, которые позволяют в момент набора текста увидеть его на экране дисплея точно таким, как он будет выглядеть на бумаге, исповедуется концепция визуального проектирования (сокращённо WYSIWYG от английских слов What You See Is What You Get — «что видите, то и получите»). LATEX создан для логического проектирования печатного документа, которое позволяет сосредоточиться на содержании текста, возлагая на компьютер заботу по его оформлению. Преимущество логического проектирования над визуальным состоит в его гибкости. При визуальном проектировании «что видите, только то и получите». Текст, однажды подготовленный к печати, при таком подходе иной раз легче переписать заново, чем поправить. В наши дни то, что Л. Лампорт назвал логическим проектированием, связывают с концепцией разделения содержания и формы. Идеи логического проектирования проникли даже в визуальные редакторы, такие как Microsoft Word. Однако лишь немногие пользователи таких редакторов умеют использовать преимущества управления стилем документа, предпочитая «раскрашивать каждую букву по отдельности». Верно и обратное: логическое проектирование испытывает немалое влияние со стороны визуального. Существуют полувизуальные редакторы, например Scientific Word. В них на стадии разметки текста можно конструировать формулы, таблицы и т. д. из визуальных заготовок, которые перед компиляцией документа переводятся в команды, напоминающие команды LATEX’а, а затем запускается компилятор наподобие LATEX’а [4].

1.7 Входной файл

LATEX преобразует размеченный исходный текст в печатный документ. Исходный текст и печатный документ — это то, что в докомпьютерную эпоху соответственно называлось рукописью и типографским оттиском. Помимо собственно «рукописи» LATEX должен получить указания, что с ней делать в виде разметки. Размеченный исходный текст записывается во входной файл, который может быть создан с помощью любого редактора, способного сохранять файлы в текстовом формате. Многие редакторы записывают файл в своём собственном формате, непонятном другим текстовым процессорам, однако они обычно могут экспортировать его в текстовый формат. Входной файл для LATEX'а, как правило, имеет расширение tex.

В самом начале входной файл для LATEX'а должен содержать команду \documentclass[options]{class}, в которой [options] и {class} являются соответственно необязательным и обязательным аргументами. Обязательный аргумент в фигурных скобках должен содержать название класса печатного документа. Существует 6 стандартных классов: article, letter, report, book, proc, slides, которые имеются в самом минимальном варианте издательской системы LATEX. Это просто текстовые файлы с расширением cls. Кроме того, издатели журналов разработали для своих специфических целей множество нестандартных классов. Необязательный аргумент вместе с указывающими на его необязательность квадратными скобками может вообще отсутствовать. В необязательном аргументе может присутствовать любое количество опций, разделённых запятыми. Опции модифицируют стиль (способ оформления) печатного документа, определяемый выбором его класса. Следующей обязательной командой является \begin{document}. Текст перед \begin{document} называется преамбулой. Преамбула обычно содержит команды, производящие дополнительную настройку выбранного класса печатного документа, а также определения новых команд LATEX'а. Собственно текст документа начинается после \begin{document}, а заканчивается командой \end{document}. Всё, что следует за \end{document}, LATEX попросту игнорирует. Перечисленные три команды дают основные указания компилятору, как должен выглядеть печатный документ. Если одна из них пропущена, LATEX выдаст сообщение об ошибке при компиляции входного файла [4].

1.8 Буквы и символы

Большинство команд LATEX'а описывают логические структуры. Входной файл содержит структуры совершенно различных размеров — от отдельной буквы до текста документа в целом. Начнём с букв. LATEX распознаёт строчные и прописные буквы, десять цифр от 0 до 9 и шестнадцать знаков препинания: . , : ; ? " ‘ ’ ( ) [ ] - / * @. Причём «‘» и «’» используются в качестве левой и правой кавычек. Символ " в некоторых случаях печатается в виде двойных правых кавычек, но не имеет левого аналога, поэтому вряд ли его стоит использовать, поскольку двойные кавычки можно набрать, удвоив одинарные.

Пять символов: + = | < > применяются в таблицах и математических формулах, хотя + и = допускаются также и в обычном тексте.

Ещё десять символов: \ # $ % & { } _ ^ ~ зарезервированы для служебного пользования.

Помимо перечисленных имеются ещё невидимые символы, а именно: пробел и признак конца строки, которые вводятся в файл соответственно при нажатии клавиш Space (пробел) и Enter (ввод). Эти и некоторые другие невидимые символы (например, табулятор) интерпретируются LATEX'ом совершенно одинаково. Поэтому мы также будем называть их пробелами. Число следующих подряд пробелов несущественно: сто пробелов интерпретируются как один, если только они не составляют пустую строку. Пустой является строка, в которой кроме признака конца строки могут быть только другие пробелы. Пустая строка метит конец абзаца.

Служебный символ «\» (обратный слеш) имеет особое значение в LATEX'е. С него начинаются команды. Большинство команд, начинаясь с обратного слеша, состоят из одной или более букв. Считывая входной файл, компилятор знает, что дошёл до конца такой команды, если встретил символ, не являющийся буквой: цифру, знак препинания, служебный символ, пробел или конец строки. Чаще всего такая команда заканчивается пробелом. Поэтому компилятор игнорирует все пробелы, следующие за ней.

LATEX автоматически делает кернинг сочетаний букв и распознаёт лигатуры. Кернение — это способ печати, который регулирует средний промежуток между соседними литерами6, т. е. изображениями букв (символов). Этот промежуток зависит от ширины литер. Некоторые пары литер смотрятся лучше, когда они стоят более тесно, другие — когда раздвинуты. Лигатура — это комбинация символов, изображаемая одной литерой. Например, комбинации ff, fi, ‘‘, ’’, << и >> печатаются лигатурами ff,

fi, “, ”, « и ». Информация, необходимая для кернинга и выделения лигатур, заложена в файле метрики шрифтов [4].

1.9 Слова и предложения

LATEX игнорирует то, как набран исходный текст во входном файле. Он фиксирует только границы слов, предложений и абзацев. Слова, как обычно, отделяются друг от друга одним или несколькими пробелами. Предложение заканчивается точкой (.), восклицательным (!) или вопросительным (?) знаками, за которыми следует пробел. После этих знаков LATEX обычно чуть увеличивает пробел. Однако отечественные типографские правила не предусматривают подобного увеличения, поэтому пакет babel по умолчанию отменяет такое увеличение для русского языка. Пустая строка означает конец абзаца. Форматирование текста не зависит от числа пробелов или пустых строк: лишние просто игнорируются.

В машинописном тексте правые и левые кавычки печатаются совершенно одинаково: ". LATEX изображает символ " как прямые (") или правые кавычки (”) в зависимости от используемого шрифта. Чтобы напечатать левые кавычки (“), набирают два символа ‘ подряд: “. Тогда уж для единообразия правые кавычки (“) набирают в виде двух апострофов: ”. В текстах на английском и других иностранных языках встречаются также одинарные кавычки «‘» и «’». Ядро LATEX’а не содержит команд, соответствующих русским кавычкам в виде «ёлочек», но пакет babel с опцией russian обеспечивает один-два подходящих способа. Удобнее всего набирать такие кавычки при помощи лигатур << и >>.

Можно напечатать тире трёх различных размеров, если набрать один, два или три символа «-» подряд. LATEX печатает «--» и «---» как лигатуры. Знак «минус» не является дефисом, хотя во входном файле они изображаются одним и тем же символом. «Минус» может появляться только в математических формулах, которые мы обсудим в главе 6. LATEX использует особые правила кернинга математических формул [4].

1.10 Строки и абзацы

Пустая строка, не содержащая даже комментария, означает конец абзаца. Количество пустых строк не имеет значения. Так же не имеет значения, начинается первая строка в абзаце с первой или тринадцатой позиции — LATEX вставит в печатный документ отступ фиксированной длины. С отступа

начинается первая строка каждого абзаца, однако стандартные классы определяют, что первый абзац после заголовка отступа не имеет. Команда \noindent отменяет отступ, а \indent, наоборот, вставляет. LATEX автоматически разбивает текст на строки и переносит слова так, чтобы пробелы между словами в соседних строках также были по возможности равны. Однако он не способен определить логически связанные сочетания слов, которые желательно располагать на одной строке. Символ «~», вставленный между соседними словами, создаёт пробел, на котором LATEX никогда не разорвёт строку [4].

1.11 Выделение текста

В машинописном тексте слова, несущие особую смысловую нагрузку, выделяют подчеркиванием. LATEX умеет подчеркивать. Однако в типографском документе для выделения текста обычно используют курсив. Курсивный текст печатает команда \emph. Чтобы выделить курсивом какой-либо фрагмент текста, его нужно сделать аргументом этой команды. В выражении \emph{курсив} обратный слеш и следующие за ним четыре буквы \emph составляют имя команды, а {курсив} — её аргумент. Большинство команд либо не имеют аргументов, как \today, либо имеют один аргумент, как \emph{...}. Есть, впрочем, некоторое количество более сложных команд, имеющих несколько аргументов, причём каждый заключён в фигурные скобки, а пробелы между аргументами и между именем команды и первым аргументом игнорируются. Команды типа \emph могут быть вложены друг в друга вполне очевидным способом. Команда \emph переключает шрифт на курсивный, если до неё использовался прямой шрифт. В остальных случаях она включает прямой шрифт.

1.12 Процедуры

Если нужно выделить большой фрагмент текста, то использование команды \emph или декларации \em может вызвать непредвиденные трудности, так как входной файл станет трудночитаемым. Иногда совсем не просто найти закрывающую фигурную скобку «}», если она очень далеко отстоит от парной ей открывающей скобки {. Многие декларации имеют аналоги в виде процедур с теми же именами, но без обратного слеша в имени. Так, декларации \em соответствует процедура em. Любая процедура

начинается с командной скобки \begin{env} и закрывается командной скобкой \end{env}, где env — имя процедуры, в данном случае — em.

Текст между командными скобками составляет тело процедуры и форматируется в соответствии с тем, как она определена. Процедуры могут центрировать текст, выравнивать его по правой или левой границе, печатать в виде списков, таблиц, рисунков, формул и т. д. Командные скобки \begin{document} и \end{document} выделяют текст самого документа. Всё, что находится после команды \end{document}, попросту игнорируется, а команды, предшествующие \begin{document}, обычно определяют класс документа. Командные скобки \begin и \end ограничивают область действия деклараций наравне с фигурными. LATEX имеет множество процедур и допускает создание новых с помощью команды \newenvironment [4].

2 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

2.1 Обоснование необходимости разработки

В современном мире эффективность процессов и точность данных играют ключевую роль в успехе бизнеса и других организационных инициатив. С ростом объемов информации и сложности операций, организациям необходимо быстро адаптироваться к изменениям и эффективно управлять информацией. Это требует современных решений для автоматизации рутинных процессов, к которым относится и создание различных документов. Теоретическое обоснование необходимости разработки программного продукта для генерации отчетов, накладных, заявлений и других документов базируется на нескольких ключевых аспектах:

Организации сталкиваются с задачами, которые включают ручное создание и обработку документов. Этот процесс часто требует значительного времени, труда и подвержен ошибкам. Автоматизация с использованием программного продукта, который генерирует документы на основе шаблонов, позволяет значительно сократить время на создание документов и уменьшить вероятность ошибок. Теоретически, автоматизация увеличивает эффективность и освобождает ресурсы для более сложных задач.

Ручное создание документов приводит к увеличению вероятности ошибок из-за человеческого фактора. Ошибки в документах могут приводить к серьезным последствиям, включая финансовые потери, юридические проблемы или снижение доверия со стороны клиентов. Теоретически, программное обеспечение, которое автоматизирует создание документов на основе шаблонов, может улучшить точность и надежность, исключая или минимизируя возможность человеческих ошибок.

Создание документов вручную может привести к непоследовательности в форматах, стилях и структуре. Программное решение, генерирующее документы из шаблонов, обеспечивает стандартизацию. Теоретически, стандартизация повышает согласованность документов, облегчает их восприятие и обеспечивает соблюдение корпоративных или правовых стандартов.

Современные организации часто работают с различными источниками данных. Программное обеспечение для генерации документов может

интегрироваться с этими источниками, автоматизируя процесс заполнения шаблонов. Это облегчает обмен данными между системами и обеспечивает оперативное обновление информации. Теоретически, это способствует более гибкому и быстрому обмену данными внутри и между организациями.

Автоматизированные процессы могут существенно повысить производительность организации. Программное обеспечение, которое создает документы на основе шаблонов, позволяет сотрудникам фокусироваться на более ценных задачах, а не на рутинной работе. Теоретически, это ведет к увеличению эффективности и производительности, что, в свою очередь, способствует росту и развитию организации.

Таким образом, теоретическое обоснование необходимости разработки программного продукта для генерации документов из шаблонов LaTex основывается на стремлении повысить эффективность, точность, стандартизацию, удобство интеграции и производительность. Автоматизация процесса создания документов играет ключевую роль в улучшении работы организаций и предоставляет значительные преимущества по сравнению с ручным подходом.

2.2 Библиотека *tkinter*

Tkinter предоставляет множество элементов, также известных как виджеты, которые используются для создания графических интерфейсов в приложениях на Python. Каждый из этих элементов имеет уникальные свойства и предназначен для определенных задач. Далее приведено более подробное описание некоторых из наиболее распространенных элементов в Tkinter.

Основное окно (Tk). Основное окно представляет собой корневой элемент, в котором размещаются все остальные компоненты приложения. Это окно может иметь заголовок, значок, размеры по умолчанию, а также возможность масштабирования. Обычно с него начинается построение графического интерфейса.

Метка (Label). Метка служит для отображения текста или изображений на экране. Она не предназначена для взаимодействия с пользователем, а скорее для передачи информации или обозначения других элементов. Метки могут содержать текст, изображения или оба вместе, и могут быть стилизованы с использованием различных шрифтов, цветов и других свойств.

Кнопка (Button). Кнопка — это элемент, который пользователь может нажимать для выполнения какого-либо действия. Кнопки могут содержать

текст или изображения и часто связаны с функцией или командой, которая вызывается при нажатии. Они могут иметь различные стили, формы, размеры и состояния (например, "нажатая" или "отжатая").

Поле ввода текста (Entry). Поле ввода текста предназначено для однострочного ввода текста. Обычно его используют для ввода коротких данных, таких как имена, адреса электронной почты или пароли. Поле ввода может иметь различные ограничения, такие как максимальная длина текста или тип вводимых данных.

Многострочное поле ввода (Text). Многострочное поле ввода позволяет вводить большой объем текста. Этот элемент используется для редактирования текстов, таких как заметки, письма или большие блоки кода. В отличие от Entry, элемент Text позволяет использовать форматирование, перенос строк и другие возможности, характерные для полноценных текстовых редакторов.

Контейнеры (Frame). Контейнеры, также известные как фреймы, используются для группировки и организации других элементов. Они помогают структурировать интерфейс, создавая логические секции или группы виджетов. Контейнеры могут быть вложенными, позволяя создавать сложные иерархии элементов.

Флажок (Checkbutton). Флажок, или чекбокс, представляет собой переключатель, который может быть установлен или снят. Этот элемент часто используется для выбора одной или нескольких опций из множества возможных. Флажки могут быть сгруппированы, чтобы обеспечить выбор нескольких вариантов.

Радиокнопка (Radiobutton). Радиокнопка позволяет выбрать один из нескольких вариантов в группе. В отличие от флажков, радиокнопки предполагают, что из группы возможен только один выбор. Они часто используются для создания опросов, выбора режимов работы и других ситуаций, где требуется ограниченный выбор.

Список (Listbox). Список — это элемент, который отображает набор элементов, обычно в вертикальном или горизонтальном формате. Пользователи могут выбирать один или несколько элементов из списка. Списки часто используются для отображения больших объемов данных или для предоставления выбора среди множества опций.

Холст (Canvas). Холст позволяет рисовать графические элементы, такие как линии, фигуры, изображения и другие визуальные объекты. Это один из наиболее гибких элементов в Tkinter, позволяющий создавать сложные

графические компоненты, включая диаграммы, графики и другие визуализации.

Полоса прокрутки (Scrollbar). Полоса прокрутки позволяет прокручивать контент в другом виджете, таком как Listbox, Text, или Canvas. Полосы прокрутки могут быть горизонтальными или вертикальными и часто привязываются к другим виджетам, обеспечивая дополнительный контроль над отображением данных.

Одной из ключевых особенностей Tkinter является его способность работать на различных операционных системах, включая Windows, macOS и Linux, что делает его кроссплатформенным решением. Это означает, что приложения, созданные с использованием Tkinter, могут быть развернуты на разных системах без необходимости вносить существенные изменения в код. Кроссплатформенность обеспечивает разработчикам возможность сосредоточиться на функциональности приложения, не беспокоясь о специфических особенностях отдельных операционных систем.

Tkinter предоставляет различные варианты компоновки элементов в окне, такие как pack, grid и place. Эти методы позволяют управлять расположением виджетов внутри окна и обеспечивают гибкость при создании графических интерфейсов. Например, метод pack обеспечивает простую компоновку элементов по порядку, grid позволяет размещать элементы в сетке, а place обеспечивает точное позиционирование элементов на экране.

Tkinter также поддерживает расширенные возможности взаимодействия с пользователем. Обработка событий в Tkinter позволяет реагировать на различные действия пользователя, такие как нажатие кнопки, перемещение мыши или ввод текста. Это позволяет создавать интерактивные приложения, которые реагируют на действия пользователя в режиме реального времени. Разработчики могут привязывать события к конкретным функциям Python, что делает процесс обработки событий интуитивно понятным.

Библиотека Tkinter поддерживает взаимодействие с внешними ресурсами, такими как изображения и файлы. Это позволяет загружать изображения в виджеты Label или Canvas, а также работать с файлами для сохранения или загрузки данных. Более того, Tkinter предоставляет возможности для создания диалоговых окон, которые могут использоваться для выбора файлов, отображения сообщений или получения информации от пользователя.

Tkinter также предлагает инструменты для создания сложных графических элементов и визуализаций. С помощью виджета Canvas разработчики могут рисовать линии, фигуры, графики и другие графические

элементы. Это открывает возможности для создания более сложных интерфейсов, включая приложения для рисования, визуализации данных и анимации.

Благодаря своему богатому набору функций и простой интеграции с Python, Tkinter является мощным инструментом для создания графических интерфейсов. Он широко используется как начинающими разработчиками, так и опытными профессионалами, поскольку предоставляет обширный набор возможностей для создания гибких, интерактивных и кроссплатформенных приложений.

2.3 Библиотека *jinja2*

Jinja2 — это мощный и гибкий шаблонизатор для Python, который позволяет генерировать динамический текст на основе статических шаблонов. Он широко используется в веб-разработке для рендеринга HTML-страниц, но также нашёл применение в создании конфигурационных файлов, отчетов, электронных писем и других текстовых документов. Jinja2 предоставляет множество инструментов для построения сложных шаблонов, позволяя разработчикам легко смешивать статический контент с динамическим.

Основным принципом работы Jinja2 является использование шаблонов с переменными, которые могут быть динамически заменены данными во время рендеринга. Эти шаблоны могут содержать переменные, условные операторы, циклы, фильтры, макросы и другие элементы, которые позволяют создавать адаптивный контент. Переменные в Jinja2 могут быть простыми, представляющими отдельные значения, или сложными, с возможностью обращения к атрибутам объектов, методам или даже элементам вложенных структур, таких как списки или словари. Это делает Jinja2 чрезвычайно гибким инструментом, способным обрабатывать сложные сценарии.

Условные операторы в Jinja2, такие как if, elif и else, дают разработчикам возможность контролировать, какой контент будет отображаться в зависимости от условий. Например, можно показывать или скрывать определенные элементы на основе значений переменных или проводить более сложные логические проверки. Циклы for позволяют итерировать по коллекциям данных, создавая повторяющиеся элементы, что очень полезно при рендеринге таблиц, списков или других структур, требующих многократного отображения.

Фильтры в Jinja2 предоставляют мощные инструменты для преобразования данных перед их отображением. С их помощью можно

форматировать даты, преобразовывать текст в заглавные буквы, выполнять операции над числами и применять другие трансформации. Фильтры могут применяться к переменным прямо в шаблоне, что позволяет разработчикам гибко изменять вывод данных без изменения логики обработки.

Макросы в Jinja2 позволяют создавать повторно используемые блоки шаблонов. Это похоже на функции в программировании, где можно определять повторяющиеся фрагменты шаблона, которые можно вызывать с разными аргументами. Макросы помогают сократить дублирование кода, обеспечивая модульность и повторное использование, что значительно упрощает разработку сложных шаблонов.

Наследование шаблонов — ещё одна важная возможность Jinja2, которая позволяет создавать базовые шаблоны и расширять их для конкретных задач. Это особенно полезно в веб-разработке, где можно определить общий каркас для всех страниц и затем добавлять уникальный контент для каждой из них. Наследование способствует поддержанию консистентности дизайна и упрощает разработку.

Jinja2 также позволяет включать другие шаблоны с помощью инструкции include. Это дает возможность разбивать сложные шаблоны на более мелкие части, которые можно повторно использовать в различных контекстах. Такой подход помогает сохранить чистоту кода и облегчает его поддержку, особенно в крупных проектах.

Благодаря своей простоте использования, гибкости и богатому набору функций, Jinja2 стал одним из самых популярных инструментов для шаблонизации в экосистеме Python. Он широко используется в таких фреймворках, как Flask и Django, для динамического рендеринга веб-страниц, но его применение выходит далеко за пределы веб-разработки. Jinja2 можно использовать для генерации документов, конфигурационных файлов, электронных писем, отчетов и множества других текстовых данных, где требуется сочетание статического и динамического контента.

Jinja2 также отличается высокой производительностью, что делает его пригодным для использования в приложениях, требующих быстрого рендеринга большого количества шаблонов. Благодаря механизму кэширования, Jinja2 может сохранять откомпилированные шаблоны в памяти, что позволяет избежать повторной компиляции при каждом запросе. Это особенно полезно в веб-приложениях, где рендеринг HTML-страниц должен происходить быстро и без задержек. Кэширование помогает снизить нагрузку на сервер и улучшает общее время отклика приложений.

Кроме того, Jinja2 позволяет разработчикам создавать пользовательские фильтры и макросы, расширяя стандартный набор возможностей. Пользовательские фильтры могут быть определены для выполнения уникальных операций над данными, таких как специальное форматирование или преобразование. Макросы могут быть расширены, чтобы включать более сложную логику или даже взаимодействовать с внешними ресурсами. Это предоставляет разработчикам большой простор для творчества и позволяет создавать высоко адаптируемые и уникальные шаблоны, которые могут удовлетворить специфические требования проекта.

Jinja2 также поддерживает сложные структуры данных, такие как списки, словари и объекты, что позволяет работать с разнообразными данными в шаблонах. Это упрощает интеграцию с другими системами и фреймворками, делая Jinja2 совместимым с широким спектром источников данных. Например, в веб-приложениях данные часто поступают из баз данных или внешних API, и Jinja2 легко обрабатывает такие источники, предоставляя интуитивный способ отображения данных в шаблонах.

Для повышения безопасности Jinja2 включает в себя механизмы защиты от распространенных уязвимостей, таких как внедрение кода и XSS-атаки (межсайтовое выполнение скриптов). Он автоматически экранирует опасные символы и предоставляет дополнительные инструменты для обеспечения безопасности шаблонов. Это особенно важно при работе с пользовательскими данными или при создании публичных веб-приложений, где безопасность является ключевым фактором.

Jinja2 также предоставляет возможность управления глобальными переменными и контекстом шаблона. Это позволяет передавать данные, которые могут быть доступны во всех частях шаблона, без необходимости передавать их явно в каждый макрос или блок. Такой подход упрощает управление общими данными и позволяет более эффективно структурировать шаблоны.

Jinja2 широко используется в сообществе Python, что означает наличие большого количества документации, обучающих материалов и примеров использования. Существует также активное сообщество разработчиков, которые постоянно создают новые инструменты, расширения и библиотеки, которые улучшают функциональность Jinja2 и делают его еще более универсальным. Это способствует легкому обучению и быстрой интеграции Jinja2 в различные проекты.

Jinja2 также поддерживает интернационализацию и локализацию, что позволяет разработчикам создавать многоязычные приложения. Благодаря

встроенным функциям интернационализации, можно легко переключаться между языками и настраивать шаблоны для отображения контента на разных языках. Это делает Jinja2 отличным выбором для глобальных проектов, которые ориентированы на международную аудиторию.

В заключение, Jinja2 является одним из самых мощных и гибких шаблонизаторов в экосистеме Python. Его возможности позволяют создавать сложные динамические шаблоны, работать с различными типами данных, использовать макросы, фильтры и наследование, а также обеспечивать высокую производительность и безопасность. Благодаря своей кроссплатформенности и широкой поддержке, Jinja2 является отличным выбором для разработчиков, которые хотят создать надежные и динамичные приложения с использованием шаблонов.

2.3 Библиотека *subprocess*

subprocess — это модуль в Python, который позволяет запускать новые процессы, взаимодействовать с ними и захватывать их выходные данные. Он предоставляет средства для выполнения системных команд, запуска внешних программ, а также для управления вводом и выводом этих программ. Модуль subprocess заменяет старые подходы, такие как os.system, os.spawn, os.popen, предоставляя более гибкие и мощные инструменты. Вот обширное описание возможностей модуля subprocess.

Модуль subprocess позволяет запускать команды в операционной системе как бы изнутри программы Python. Это делает его чрезвычайно полезным для автоматизации системных задач, запуска внешних утилит или программ, а также для взаимодействия с другими процессами. Например, с помощью subprocess, вы можете запустить команду ls или dir, чтобы просмотреть содержимое директории, или команду ping, чтобы проверить соединение с удаленным сервером.

Одной из основных функций модуля subprocess является функция subprocess.run, которая выполняет команду и возвращает объект CompletedProcess, содержащий информацию о результате выполнения команды. При помощи этой функции вы можете передавать аргументы команды в виде списка строк, что обеспечивает безопасное и надежное выполнение команд. Это важно для избежания уязвимостей, связанных с выполнением небезопасных команд, таких как инъекции команд (command injection).

Модуль subprocess также позволяет перенаправлять стандартный ввод, вывод и ошибки процесса. Это означает, что вы можете захватить данные, которые программа записывает в стандартный вывод (stdout), или перенаправить ошибки в стандартный вывод (stderr). Это особенно полезно, когда вы хотите проанализировать выходные данные внешней команды или обработать ошибки, возникающие при ее выполнении. Например, вы можете перенаправить вывод в строку, чтобы использовать ее в дальнейшем в программе Python.

Еще одной мощной возможностью модуля subprocess является поддержка асинхронного выполнения. С помощью subprocess.Popen вы можете запустить процесс в фоне и взаимодействовать с ним по мере необходимости. Это позволяет запускать длительные задачи, не блокируя основное приложение Python, что особенно полезно при выполнении параллельных или асинхронных операций.

Модуль subprocess также предоставляет контроль над процессами, позволяя завершать процессы принудительно или ждать их завершения с использованием метода Popen.wait. Это полезно в сценариях, где нужно контролировать время выполнения процесса или обрабатывать случаи, когда процесс не завершается в ожидаемый срок. Также есть возможность устанавливать таймауты для процессов, чтобы предотвратить зависание приложения.

Модуль subprocess предоставляет ряд дополнительных возможностей, которые позволяют разработчикам более точно контролировать поведение запускаемых процессов и их взаимодействие с основным приложением. Одной из таких возможностей является управление текущим рабочим каталогом при запуске процесса. Вы можете указать каталог, из которого будет запущен новый процесс, что полезно, когда нужно выполнить команду в определенном контексте файловой системы.

subprocess также позволяет контролировать использование оболочки (shell) при запуске процессов. Указание параметра shell=True позволяет запускать команды с использованием системной оболочки, такой как Bash на Linux или Command Prompt на Windows. Однако важно помнить, что использование оболочки может быть потенциально небезопасным, так как оно подвержено инъекциям команд. Поэтому рекомендуется использовать прямой вызов команд без оболочки, когда это возможно, чтобы минимизировать риски безопасности.

Еще одной полезной возможностью subprocess является поддержка передачи входных данных в процесс через стандартный ввод (stdin). Это позволяет передавать данные в запущенный процесс, что может быть полезно при взаимодействии с программами, которые ожидают пользовательского ввода или данных из других источников. Вы можете использовать этот механизм для автоматизации взаимодействия с командной строкой или для передачи данных в работающие процессы.

Кроме того, subprocess предоставляет возможности для управления потоками ввода-вывода с использованием пайпов (pipes). С помощью этой функции вы можете перенаправлять стандартный ввод, вывод или ошибки процесса в другие процессы или файлы. Это позволяет создавать цепочки процессов, где вывод одного процесса становится вводом другого. Такая функциональность особенно полезна при автоматизации сложных системных задач, где несколько команд работают в связке.

Модуль subprocess также предоставляет возможности для обработки завершения процессов и управления ошибками. Вы можете проверять код завершения процесса, чтобы определить, было ли выполнение успешным или возникла ошибка. Это позволяет реализовывать логику обработки ошибок и принимать соответствующие действия в случае неудачного выполнения команды. При возникновении исключительных ситуаций, таких как превышение времени ожидания или ошибка при запуске процесса, subprocess генерирует исключения, которые можно обрабатывать для корректного завершения приложения.

Модуль subprocess также предоставляет возможности управления окружением процессов. Вы можете передавать переменные окружения в процесс, что позволяет контролировать контекст выполнения команды. Это особенно полезно при взаимодействии с системными утилитами или программами, которые зависят от определенных переменных окружения.

Таким образом, модуль subprocess в Python является мощным инструментом для запуска внешних процессов, взаимодействия с ними и управления их вводом и выводом. Его гибкость и широкий спектр возможностей позволяют разработчикам выполнять системные команды, автоматизировать задачи и взаимодействовать с внешними программами, делая его незаменимым в многих сценариях. С помощью subprocess, разработчики могут эффективно интегрировать Python-программы с системными утилитами и другими внешними инструментами, расширяя возможности своих приложений и автоматизируя рутинные задачи.

3 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

При запуске программы появляется окно с кнопками «Создать отчет по лабораторной работе» и «Создать отчет по своему шаблону» (Рисунок 3.1). На этом этапе пользователь выбирает нужный для него вариант и открывается соответствующее окно.

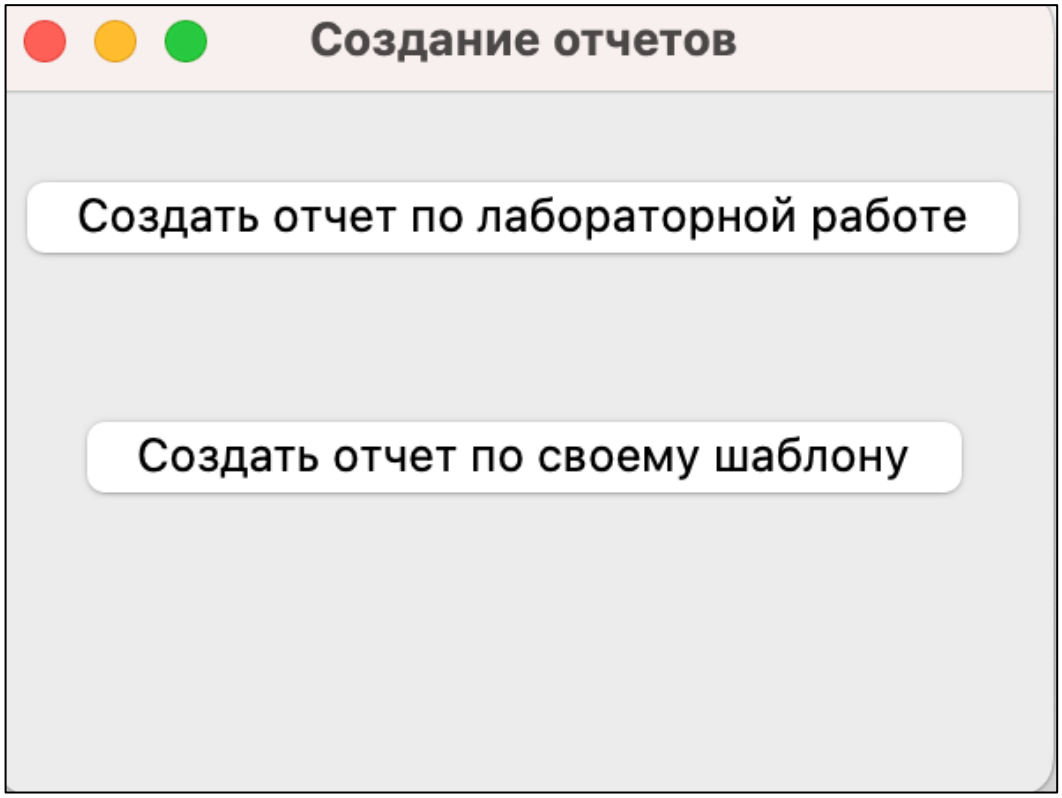

Рисунок 3.1 – Начальное окно программы

При выборе первого варианта, открывается окно с полями для ввода данных для заполнения шаблона lab.tex и кнопкой «Создать» (Рисунок 3.2).

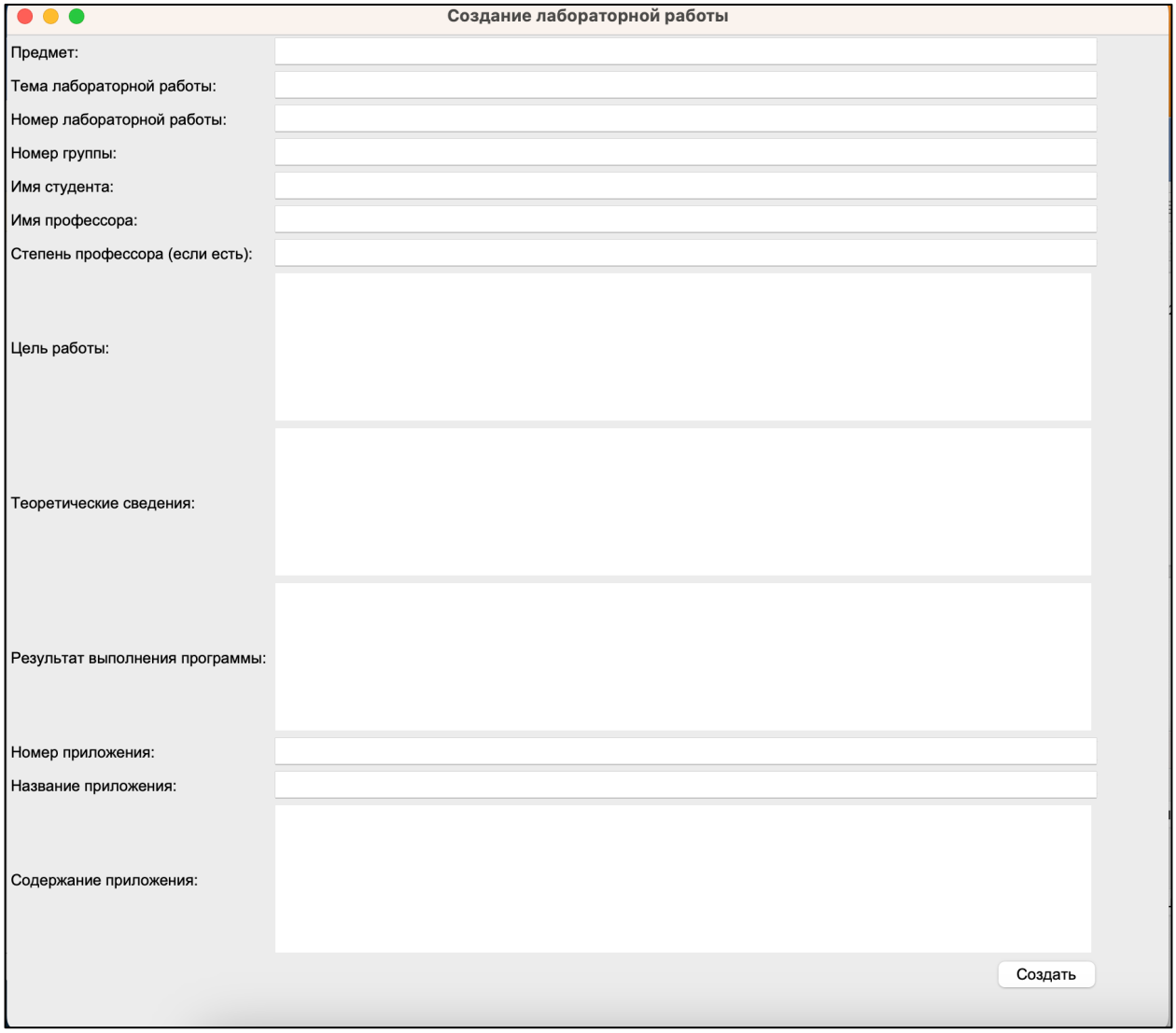

Рисунок 3.2 – Окно создания отчета по лабораторной работе

При нажатии на кнопку «Создать» проводится проверка, заполнены ли все обязательные поля. Если нет, то выводится соответствующее сообщение (Рисунок 3.3)

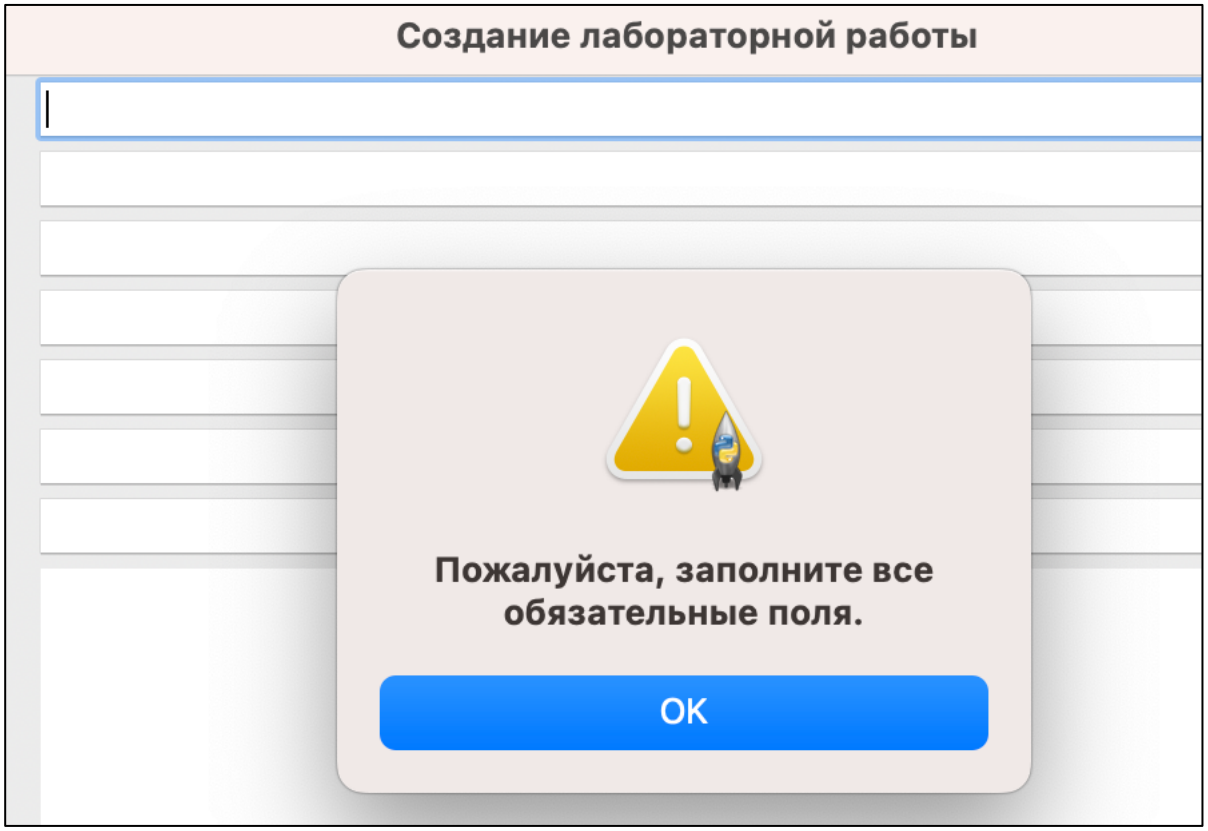

Рисунок 3.3 – Сообщение об ошибке при незаполненных полях

Если все поля заполнены (Рисунок 3.4), то данные сохраняются, очищаются от конфликтных символов и передаются в функцию для заполнения шаблона.

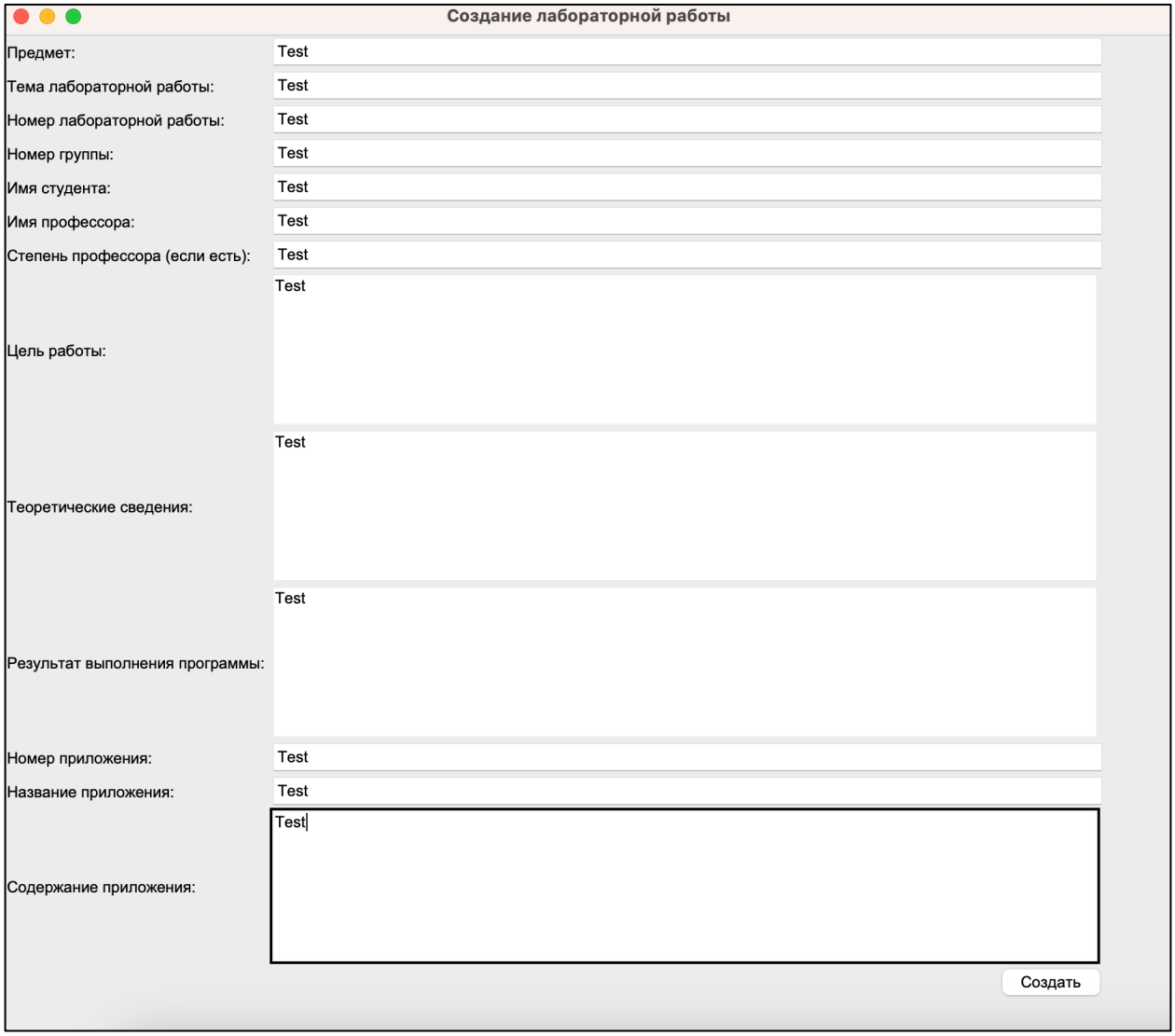

Рисунок 3.4 – Окно создания отчета по лабораторной работе с заполненными полями

Далее пользователю предлагается выбрать папку для сохранения отчета и ввести имя файла (Рисунок 3.5).

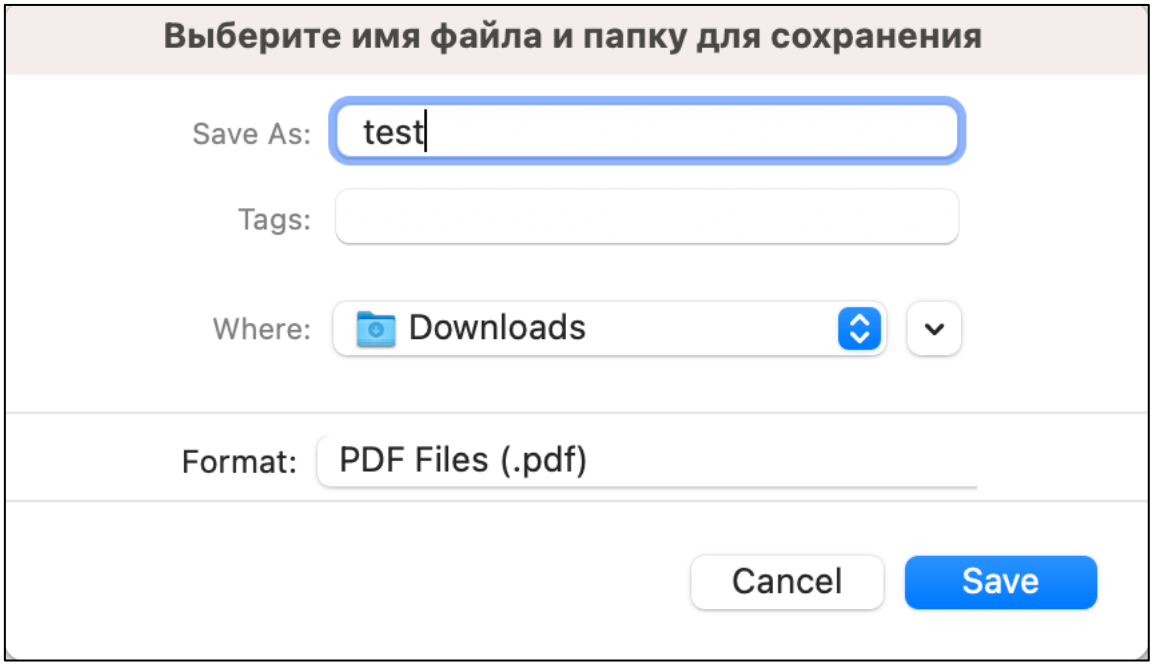

Рисунок 3.5 – Окно с запросом выбора папки сохранения и ввода имени файла

После успешного выбора папки сохранения и ввода имени файла происходит подстановка данных в шаблон и создание pdf-файла (Рисунок 3.6).

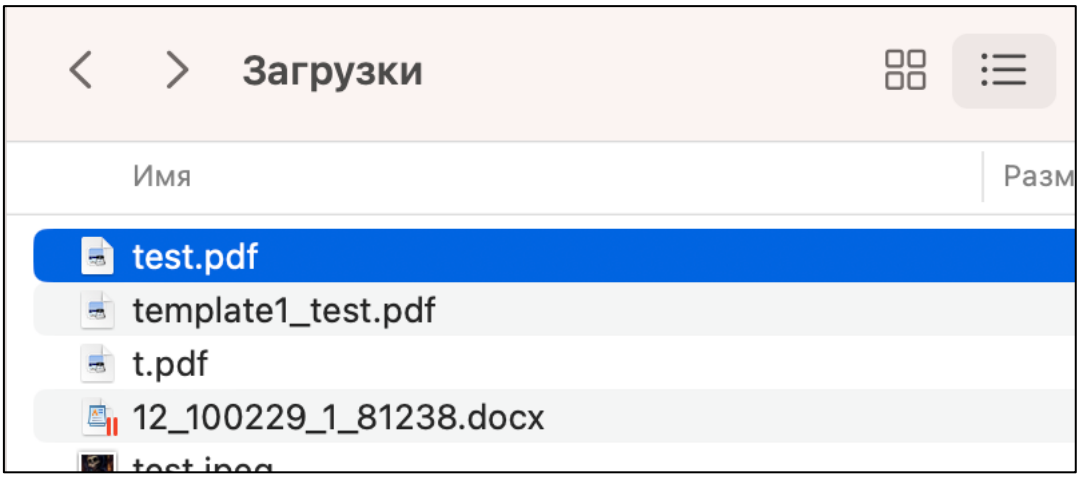

Рисунок 3.6 – Пример созданного файла в указанной папке

Также пользователь может воспользоваться контекстным меню для вставки изображения или листинга кода (Рисунок 3.7).

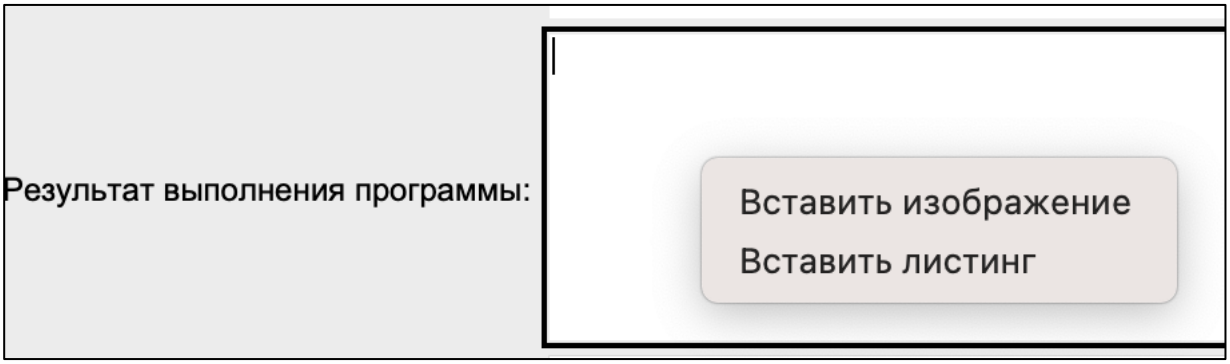

Рисунок 3.7 – Контекстное меню

Если выбрать пункт «Вставить изображение», то сначала пользователю будет предложено выбрать файл с изображением и, после выбора, на место курсора в активном поле будет вставлена конструкция latex (Рисунок 3.8). В конечном pdf-файле изображение подставится на место этой конструкции (Рисунок 3.9).

Рисунок 3.8 – Пример вставки изображения

Пример текста Пример текста Пример текста

Рисунок 2.1 – Подпись вашего изображения

Рисунок 3.9 – Пример отчета с изображением

Если же в контекстном меню выбрать пункт «Вставить листинг», то на место курсора будет вставлена конструкция latex (Рисунок 3.10), которая в pdf преобразует листинг в требуемый вид (Рисунок 3.11).

```
   \begin{lstlisting}[caption=Подпись вашего листинга]
#include <stdio.h>

int main() /* Пример вашего кода" */
{
  printf("Hello world!");
  return 0;
```

Рисунок 3.10 – Пример вставки листинга

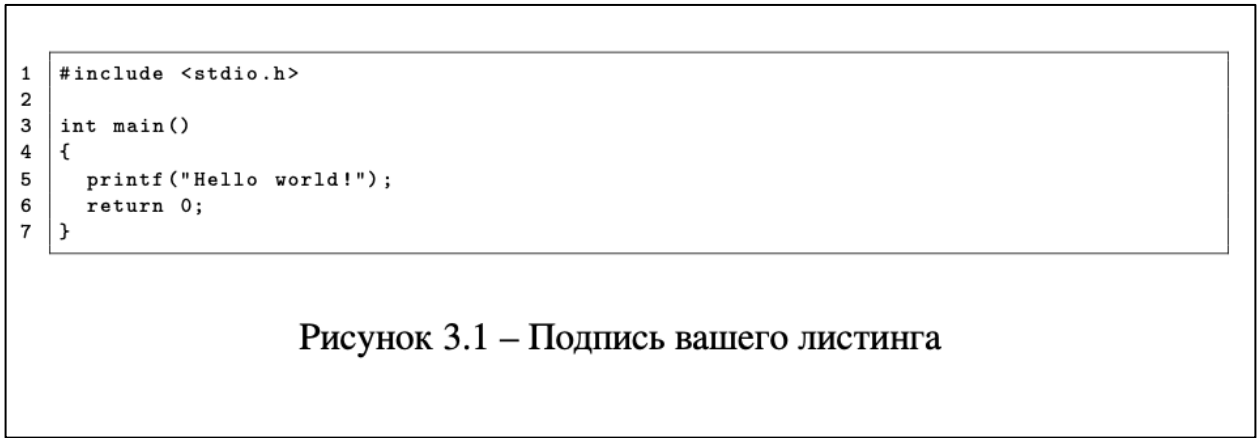

Рисунок 3.11 – Пример отчета с листингом

Также пользователю доступна функция создания отчета по собственному шаблону. Изначально ему будет предложено выбрать файл с шаблоном. После чего, в шаблоне будут найдены все переменный, которые должны быть заполнены пользователем, а затем будет создано окно с полями для ввода всех необходимых данных (Рисунок 3.12).

Рисунок 3.12 – Пример окна для создания отчета по собственному шаблону пользователя

После ввода всех значений будут выполнены такие же операции, как и при создании отчета по лабораторной работе.

ЗАКЛЮЧЕНИЕ

В данной курсовой работы мы можем подвести итог результатов исследования и разработки генератора отчетов с использованием LaTex.

Во-первых, мы детально рассмотрели концепцию генератора отчетов, исследовали его структуру и процессы, которые позволяют автоматизировать создание профессионально выглядящих документов. Наш анализ показал, что LaTex является мощным инструментом для верстки сложных документов, обеспечивая высокое качество вывода, гибкость в оформлении и множество возможностей по структурированию информации.

Во-вторых, в ходе реализации проекта была создана программа на языке Python, которая генерирует отчеты на основе LaTex-шаблонов. Использование Python для обработки данных и шаблонизации с помощью Jinja2 позволило создать динамический процесс, в котором можно легко менять входные данные и получать готовые документы. Было показано, что такой подход обеспечивает высокую производительность, позволяет избежать ручного труда и снижает вероятность ошибок при создании отчетов.

Кроме того, интеграция с LaTex открывает множество перспектив для дальнейшего развития проекта. Например, можно добавлять более сложные элементы оформления, такие как таблицы, графики, или вставки изображений, чтобы сделать отчеты более информативными и наглядными. Возможность автоматизировать генерацию отчетов также означает, что этот инструмент может быть использован в различных контекстах, от научных исследований до бизнес-анализа.

В целом, разработка генератора отчетов с использованием LaTex продемонстрировала свою эффективность и универсальность. Этот проект может быть использован в различных областях, где требуется быстрое и качественное создание документов. В дальнейшем, возможны расширения функционала, такие как интеграция с другими системами, улучшение пользовательского интерфейса и дополнительные возможности по шаблонизации.

Таким образом, проведенная работа показала, что генератор отчетов с использованием LaTex может стать ценным инструментом, способствующим автоматизации процессов создания документов и повышению эффективности работы в самых разных сферах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Понятие платформы программного обеспечения. Сравнительная характеристика используемых платформ (Windows, Linux и др.) [Электронный ресурс]. – Режим доступа: https://studfile.net/preview/4235646/. – Дата доступа: 26.03.2024.

[2] Язык программирования Python [Электронный ресурс]. – Режим доступа: https://web-creator.ru/articles/python. – Дата доступа: 26.03.2024.

[3] Visual Studio Code [Электронный ресурс]. – Режим доступа: https://blog.skillfactory.ru/glossary/visual-studio-code/. – Дата доступа: 26.03.2024.

[4] LaTex по-русски [Электронный ресурс]. – Режим доступа: https://elitagroup.ru/content/school/web/20170130/. – Дата доступа: 26.03.2024.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

Листинг 1 – *lab.tex*

```
\documentclass[a4paper, hidelinks, 14pt]{extarticle}

\usepackage{tempora}
\input{lab_template/sys/packages} % Подключаемые пакеты
\input{lab_template/sys/styles}   % Пользовательские стили
\begin{document}

\input{lab_template/sys/names}    % Переопределение именований

\begin{titlepage}
    \thispagestyle{empty}
    \setlength{\parindent}{0ex} % set paragraph indenting to zero

    \begin{center}
        Министерство образования Республики Беларусь \\
        \vspace{1ex}
        Учреждение образования\\
        БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ\\
        ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
        \begin{flushleft}
       \vspace{1ex}
        Факультет компьютерных систем и
      сетей \\
        Кафедра информатики\\
        Дисциплина {{ subject }}
        \end{flushleft}
    \end{center}

    \vspace{50mm}

    \begin{center}
        \textbf{ ОТЧЕТ } \\
        к лабораторной работе №{{ lab_num }} \\
        на тему <<{{ lab_topic }}>>
    \end{center}

    \vspace{50mm}

    \begin{minipage}{.55\linewidth}
        Выполнил студент группы {{ group_num }}

        \smallskip

        Проверил {{ degree }}
    \end{minipage}
    \hfill
    \begin{minipage}{.4\linewidth}
        \begin{flushright}
            {{ student_name }}

            \smallskip
```

```
            {{ professor_name }}
        \end{flushright}
    \end{minipage}

    \vfill
    \begin{center}
        Минск {{ year }}
    \end{center}

    \setlength{\parindent}{1.25cm} % reset paragraph indenting
\end{titlepage}

\renewcommand{\cftdotsep}{0.5}
\setcounter{page}{2} % title and task pages
\tableofcontents
\newpage

\addtocontents{toc}{\protect\setcounter{tocdepth}{0}}
\section{ЦЕЛЬ РАБОТЫ}
\addtocontents{toc}{\protect\setcounter{tocdepth}{2}}
\addcontentsline{toc}{section}{1 Цель работы}
{{ aims }}
\newpage

\addtocontents{toc}{\protect\setcounter{tocdepth}{0}}
\section{ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ}
\addtocontents{toc}{\protect\setcounter{tocdepth}{2}}
\addcontentsline{toc}{section}{2 Теоретические сведения}
{{ theory }}
\newpage



\addtocontents{toc}{\protect\setcounter{tocdepth}{0}}
\section{РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ}
\addtocontents{toc}{\protect\setcounter{tocdepth}{2}}
\addcontentsline{toc}{section}{3 Результат выполнения программы}
{{ result }}
\newpage


%\input{text/conclusion}  % Заключение

\renewcommand{\thefigure}{\Asbuk{section}.\arabic{figure}}
\renewcommand{\thetable}{\Asbuk{section}.\arabic{table}}
\renewcommand{\thelstlisting}{\Asbuk{section}.\arabic{lstlisting}}

\pagestyle{fancy}
\thispagestyle{plain}

\section*{
        \begin{center}
        ПРИЛОЖЕНИЕ {{ appendix_num }}\\
        {{ appendix_title }}% Центрирование
        \end{center}
        }
\addcontentsline{toc}{section}{Приложение {{ appendix_num }} {{ appendix_title
}}}
{{ appendix }}
\newpage
```

```
\end{document}
```

Листинг 2 – *styles.tex*

```
    %%% Язык текста %%%
\selectlanguage{russian}

%%% Кодировки и шрифты %%%
%\renewcommand{\rmdefault}{ftm} % Включаем Times New Roman


%%% Макет страницы %%%
\geometry{a4paper,top=20mm,left=31mm,right=15mm,bottom=27mm}
\setstretch{1.05}

%%% Выравнивание и переносы %%%
\sloppy                         % Избавляемся от переполнений
\clubpenalty=10000              % Запрещаем разрыв страницы после первой строки абзаца
\widowpenalty=10000             %  Запрещаем разрыв страницы после последней строки абзаца
\interfootnotelinepenalty=10000 % Запрет разрывов сносок

%%% Настройки полей %%%

% Титульная страница
\fancypagestyle{empty}{%
\fancyhf{} % clear all header and footer fields
\renewcommand{\headrulewidth}{0pt}
\renewcommand{\footrulewidth}{0pt}
% \setlength{\footskip}{9mm}
\setlength{\headheight}{4mm}
}

% Основной текст
\fancypagestyle{plain}{%
\fancyhf{} % clear all header and footer fields
\fancyfoot[R]{\thepage}
\renewcommand{\headrulewidth}{0pt}
\renewcommand{\footrulewidth}{0pt}
\setlength{\footskip}{11mm}
\setlength{\headheight}{4mm}
}

\pagestyle{plain}

%%% Оформление текста

\setlength{\parskip}{0mm}
\setlength{\parindent}{1.25cm}
\raggedbottom{}

\renewcommand\floatpagefraction{.9}
\renewcommand\dblfloatpagefraction{.9} % for two column documents
\renewcommand\topfraction{.9}
\renewcommand\dbltopfraction{.9}       % for two column documents
\renewcommand\bottomfraction{.9}
\renewcommand\textfraction{.1}
\setcounter{totalnumber}{50}
\setcounter{topnumber}{50}
\setcounter{bottomnumber}{50}
```

```
%%% Оформление заголовков
\newcommand{\sectionbreak}{\clearpage}

\titleformat{\section}{\large\bfseries}{\thesection}{\wordsep}{}
\titlespacing*{\section}{\parindent}{\baselineskip}{\baselineskip}

\titleformat{name=\section,numberless}{\large\bfseries\filcenter}{}{0mm}{}
\titlespacing*{name=\section,numberless}{0mm}{\baselineskip}{\baselineskip}

\titleformat{name=\subsection}{\normalsize\bfseries}{\thesubsection}{\wordsep
}{}
\titlespacing*{\subsection}{\parindent}{\baselineskip}{\baselineskip}

\titleformat{name=\subsection,numberless}{\normalsize\bfseries}{}{0mm}{}
\titlespacing*{name=\subsection,numberless}{0mm}{\baselineskip}{\baselineskip
}

\titleformat{name=\subsubsection}{\normalsize\bfseries}{\thesubsubsection}{\w
ordsep}{}
\titlespacing*{\subsubsection}{\parindent}{\baselineskip}{\baselineskip}

\titleformat{name=\subsubsection,numberless}{\normalsize\bfseries}{}{0mm}{}
\titlespacing*{name=\subsubsection,numberless}{0mm}{\baselineskip}{\baselines
kip}

\counterwithout{paragraph}{subsubsection}
\counterwithin{paragraph}{subsection}
\renewcommand{\theparagraph}{\thesubsection.\arabic{paragraph}}
\setcounter{secnumdepth}{4}

\titleformat{name=\paragraph}[runin]{\normalsize\bfseries}{\theparagraph}{\wo
rdsep}{}
\titlespacing*{\paragraph}{\parindent}{\baselineskip}{\wordsep}

%%% Оформление списков
\setlist[1]{itemindent=1.85cm,leftmargin=0mm,itemsep=0mm,topsep=0mm,parsep=0m
m}
\setlist[itemize,1]{label=---}
\setlist[enumerate,1]{label=\arabic*}

\setlist[2]{itemindent=3.1cm,leftmargin=0mm,itemsep=0mm,topsep=0mm,parsep=0mm
}

% Стиль для списков, на которые есть ссылки в тексте
\AddEnumerateCounter{\asbuk}{\@asbuk}{\cyrm}
\newlist{reflist}{enumerate}{1}
\setlist*[reflist,1]{label=\asbuk*)}
\setlist*[reflist,2]{label=\arabic*)}

%%% Оформление сносок

\deffootnote[1.65cm]{0mm}{1.25cm}{\textsuperscript{\thefootnotemark) }}
\renewcommand{\footnotesize}{\normalsize\selectfont}
\setlength{\footnotesep}{\parsep}

%%% Оформление ссылок
\urlstyle{same}

%%% Размеры текста формул %%%

\DeclareMathSizes{12}{12}{6}{4}
```

```
%%% Расстояние между формулами

\AtBeginDocument{%
  \setlength\abovedisplayskip{\baselineskip}%
  \setlength\belowdisplayskip{\baselineskip}%
  \setlength\abovedisplayshortskip{\baselineskip}%
  \setlength\belowdisplayshortskip{\baselineskip}%
}

%%% Расстояние между плавающими элементами

\setlength{\floatsep}{1.5\baselineskip plus 0mm minus 0mm}      % between top
floats
\setlength{\textfloatsep}{1.5\baselineskip  plus  0mm  minus  0mm}  %  between
top/bottom floats and text
\setlength{\intextsep}{1.5\baselineskip plus 0mm minus 0mm}     % between text
and float
\setlength{\dbltextfloatsep}{1.5\baselineskip plus 0mm minus 0mm}
\setlength{\dblfloatsep}{1.5\baselineskip plus 0mm minus 0mm}

%% Нумерация плавающих элементов

\counterwithin{figure}{section}
\counterwithin{table}{section}
\counterwithin{equation}{section}

\makeatletter
\AtBeginDocument{%
\renewcommand{\thetable}{\thesection.\arabic{table}}
\renewcommand{\thelstlisting}{\thesection.\arabic{lstlisting}}
\renewcommand{\thefigure}{\thesection.\arabic{figure}}
\let\c@lstlisting\c@figure}
\makeatother

%% Подписи плавающих элементов

\fboxsep=4mm
\fboxrule=0.1mm

\captionsetup[figure]{
  labelsep=endash,
  justification=centering,
  singlelinecheck=false,
  position=bottom,
  parskip=\parskip,
  belowskip=-0.6\baselineskip,
  skip=1.4\baselineskip}

\captionsetup[table]{
  labelsep=endash,
  justification=raggedright,
  singlelinecheck=false,
  position=top,
  belowskip=-0.4\baselineskip,
  skip=0mm}

\captionsetup[longtable]{
  labelsep=endash,
  justification=raggedright,
  singlelinecheck=false,
  position=top,
```

```
  belowskip=0.4\baselineskip,
  skip=0mm}

\captionsetup[lstlisting]{
  labelsep=endash
}

\lstset{
basicstyle=\scriptsize\ttfamily,
numberstyle=\scriptsize\ttfamily,
keywordstyle=\bfseries,
commentstyle=\itshape,
numbers=left,
stepnumber=1,
frame=single,
resetmargins=true,
xleftmargin=7mm,
xrightmargin=2mm,
captionpos=b,
keepspaces=true,
breaklines=true,
aboveskip=1.6\baselineskip,
belowskip=1.4\baselineskip,
abovecaptionskip=1.2\baselineskip}

\renewcommand{\arraystretch}{1.5}

%%% Настройка размеров вертикальных отступов

\renewcommand{\smallskip}{\vspace{0.3\baselineskip}}
\renewcommand{\bigskip}{\vspace{0.8\baselineskip}}

%%% Содержание %%%
\renewcommand{\cfttoctitlefont}{\hfil \large\bfseries}

\setlength{\cftparskip}{0mm}
\setlength{\cftbeforesecskip}{0mm}
\setlength{\cftaftertoctitleskip}{\baselineskip}
\cftsetpnumwidth{4mm}

\renewcommand{\cftsecfont}{}
\renewcommand{\cftsecpagefont}{\normalsize}
\renewcommand{\cftsecleader}{\cftdotfill{\cftdotsep}}

\setlength{\cftsecindent}{0mm}
\setlength{\cftsecnumwidth}{4mm}

\setlength{\cftsubsecindent}{4mm}
\setlength{\cftsubsecnumwidth}{8mm}

\setlength{\cftsubsubsecindent}{12mm}
\setlength{\cftsubsubsecnumwidth}{12mm}

%%% Библиография %%%

\makeatletter
\bibliographystyle{sys/ugost2003} % Оформляем библиографию в соответствии с
ГОСТ 7.1 2003

\let\oldthebibliography=\thebibliography
\let\endoldthebibliography=\endthebibliography
\renewenvironment{thebibliography}[1]{
```

```
  \begin{oldthebibliography}{#1}
    \setlength{\parskip}{0mm}
    \setlength{\itemsep}{0mm}
}
{
\end{oldthebibliography}
}
```

Листинг 3 – *packages.tex*

```
%%% Поля и разметка страницы %%%
\usepackage{pdflscape}   % Для включения альбомных страниц
\usepackage{geometry} % Для последующего задания полей
\usepackage{setspace} % Для интерлиньяжа
\usepackage[14pt]{extsizes}
\usepackage{titlesec}
\usepackage{tocloft}
\usepackage{enumitem}
\usepackage{fancyhdr}
\usepackage{chngcntr} % Для нумерации параграфов
\usepackage{calc}     % Для расчета ширины элементов
\usepackage{url}      % Для настройки отображение ссылок
\usepackage{pdfpages} % Для встраивания PDF

%%% Кодировки и шрифты %%%
\usepackage{cmap}                  % Улучшенный поиск русских слов в полученном
pdf-файле
\usepackage[T1,T2A]{fontenc}       % Поддержка русских букв
\usepackage[utf8x]{inputenc}       % Кодировка utf8
\usepackage[english, russian]{babel} % Языки: русский, английский

%\usepackage{pscyr}                                    %    Красивые    русские
шрифты пофиксить

%%% Математические пакеты %%%
\usepackage{amsthm,amsfonts,amsmath,amssymb,amscd} % Математические дополнения
от AMS

%%% Оформление абзацев %%%
\usepackage{indentfirst} % Красная строка

%%% Цвета %%%
\usepackage{color}

%%% Таблицы %%%
\usepackage{longtable}             % Длинные таблицы
\usepackage{tabularx}
\usepackage{tabulary}
\usepackage{multirow,makecell,array} % Улучшенное форматирование таблиц

%%% Сноски %%%
\usepackage{tablefootnote} % Для сносок в таблицах
\usepackage{scrextend}

%%% Общее форматирование
\usepackage[tableposition=top]{caption}
\captionsetup[figure]{labelsep=space,justification=centering,singlelinecheck=
false}
\captionsetup[lstlisting]{labelsep=space,justification=centering,singlelinech
eck=false}
```

```
\usepackage{soul}                  % Поддержка переносоустойчивых подчёркиваний
и зачёркиваний
\usepackage{multicol}

%%% Библиография %%%
\usepackage{cite} % Красивые ссылки на литературу

%%% Гиперссылки %%%
\usepackage[unicode,plainpages=false,pdfpagelabels=false]{hyperref}

%%% Изображения %%%
\usepackage{graphicx} % Подключаем пакет работы с графикой

%%% Листинги %%%
\usepackage{listings} %% собственно, это и есть пакет listings

%%% Рисунки TEX %%%
\usepackage{tikz}
```

Листинг 4 – *main.py*

```
import tkinter as tk
from tkinter import filedialog, messagebox
from jinja2 import Template
import subprocess
import os
import datetime
import time
import re

# Функция для генерации лабораторной работы
def generate_random_report(tex_file_path, invoice_data):
    # Заменяем потенциально проблемные символы
    problem_characters = {
        '\u0301': "",  # Комбинированный акцент
        '\u2013': "-",  # Длинное тире (en-dash)
        '\u2014': "--",  # Длинное тире (em-dash)
        '\u2018': "'",  # Левый одиночный кавычки
        '\u2019': "'",  # Правый одиночный кавычки
        '\u201c': '"',  # Левые двойные кавычки
        '\u201d': '"',  # Правые двойные кавычки
        '\u2026': "...",  # Многоточие
        '\u2032': "'",  # Минутный штрих
        '\u2033': '"',  # Секундный штрих
        # Добавьте другие потенциально опасные символы здесь
    }

    cleaned_invoice_data = {
    clean_text(key,        problem_characters):        clean_text(value,
problem_characters)
    for key, value in invoice_data.items()
    }

    with open(tex_file_path, 'r') as file:
        template_content = file.read()
```

```
            template = Template(template_content)
            rendered_content = template.render(cleaned_invoice_data)

        # Используем asksaveasfilename для запроса имени файла и папки для
сохранения
            save_path = filedialog.asksaveasfilename(
                title="Выберите имя файла и папку для сохранения",
                defaultextension=".pdf",
                filetypes=[("PDF Files", "*.pdf")]
            )

            if not save_path:
                return  # Если пользователь отменил выбор, выходим из функции

            # Сохраняем LaTex-код в выбранное имя файла (заменяем на .tex)
            tex_path = "generated_report.tex"
            with open(tex_path, 'w') as output_file:
                output_file.write(rendered_content)


            # Генерируем PDF из LaTex
            subprocess.run(['pdflatex', tex_path])
            time.sleep(0.2)
            subprocess.run(['pdflatex', tex_path])
            time.sleep(0.2)
            subprocess.run(['pdflatex', tex_path])
            time.sleep(2)




            # Проверяем, сгенерирован ли файл PDF, и перемещаем его на выбранное
место
            if os.path.exists("generated_report.pdf"):
                subprocess.run(['mv',    "generated_report.pdf",    save_path],
check=True)

            # Удаляем временные файлы, если они существуют
            temp_files   =   ["generated_report.tex",   "generated_report.aux",
"generated_report.log",    "generated_report.out",    "generated_report.toc",
"generated_report.fdb_latexmk",                        "generated_report.fls",
"generated_report.synctex.gz"]
            remove_files(temp_files)

            messagebox.showinfo("Успех",  f"Лабораторная  работа  сохранена  как
{save_path}")


        # Функция для генерации лабораторной работы
        def  generate_lab(subject,  lab_topic,  lab_num,  group_num,  degree,
student_name,   professor_name,   aims,   theory,   result,   appendix_num,
appendix_title, appendix):
            # Заменяем потенциально проблемные символы
            problem_characters = {
```

```
            '\u0301': "",  # Комбинированный акцент
            '\u2013': "-",  # Длинное тире (en-dash)
            '\u2014': "--",  # Длинное тире (em-dash)
            '\u2018': "'",  # Левый одиночный кавычки
            '\u2019': "'",  # Правый одиночный кавычки
            '\u201c': '"',  # Левые двойные кавычки
            '\u201d': '"',  # Правые двойные кавычки
            '\u2026': "...",  # Многоточие
            '\u2032': "'",  # Минутный штрих
            '\u2033': '"',  # Секундный штрих
            # Добавьте другие потенциально опасные символы здесь
        }

        fields_to_clean = [subject, lab_topic, lab_num, group_num, degree,
student_name,   professor_name,   aims,   theory,   result,   appendix_num,
appendix_title, appendix]

        # Очищаем все поля от проблемных символов
        cleaned_fields = [clean_text(field, problem_characters) for field in
fields_to_clean]

        # Распаковываем очищенные значения
        subject,  lab_topic,  lab_num,  group_num,  degree,  student_name,
professor_name, aims, theory, result, appendix_num, appendix_title, appendix =
cleaned_fields

        invoice_data = {
            "subject": subject,
            "lab_topic": lab_topic,
            "lab_num": lab_num,
            "group_num": group_num,
            "year": str(datetime.datetime.now().year),
            "degree": degree,
            "student_name": student_name,
            "professor_name": professor_name,
            "aims": aims,
            "theory": theory,
            "result": result,
            "appendix_num": appendix_num,
            "appendix_title": appendix_title,
            "appendix": appendix
        }

        with open('lab_template/lab.tex', 'r') as file:
            template_content = file.read()

        template = Template(template_content)
        rendered_content = template.render(invoice_data)

    # Используем asksaveasfilename для запроса имени файла и папки для
сохранения
        save_path = filedialog.asksaveasfilename(
            title="Выберите имя файла и папку для сохранения",
            defaultextension=".pdf",
```

```
            filetypes=[("PDF Files", "*.pdf")]
        )

        if not save_path:
            return  # Если пользователь отменил выбор, выходим из функции

        # Сохраняем LaTex-код в выбранное имя файла (заменяем на .tex)
        tex_path = "generated_lab.tex"
        with open(tex_path, 'w') as output_file:
            output_file.write(rendered_content)


        # Генерируем PDF из LaTex
        subprocess.run(['pdflatex', tex_path])
        time.sleep(0.2)
        subprocess.run(['pdflatex', tex_path])
        time.sleep(0.2)
        subprocess.run(['pdflatex', tex_path])
        time.sleep(2)




        # Проверяем, сгенерирован ли файл PDF, и перемещаем его на выбранное
место
        if os.path.exists("generated_lab.pdf"):
            subprocess.run(['mv',      "generated_lab.pdf",      save_path],
check=True)

        # Удаляем временные файлы, если они существуют
        temp_files     =     ["generated_lab.tex",     "generated_lab.aux",
"generated_lab.log",         "generated_lab.out",         "generated_lab.toc",
"generated_lab.fdb_latexmk", "generated_lab.fls", "generated_lab.synctex.gz"]
        remove_files(temp_files)

        messagebox.showinfo("Успех",  f"Лабораторная  работа  сохранена  как
{save_path}")


    # Функция для удаления файлов, если они существуют
    def remove_files(file_list):
        for file_name in file_list:
            if os.path.exists(file_name):
                os.remove(file_name)

    # Функция для очистки текста от проблемных символов
    def clean_text(text, problem_characters):
        for key, value in problem_characters.items():
            text = text.replace(key, value)
        return text


    # Функция для вставки кода LaTex с выбранным изображением
    def insert_image(target_widget):
```

```
            # Открываем диалоговое окно для выбора файла изображения
            filepath = filedialog.askopenfilename(
                title="Выберите изображение",
                filetypes=[("PNG",   "*.png"),   ("JPEG",   "*.jpg"),   ("JPEG",
"*.jpeg"), ("GIF", "*.gif"), ("BMP", "*.bmp"), ("TIFF", "*.tiff")]
            )

            if not filepath:
                return  # Если файл не выбран, просто возвращаемся

            # Форматируем путь для LaTex (заменяем обратные слеши на прямые)
            filepath = filepath.replace("\\", "/")

            # Текст с LaTex-кодом для вставки изображения
            code = f"""
        \\begin{{figure}}[htbp]
            \centering
            \includegraphics[width=150mm,height=92mm]{{{filepath}}}
            \caption{{Подпись вашего изображения}}
        \end{{figure}}
        """

            # Вставляем LaTex-код в целевой виджет
            target_widget.insert(tk.INSERT, code)

        # Функция для вставки листинга кода
        def insert_listing(target_widget):
            code = """
            \\begin{lstlisting}[caption=Подпись вашего листинга]
        #include <stdio.h>

        int main() /* Пример вашего кода" */
        {
          printf("Hello world!");
          return 0;
        }
        \end{lstlisting}
        """

            # Вставляем текст на текущую позицию курсора в target_widget
            target_widget.insert(tk.INSERT, code)  # tk.INSERT определяет позицию
курсора

        def create_lab_gui():
            # Функция для генерации лабораторной работы
            def on_create_button_click():
                subject = subject_entry.get()
                lab_topic = lab_topic_entry.get()
                lab_num = lab_num_entry.get()
                group_num = group_num_entry.get()
                degree = degree_entry.get()
                student_name = student_name_entry.get()
                professor_name = professor_name_entry.get()
                aims = aims_text.get("1.0", tk.END).strip()
```

```
            theory = theory_text.get("1.0", tk.END).strip()
            result = result_text.get("1.0", tk.END).strip()
            appendix_num = appendix_num_entry.get()
            appendix_title = appendix_title_entry.get()
            appendix = appendix_text.get("1.0", tk.END).strip()

            if not all([subject, lab_topic, lab_num, group_num, student_name,
professor_name, aims, theory, result, appendix_num, appendix_title, appendix]):
                messagebox.showwarning("Ошибка", "Пожалуйста, заполните все
обязательные поля.")
                return

            generate_lab(subject, lab_topic, lab_num, group_num, degree,
student_name, professor_name, aims, theory, result, appendix_num,
appendix_title, appendix)

        # Создаем основное окно Tkinter и устанавливаем его размер
        new_window = tk.Toplevel()
        new_window.title("Создание лабораторной работы")
        new_window.geometry("900x900")  # Устанавливаем размер окна

        # Размер шрифта для меток и полей ввода
        font = ("Arial", 12)

        # Метки и поля ввода для необходимых аргументов
        tk.Label(new_window, text="Предмет:", font=font).grid(row=0,
column=0, sticky='w')
        subject_entry = tk.Entry(new_window, font=font, width=90)  #
Увеличиваем ширину
        subject_entry.grid(row=0, column=1, sticky='w')

        tk.Label(new_window, text="Тема лабораторной работы:",
font=font).grid(row=1, column=0, sticky='w')
        lab_topic_entry = tk.Entry(new_window, font=font, width=90)  #
Увеличиваем ширину
        lab_topic_entry.grid(row=1, column=1, sticky='w')

        tk.Label(new_window, text="Номер лабораторной работы:",
font=font).grid(row=2, column=0, sticky='w')
        lab_num_entry = tk.Entry(new_window, font=font, width=90)  #
Увеличиваем ширину
        lab_num_entry.grid(row=2, column=1, sticky='w')

        tk.Label(new_window, text="Номер группы:", font=font).grid(row=3,
column=0, sticky='w')
        group_num_entry = tk.Entry(new_window, font=font, width=90)  #
Увеличиваем ширину
        group_num_entry.grid(row=3, column=1, sticky='w')

        tk.Label(new_window, text="Имя студента:", font=font).grid(row=4,
column=0, sticky='w')
        student_name_entry = tk.Entry(new_window, font=font, width=90)  #
Увеличиваем ширину
        student_name_entry.grid(row=4, column=1, sticky='w')
```

```
        tk.Label(new_window, text="Имя профессора:", font=font).grid(row=5,
column=0, sticky='w')
        professor_name_entry = tk.Entry(new_window, font=font, width=90)  #
Увеличиваем ширину
        professor_name_entry.grid(row=5, column=1, sticky='w')

        tk.Label(new_window,  text="Степень  профессора  (если  есть):",
font=font).grid(row=6, column=0, sticky='w')
        degree_entry  =  tk.Entry(new_window,  font=font,  width=90)    #
Увеличиваем ширину
        degree_entry.grid(row=6, column=1, sticky='w')

        # Используем Text для больших полей ввода, увеличивая и ширину, и
высоту
        tk.Label(new_window,  text="Цель  работы:",  font=font).grid(row=7,
column=0, sticky='w')
        aims_text = tk.Text(new_window, font=font, height=8, width=90)  #
Увеличиваем и высоту, и ширину
        aims_text.grid(row=7, column=1, sticky='w')

        tk.Label(new_window,         text="Теоретические         сведения:",
font=font).grid(row=8, column=0, sticky='w')
        theory_text = tk.Text(new_window, font=font, height=8, width=90)  #
Увеличиваем и высоту, и ширину
        theory_text.grid(row=8, column=1, sticky='w')

        tk.Label(new_window,   text="Результат   выполнения   программы:",
font=font).grid(row=9, column=0, sticky='w')
        result_text = tk.Text(new_window, font=font, height=8, width=90)  #
Увеличиваем и высоту, и ширину
        result_text.grid(row=9, column=1, sticky='w')

        tk.Label(new_window,            text="Номер            приложения:",
font=font).grid(row=10, column=0, sticky='w')
        appendix_num_entry = tk.Entry(new_window, font=font, width=90)  #
Увеличиваем ширину
        appendix_num_entry.grid(row=10, column=1, sticky='w')

        tk.Label(new_window,           text="Название           приложения:",
font=font).grid(row=11, column=0, sticky='w')
        appendix_title_entry = tk.Entry(new_window, font=font, width=90)  #
Увеличиваем ширину
        appendix_title_entry.grid(row=11, column=1, sticky='w')

        tk.Label(new_window,          text="Содержание          приложения:",
font=font).grid(row=12, column=0, sticky='w')
        appendix_text = tk.Text(new_window, font=font, height=8, width=90)  #
Увеличиваем и высоту, и ширину
        appendix_text.grid(row=12, column=1, sticky='w')

        # Кнопка, которая вызывает функцию `generate_lab`
        generate_button = tk.Button(new_window, text="Создать", font=font,
command=on_create_button_click)
```

```
        generate_button.grid(row=13, column=1, sticky='e')

        # Указывает на активный элемент text
        current_focus = None

        # Создаем контекстное меню
        context_menu = tk.Menu(new_window, tearoff=0)  # tearoff=0 означает,
что меню нельзя "открепить"
        context_menu.add_command(
            label="Вставить изображение",
            command=lambda: insert_image(current_focus)  # Вставка текста в
активное поле
        )
        context_menu.add_command(
            label="Вставить листинг",
            command=lambda: insert_listing(current_focus)  # Вставка текста
в активное поле
        )

        # Функция для отображения контекстного меню при нажатии правой кнопкой
мыши
        def on_right_click(event):
            if current_focus is not None:
                context_menu.post(event.x_root, event.y_root)

        # Привязываем контекстное меню к нужным элементам
        aims_text.bind("<Button-2>",  on_right_click)    #  Нажатие  правой
кнопкой мыши
        theory_text.bind("<Button-2>",  on_right_click)    #  Нажатие  правой
кнопкой мыши
        result_text.bind("<Button-2>",  on_right_click)    #  Нажатие  правой
кнопкой мыши
        appendix_text.bind("<Button-2>",  on_right_click)   #  Нажатие  правой
кнопкой мыши

        # Функция для переключения состояния фокуса
        def on_focus_in(event):
            nonlocal current_focus
            current_focus = event.widget  # Теперь поле активно

        def on_focus_out(event):
            nonlocal current_focus
            current_focus = None  # Поле больше не активно

        # Привязываем обработчики фокуса к полям
        aims_text.bind("<FocusIn>", on_focus_in)
        aims_text.bind("<FocusOut>", on_focus_out)

        theory_text.bind("<FocusIn>", on_focus_in)
        theory_text.bind("<FocusOut>", on_focus_out)

        result_text.bind("<FocusIn>", on_focus_in)
        result_text.bind("<FocusOut>", on_focus_out)
```

```
        appendix_text.bind("<FocusIn>", on_focus_in)
        appendix_text.bind("<FocusOut>", on_focus_out)

        # Запускаем главный цикл приложения
        return new_window


    # Функция, которая открывает диалоговое окно для выбора файла, затем
создает динамический интерфейс для ввода данных
    def create_random_report_gui():
        # Открываем диалоговое окно для выбора файла .tex
        tex_file_path = filedialog.askopenfilename(
            title="Выберите файл LaTex",
            filetypes=[("LaTex Files", "*.tex"), ("Все файлы", "*.*")]
        )

        if not tex_file_path:
            messagebox.showwarning("Ошибка", "Файл не был выбран.")
            return

        # Читаем содержимое файла .tex
        with open(tex_file_path, 'r') as tex_file:
            tex_content = tex_file.read()

        # Ищем все конструкции с двойными фигурными скобками
        placeholders = re.findall(r'{{(.+?)}}', tex_content)

        # Удаляем пробелы в начале и в конце строк
        placeholders = [p.strip() for p in placeholders]

        # Создаем словарь с пустыми значениями для найденных placeholders
        placeholder_dict = {placeholder: "" for placeholder in placeholders}

        # Создаем новое окно Tkinter для ввода значений в placeholders
        new_window = tk.Toplevel()  # Создаем новое окно
        new_window.title("Заполнение значений")

        # Создаем метки и поля ввода для каждого placeholder
        entry_dict = {}  # Словарь для хранения виджетов ввода

        for idx, placeholder in enumerate(placeholders):
            label = tk.Label(new_window, text=f"Введите значение для {{
{placeholder} }}:")
            label.grid(row=idx, column=0, sticky='w')

            text_entry = tk.Text(new_window, height=2, width=50)
            text_entry.grid(row=idx, column=1, sticky='w')

            # Сохраняем виджет в словаре для последующей обработки
            entry_dict[placeholder] = text_entry

        # Функция, которая сохраняет введенные значения в placeholder_dict
        def save_values():
            for placeholder, text_entry in entry_dict.items():
```

```
                value = text_entry.get("1.0", tk.END).strip()  # Получаем
введенное значение
                placeholder_dict[placeholder] = value

            generate_random_report(tex_file_path, placeholder_dict)
            messagebox.showinfo("Сохранено", "Значения сохранены!")

        # Кнопка для сохранения значений
        save_button = tk.Button(
            new_window,
            text="Сохранить",
            command=save_values
        )
        save_button.grid(row=len(placeholders), column=1, sticky='e')

    # Создаем главное окно Tkinter
    root = tk.Tk()
    root.title("Создание отчетов")

    # Размер окна
    root.geometry("300x200")

    # Кнопка для создания отчета по лабораторной работе
    lab_report_button = tk.Button(
        root,
        text="Создать отчет по лабораторной работе",
        command=create_lab_gui
    )
    lab_report_button.pack(pady=20)  # Устанавливаем вертикальный отступ

    # Кнопка для создания отчета по своему шаблону
    custom_report_button = tk.Button(
        root,
        text="Создать отчет по своему шаблону",
        command=create_random_report_gui
    )
    custom_report_button.pack(pady=20)  # Устанавливаем вертикальный отступ

    # Запускаем главный цикл Tkinter
    root.mainloop()
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Функциональная схема алгоритма

ПРИЛОЖЕНИЕ В
(обязательное)
Блок-схема алгоритма

ПРИЛОЖЕНИЕ Г
(обязательное)
Графический интерфейс пользователя

ПРИЛОЖЕНИЕ Е
(обязательное)
Ведомость документов