

**Оглавление**

[Введение 4](#_Toc108261101)

[1. Анализ заданий курсовой работы 5](#_Toc108261102)

[1.1 Исходные данные к заданиям курсовой работы 5](#_Toc108261103)

[1.2 Анализ методических указаний, входных данных к заданиям курсовой работы 11](#_Toc108261104)

[1.3 Выводы по главе 1 12](#_Toc108261105)

[2. Разработка программного продукта для решения прикладных задач 13](#_Toc108261106)

[2.1 Работа с набором данных 13](#_Toc108261107)

[2.1.1 Построение алгоритма решения задания без графического интерфейса 14](#_Toc108261108)

[2.2 Разработка экспертной системы 15](#_Toc108261109)

[2.2.1 Построение алгоритма решения задания с графическим интерфейсом 15](#_Toc108261110)

[2.2.2 Разработка программной реализации на языке программирования 16](#_Toc108261111)

[2.3 Разработка аналитической системы 18](#_Toc108261112)

[2.3.1 Построение алгоритма решения задания с графическим интерфейсом 18](#_Toc108261113)

[2.3.2 Разработка программной реализации на языке программирования с использованием дополнительных библиотек 19](#_Toc108261114)

[2.4 Разработка аналитической системы 21](#_Toc108261115)

[2.4.1 Построение алгоритма решения задания с графическим интерфейсом 21](#_Toc108261116)

[2.4.2 Разработка программной реализации на языке программирования с использованием дополнительных библиотек 22](#_Toc108261117)

[2.5 Тестирование и отладка 25](#_Toc108261118)

[2.6 Выводы по главе 2 26](#_Toc108261119)

[3. Выводы 27](#_Toc108261120)

[4. Список используемой литературы 28](#_Toc108261121)

[Приложение 1. 29](#_Toc108261122)

[Приложение 2. 30](#_Toc108261123)

[Приложение 3. 37](#_Toc108261124)

[Приложение 4. 47](#_Toc108261125)

Введение

В условиях быстро развивающейся современной высокотехнологичной

экономики актуальной задачей является разработка различных автоматизированных систем. Они позволяют наиболее эффективно использовать различные современные средства информационной вычислительной техники и решать прикладные задачи. Для успешного решения таких задач нужно уметь использовать высокоуровневые средства разработки, декомпозировать задачу, осуществлять тестирование и сопровождение программных продуктов.

В данной работе решаются задачи высокоуровневого программирования, согласно индивидуальному заданию.

Актуальность курсовой работы связана с наличием большого количества прикладных задач, которые необходимо эффективно решать с помощью инструментов высокоуровневых методов программирования.

1. Анализ заданий курсовой работы

## 1.1 Исходные данные к заданиям курсовой работы

Курсовая работа состоит из 4 заданий.

**Задание 1 Работа с наборами данных БЕЗ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА (GUI) :**

Формулировка:

Во внешнем файле resourse\_1.txt дан текст. Выведите все слова, встречающиеся в тексте, по одному на каждую строку, через пробел укажите количество повторений. Слова должны быть отсортированы по убыванию их количества появления в тексте, а при одинаковой частоте появления — в лексикографическом порядке. Вывод должен осуществляться в текстовый файл result\_1.txt. При необходимости можно продублировать вывод в консоль.

Методические указания:

После того, как вы создадите словарь всех слов, необходимо отсортировать его по частоте встречаемости слова. Желаемого можно добиться, если создать список, элементами которого будут кортежи из двух элементов: частота встречаемости слова и само слово. Например, [(2, 'hi'), (1, 'what'), (3, 'is')]. Тогда стандартная сортировка будет сортировать список кортежей, при этом кортежи сравниваются по первому элементу, а если они равны — то по второму. Знаки препинания не должны учитываться. Программу сохранить под именем exercise\_1.py

Входные данные:

Преподаватель вводит текст в текстовый файл resourse\_1.txt (5-6 абзацев) и сохраняет его. hi hi what is your name my name is bond james bond my name is damme van damme claude van damme jean claude van damme

Выходные данные:

Выведите ответ на задание № 1 в текстовый файл result\_1.txt. Damme 4 Is 3 Name 3 13 Van 3 Bond 2 claude 2 hi 2 my 2 james 1 jean 1 what 1

**Задание 2 Разработка экспертной системы С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ (GUI):**

Формулировка:

Некоторый банк хочет внедрить систему управления счетами клиентов, поддерживающую следующие операции: 1. Пополнение счета клиента. 2. Снятие денег со счета. 3. Запрос остатка средств на счете. 4. Перевод денег между счетами клиентов. 5. Начисление процентов всем клиентам.

Методические указания:

Необходимо реализовать такую систему. Первоначально у банка 1 клиент. Клиент(ы) банка идентифицируются именами (уникальная строка, не содержащая пробелов). Вам необходимо задать в качестве имени клиента – свою фамилию на английском языке с большой буквы. На вашу фамилию должен быть открыт счет с суммой равной вашему ID. Ivanov 70121903 В отдельном поле должна быть предусмотрена возможность ввода простых команд, которые поддерживают следующие операции:

* *DEPOSIT name sum:* Зачислить сумму sum на счет клиента name. Если клиента нет, то он создается и на него за
* *WITHDRAW name sum:* Снять сумму sum со счета клиента name. Если клиента, то счет создается. Баланс при выполнении такой операции у вновь созданного клиен
* *BALANCE name:* Узнать остаток средств на счету клиента name. Для каждого запроса BALANCE программа должна вывести остаток на счету данного клиента. Если же у клиента с запрашиваемым именем не открыт счет в банке, выводится сообщение «NO CLIENT». Если пользователь не указал имя клиента – то выводится баланс всех существующих клиентов.
* *TRANSFER name1 name2 sum:* Перевести сумму sum со счета клиента name1 на счет клиента name2. Если у какого-либо клиента, то он заводится в системе и ему создается счет с переведенной суммой.
* *INCOME p:* Начислить всем клиентам, у которых открыты счета, p% от суммы счета. Проценты начисляются только клиентам с положительным остатком на счету, если у клиента остаток отрицательный, то его счет не меняется. После начисления процентов сумма на счету остается целой, то есть начисляется только целое число денежных единиц. Дробная часть начисленных процентов отбрасывается.

ПРОГРАММА ДОЛЖНА ОБРАБАТЫВАТЬ ТЕКСТОВЫЕ КОМАНДЫ ИЗ ЛЕВОГО ПОЛЯ ТОЛЬКО ПОСЛЕ НАЖАТИЯ КНОПКИ «Calculate». То есть, пользователь СНАЧАЛА вводит желаемые команды, при этом каждая новая команда вводится с новой строки, а ПОТОМ нажимает на кнопку «Calculate». Результат должен быть выведен в поле справа. Количество команд, которые может ввести пользователь за один раз – не более 20. При необходимости следует предусмотреть прокрутку в поле. Пользователю допускается вводить «пустые строки» - несколько раз нажимать на кнопку «enter». При нажатии клавиши ввода «enter», фокус не должен переходить на кнопку «Calculate». Формат и внешний вид окна определяет разработчик. Для «очистки» левого и правого полей можно предусмотреть кнопку «Clear», при этом данные о ранее введенных клиентах не должны быть потеряны. ВАЖНО: 1. Команды вводятся пользователем только большими буквами. Сама команда, имя клиента, суммы (числа) разделяются пробелами. 2. Предполагается, что пользователь такой системы грамотный и команды с аргументами вводит без ошибок в рамках их вышесформулированного синтаксиса. 3. Как только для несуществующего ранее клиента проводится операция пополнения (DEPOSIT), снятия (WITHDRAW) или перевода денег (TRANSFER), он вносится в систему, ему заводится счет с указанным балансом. Все дальнейшие операции проводятся только с этим счетом. Сумма на счету может быть как положительной, так и отрицательной, при этом всегда является целым числом. Программу сохранить под именем exercise\_2.py

**Задание 3 Разработка аналитической системы С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ (GUI):**

Методические указания:

Необходимо разработать программу и GUI для реализации стандартных функций калькулятора. Примерные вид внешнего интерфейса – стандартный. Функционал «инженерного» режима работы калькулятора добавляется при нажатии на кнопку перехода в расширенный режим, либо выбора соответствующего пункта меню. Расширенный режим работы определяется внешним видом «цифрового дисплея», количеством ячеек памяти (кнопки M+, M-, MS, MR, MS) и кнопками, отвечающими за дополнительные функции. Количество строк «цифрового дисплея» должно определяться как последовательная сумма всех цифр ID студента. Суммирование отдельных цифр числа должно осуществляться до получения однозначного числа, состоящей из 1-й цифры. Количество ячеек памяти калькулятора должно определяться как последовательная сумма последних 3-х чисел ID. Суммирование отдельных цифр числа должно осуществляться до получения однозначного числа, состоящей из 1-й цифры. Дополнительные функции расширенного режима: Dms, 10^x, Pi, tanh, Ln переводит из десятичного вида в формат в градусы, минуты, секунды; возведение десяти в произвольную степень, число Пи, гиперболический тангенс, натуральный логарифм.

Входные данные:

Преподаватель с помощью мыши вводит числа и выполняет арифметические операции. Преподаватель с помощью мыши вводит числа и выбирает заявленные дополнительные функции. Преподаватель вводит намеренно некорректные операции (попытка деления на 0) 3 Для входных и выходных данных в методических материалах информация, аналогичная первому заданию – это было сочтено опечаткой и заменено на подходящее по логическому смыслу. Оригинальный текст в методических материалах.

Выходные данные:

На цифровом дисплее должен отображаться результат, либо сообщение об ошибке при некорректной операции. Проверяется количество строк в «цифровом дисплее» калькулятора и количество ячеек памяти.

**Задача 4 Задача о Ханойских башнях С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ (GUI):**

Методические указания:

Основная часть работы над задачей студента сводится к правильной визуализации полученных результатов, а также оптимальному поиску промежуточных результатов. Схема ханойских башен должна выглядеть примерно таким образом, как указано на рисунке 1:

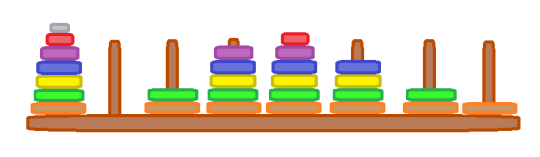


Рисунок 1 – Примерная схема решения задачи

Поскольку диски близких размеров будут отличаться всего на один пиксель, то для контроля на каждом диске необходимо проставить его диамер в виде цифры. Так как в конце задачи все диски будут находиться на первом шпинделе, а общее количество дисков теоретически может быть равно 72, то рекомендуется сделать толщину одного диска примерно равной 10-12 пикселям, для указанного окна. Входные данные: Идентификатор студента. Промежуточные проценты, вводимые преподавателем в соответсвующие поля над кнопками. Выходные данные: На цифровом дисплее должно отображаться окно с начальным расположением дисков на шпинделях Ханойских башень. Шпиндели пронумерованы, на дисках также обозначены соответствующие диаметры. Под ней отображается шесть кнопок и четыре поля для ввода цифр. В нижней части экрана демонстрируется надпись «Итерация 0».

## 1.2 Анализ методических указаний, входных данных к заданиям курсовой работы

Поскольку в методических указаниях к выполнению курсовой работы указано использование языка программирования Python, среды разработки PyCharm, а также стандартных библиотек и модулей стандартной библиотеки, то для решения заданий курсовой работы были выбраны следующие средства:

Язык программирования Python3.9 – согласно рекомендациям;

Среда разработки PyCharm – согласно рекомендациям;

Графическая библиотека Tkinter – наиболее простая библиотека, к тому же, изученная ранее в рамках курса.

Встроенная библиотека math – для выполнения математических операций.

## 1.3 Выводы по главе 1

Изучив полученный материал из главы 1, можно сделать вывод, что задания включают наиболее часто используемые прикладные задачи высокоуровневого программирования – это и работа с файлами, использование структур данных и алгоритмов, построение приложений с графическим пользовательским интерфейсом.

Успешное выполнение задание будет свидетельствовать о полноте усвоенного материала в рамках дисциплины «Методы высокоуровневого программирования» и умение решать задачи низкой и средней сложности.

Задания выстроены от простого к сложному и если на выполнение первых двух заданий уйдет пара часов, то на два последний потребуется гораздо больше сил и времени, необходимо будет разработать архитектуру приложения, а в задаче с ханойскими башнями, найти оптимальный по производительности и количеству итераций алгоритм решения.

1. Разработка программного продукта для решения прикладных задач

## 2.1 Работа с набором данных

Для чтения и записи файлов будет использоваться менеджер контекста with. Для подсчета и хранения уникальных слов будет использоваться словарь, где ключом будет слово, а значением - количество упоминаний этого слова.

## 2.1.1 Построение алгоритма решения задания без графического интерфейса

В целом, данная задача не представляет особого интереса и решается в несколько строчек кода. Блок схема алгоритма изображена на рисунке 2.



Рисунок 2 – Блок схема алгоритма задания 1

Листинг программного кода с комментариями приведен в

приложении 1.

## 2.2 Разработка экспертной системы

## 2.2.1 Построение алгоритма решения задания с графическим интерфейсом

Оптимальным алгоритмом для решения задачи кажется следующая

последовательность действий:

* нажатие кнопки «Calculate»;
* чтение данных из поля ввода;
* разбиение на команды;
* выполнение команды;
* вывод данных.

Блок схема предложенного алгоритма изображена на рисунке 3

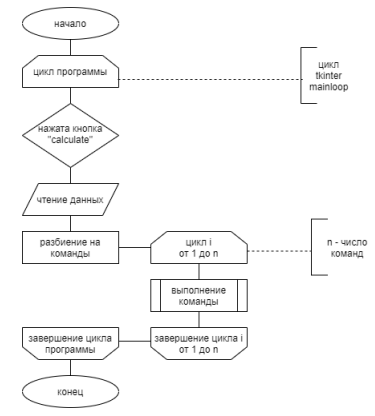


Рисунок 3 – блок схема алгоритма для задания 2

## 2.2.2 Разработка программной реализации на языке программирования

Исходный код этой задачи стоило бы разделить на несколько файлов, чтобы не нагромождать все в одном, но в требованиях к курсовой работе было указано, что код должен находиться в одном файле, так и сделаем.

Графический интерфейс был реализован средствами библиотеки Tkinter. Для выполнения поставленной задачи были спроектированы и реализованы вспомогательные сущности – классы, а именно:

* Класс Client для хранения информации о клиенте и инкапсуляции его инварианта
* Статический класс AccountsManager для управления счетами клиентов и действий с ними

Когда пользователь вводит команды и наживает calculate, происходит валидация входных данных и удаление лишних пробелов. В случае ошибок, информация о них будет выведена в поле для вывода (справа).

Поскольку в ТЗ было четко прописано что: "Предполагается, что пользователь такой системы грамотный и команды с аргументами вводит без ошибок в рамках их вышесформулированного синтаксиса." Дополнительных проверок на существование команд и соответствие аргументов реализовано не будет. Но исключения все еще будут обрабатываться, чтобы программа не «падала»

Так как данная программа работает с банковскими счетами клиентов, то было решено сделать простенькую систему транзакций. Так вся последовательность введенных пользователем команд принимается за одну транзакцию и если при выполнении одной из команд возникнет ошибка, то изменения, внесенные другими командами, не будут закреплены, то есть состояние счетов клиентов не изменится. С помощью команды TEST\_FAIL можно проверить работу этой системы.

Исходя из вышесказанного, было решено реализовать в классе AccountsManager методы, по названию соответствующие допустимым операциям и вызывать эти методы по имени прямо из поля для ввода.

В обозначенных в ТЗ условиях, такой подход видится наиболее простым, хотя в каком-то серьезном проекте так делать не стоит.

Вся бизнес-логика находится в классе AccountsManager, а взаимодействия с GUI обрабатываются обычными функциями внутри модуля. При этом обработчик нажатий на кнопку calculate напрямую взаимодействует с AccountsManager, что некорректно с точки зрения модели MVC, но в данной ситуации, оверинженеринг в виде разделения приложения на слои не имеет смысла.

Внешний вид рабочего приложения продемонстрирован на рисунке 4

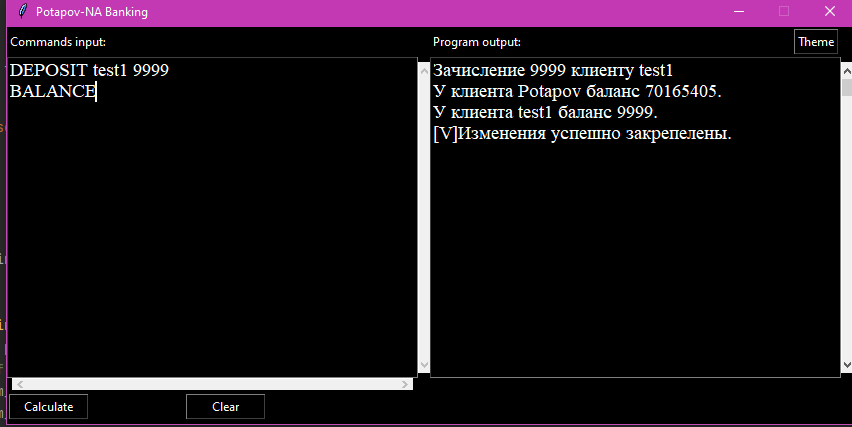


Рисунок 4. Внешний вид приложения.

Листинг программного кода с комментариями приведен в

приложении 2

## 2.3 Разработка аналитической системы

## 2.3.1 Построение алгоритма решения задания с графическим интерфейсом

Алгоритм задачи сводится к вводу от пользователя и обработки этих команд внутри программы.

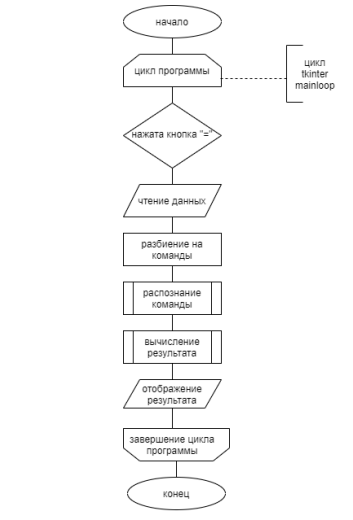


Рисунок 5 – Блок схема алгоритма аналитической системы

## 2.3.2 Разработка программной реализации на языке программирования с использованием дополнительных библиотек

Для наиболее простой (как казалось по началу) реализации математических операций было решено использовать команду eval, в которую передается выражение в виде строки и возвращается результат, как будто это команда была введена напрямую в интерпретатор python.

Реализовав калькулятор таким образом я понял, что лучшим решением было бы использование отдельного класса с методами для разных математических операций и уже внутри этих методов, по необходимости, использовать функцию eval, но менять готовую реализацию я уже не стал.

Для реализации дополнительных функций, таких как вычисление синуса, косинуса и тангенса, была использована библиотека math.

Для управления ячейками памяти был создан статический класс Memory.

Листинг исходного кода с подробными комментариями приведен в приложении 3.

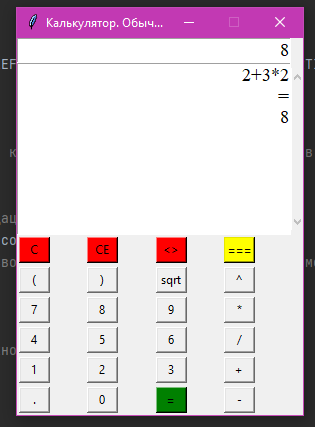


Рисунок 6. Внешний вид калькулятора в обычном режиме.

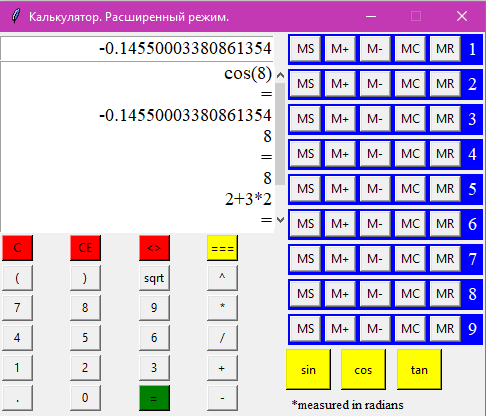


Рисунок 7. Внешний вид калькулятора в расширенном режиме.

## 2.4 Разработка аналитической системы

## 2.4.1 Построение алгоритма решения задания с графическим интерфейсом

Решение задачи ханойских башен с 8 шпинделями было решено реализовать с помощью рекурсии. При этом использовался стандартный алгоритм для ханойских башен с 3мя шпинделями, который повторялся много раз для разных групп шпинделей. Сам алгоритм для ханойских башен с 3мя шпинделями приведен на рисунке 8.

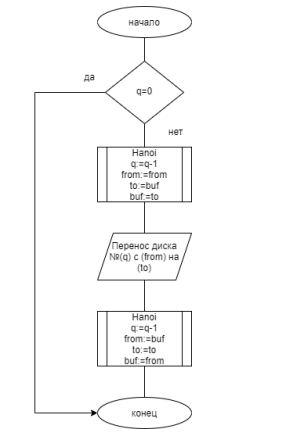


Рисунок 8. Алгоритм для ханойских башен с 3мя шпинделями.

## 2.4.2 Разработка программной реализации на языке программирования с использованием дополнительных библиотек

Пожалуй, это задача заняла больше всего времени для ее программной реализации. Изначальная версия алгоритма работала очень долго и подразумевала миллионы итераций. Плюс ко всему, каждую итерацию было решено сохранять в памяти для возможности дальнейшей визуализации положения дисков на любой итерации без необходимости прерасчета алгоритма. Таким образом, получилось решить задачу за 10919 итераций, каждая их которых сохранена в памяти и может быть мгновенно визуализирована. Расчет алгоритма и сохранение в памяти происходит при запуске программы. На моем компьютере это занимает менее 1 секунды.



Рисунок 9. Окно программы при запуске.

В окне программы предусмотрены все кнопки согласно заданию, а так же добавлены две дополнительные кнопки со стрелочками для перемещения между соседними итерациями.



Рисунок 10. Окно программы на последней итерации алгоритма.

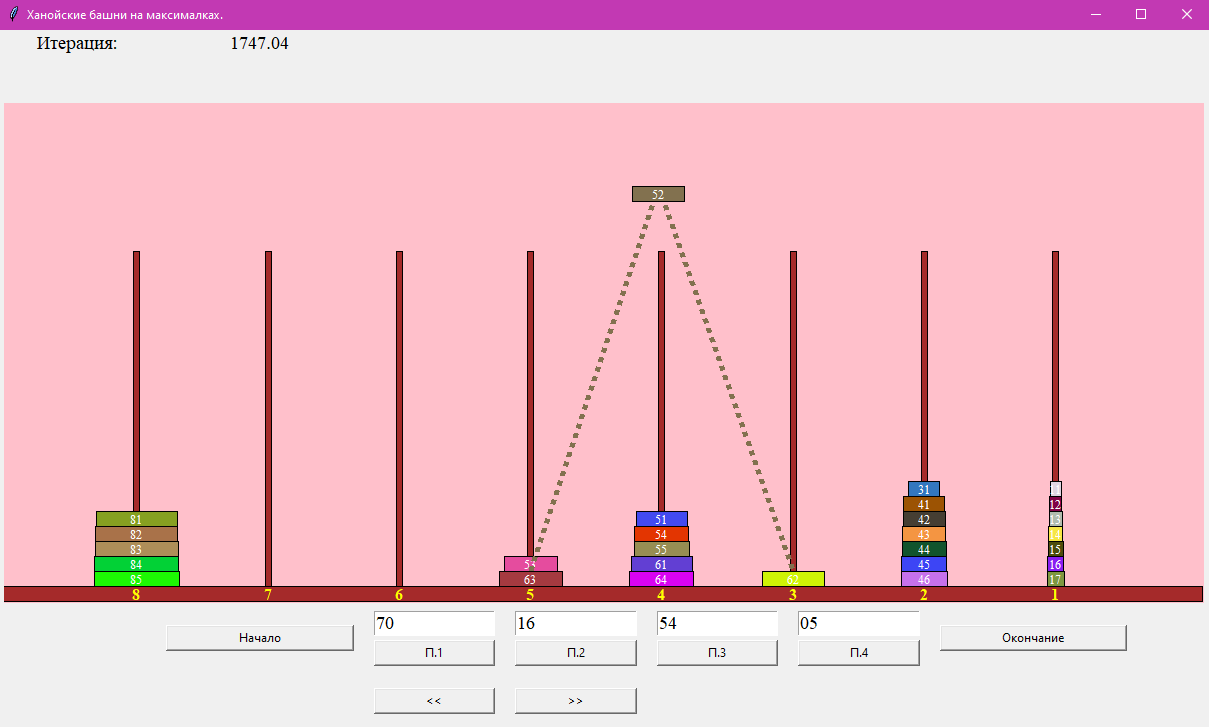


Рисунок 11. Окно программы на промежуточной итерации, соответствующей 16% от общего числа итераций.

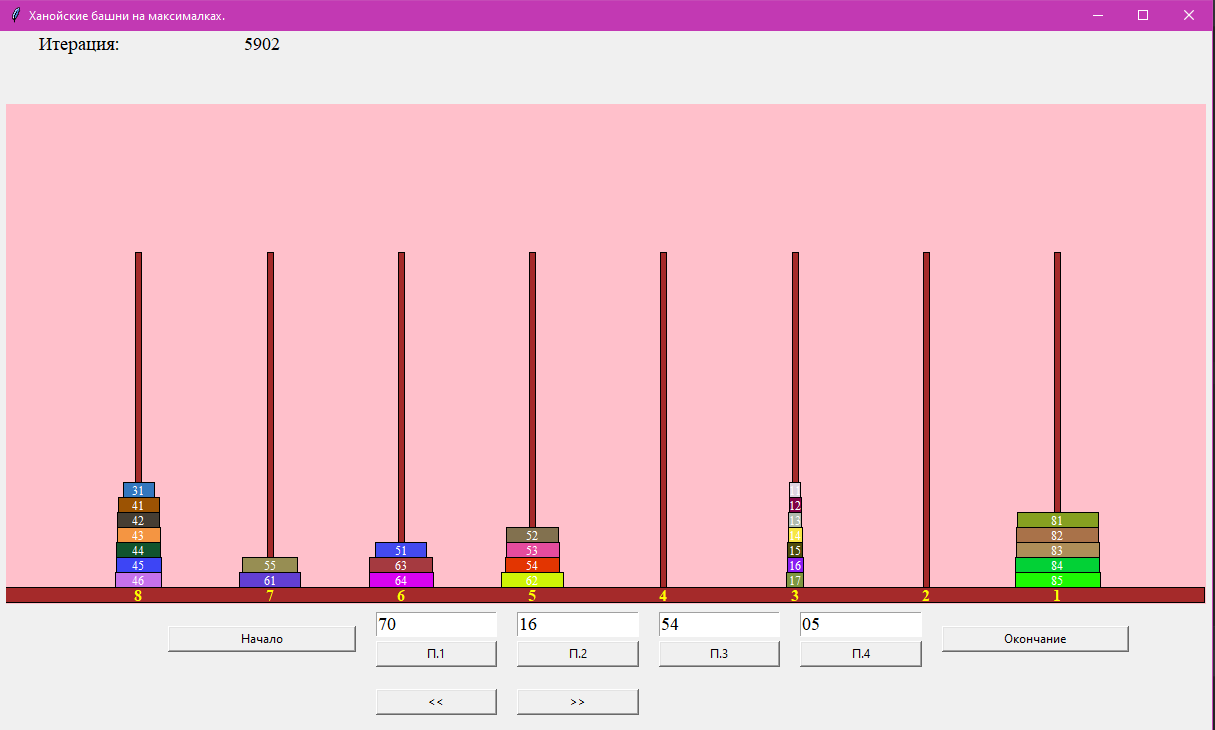


Рисунок 12. Окно программы на случайно итерации, выбранной через кнопки со стрелочками.

Листинг исходного кода с подробными комментариями приведен в приложении 4.

## 2.5 Тестирование и отладка

Тестирование всех алгоритмов было проведено в ручном режиме.

Для отладки алгоритма с ханойскими башнями был использован встроенный profiler pyCharm. Так с помощью полученной информации из профайлера скорость работы алгоритма была сведена к минимуму.

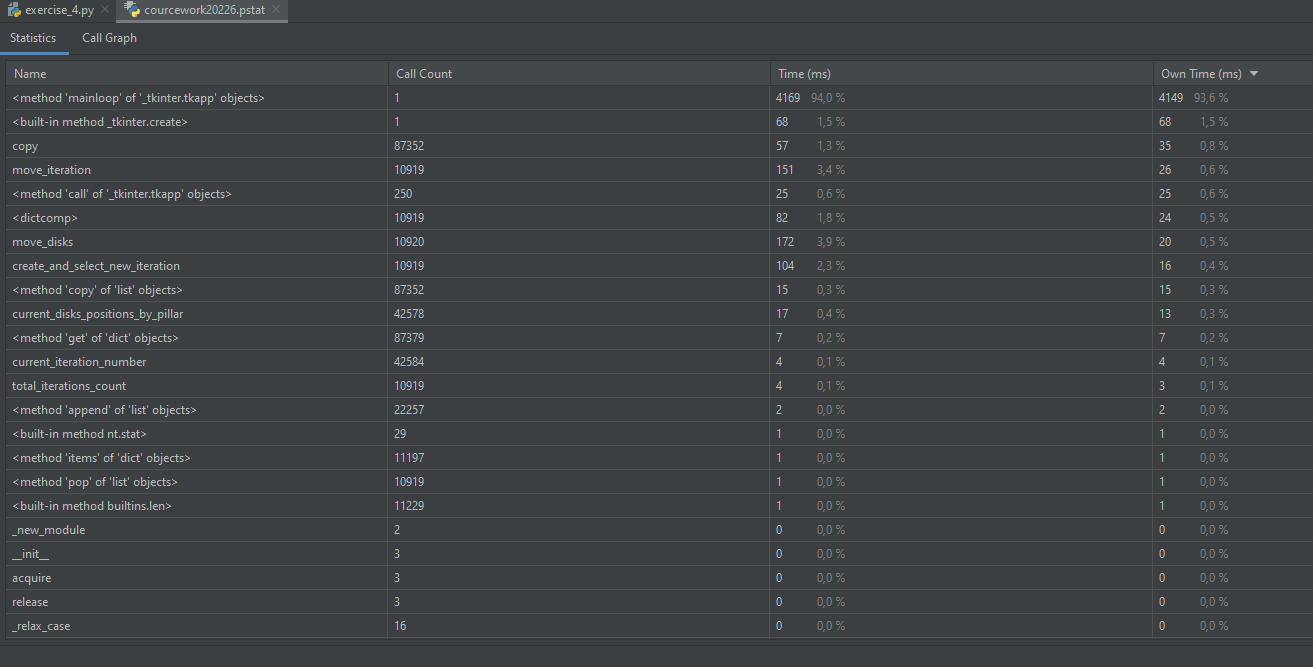


Рисунок 13. Статистика по загрузке итоговой версии программы.

## 2.6 Выводы по главе 2

Язык Python обладает обширной стандартной библиотекой, которая помогает разработчику программного продукта решать прикладные задачи разного уровня сложности и направленности.

Богатая инфраструктура, сложившаяся вокруг языка Python, позволяет разработчику выбирать из множества инструментов, к примеру, среда разработки PyCharm предлагает богатые возможности для отладки программного обеспечения.

1. Выводы

Для решения задач, даже содержащих небольшие алгоритмы, необходимо проектировать и декомпозировать программный код, выделять основные сущности и строить понятную структуру проекта. При таком подходе исходный код программы будет читаемым, а его поддержка на поздний стадиях разработки будет проще, чем если бы код был написан в процедурном стиле.

Хоть язык python и находится где-то между ООП и функциональным языком программирования, но я отдаю предпочтение именно ООП, хоть и в данной курсовой задаче весь потенциал такого подхода не раскрывается.

Богатая коллекция встроенных библиотек языка python позволяет реализовывать рутинные операции без необходимости скачивать сторонние библиотеки или реализовывать стандартные функции самостоятельно.

Получившиеся в результате разработки программы успешно выполняются интерпретатором языка Python версии 3.9

1. Список используемой литературы
2. Основы программирования на языке Python: учебное пособие Автор: Буйначев С. К. , Боклаг Н. Ю.
3. Язык программирования высокого уровня Python : функции, структуры данных, дополнительные модули: учебное пособие Автор: Шелудько В. М.
4. (Tkinter. Программирование GUI на Python [Электронный ресурс]. URL: https://younglinux.info/tkinter/tkinter (дата обращения: 28.11.2021)
5. (Questions tagged [python] [Электронный ресурс]. URL: https://stackoverflow.com/questions/tagged/python (дата обращения: 28.11.2021)
6. (Глава 9. Создание графического интерфейса [Электронный ресурс]. URL: https://metanit.com/python/tutorial/9.9.php (дата обращения: 27.11.2021)
7. (Python Tkinter Entry – How to use [Электронный ресурс]. URL: https://pythonguides.com/python-tkinter-entry/ (дата обращения: 27.11.2021)

## Приложение 1.

import re  
  
# Объявляем константы  
INPUT\_FILE\_NAME = 'resource\_1.txt'  
OUTPUT\_FILE\_NAME = 'result\_1.txt'  
PUNCTUATION\_MARKS\_RE = re.compile('[!#$%&\"\'\*+-.^\_`|~:()\[\]{}]')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Читаем входные данные из файла  
 with open(INPUT\_FILE\_NAME, 'r', encoding='UTF-8') as \_file:  
 \_input\_data = \_file.read().strip()  
  
 # Удаляем пунктуационные символы чтобы оставить только "чистые" слова  
 \_input\_data = PUNCTUATION\_MARKS\_RE.sub('', \_input\_data)  
 # Создаем список слов  
 \_input\_data = \_input\_data.split()  
  
 # Записываем уникальные слова в словарь и считаем их количество  
 \_count\_by\_word = {}  
  
 for \_word in \_input\_data:  
 if \_word in \_count\_by\_word:  
 \_count\_by\_word[\_word] += 1  
 else:  
 \_count\_by\_word[\_word] = 1  
  
 # Получаем из словаря список кортежей (ключ, значение) и сортируем этот список по ключам, которые лежат в кортежах  
 \_sorted\_tuples = sorted(\_count\_by\_word.items(), key=lambda x: x[0])  
 # Еще раз сортируем список, но в этот раз по значениям, которые лежат в кортежах  
 \_sorted\_tuples = sorted(\_sorted\_tuples, key=lambda x: x[1], reverse=True)  
 # Формируем список из кортежей, преобразованых в строку  
 \_output\_data = [f'{x} {y}' for x, y in \_sorted\_tuples]  
 # Превращаем список в строку  
 \_output\_data = '\n'.join(\_output\_data)  
 print(\_output\_data)  
 # Пишем в файл  
 with open(OUTPUT\_FILE\_NAME, 'w', encoding='UTF-8') as \_file:  
 \_file.write(\_output\_data)

## Приложение 2.

from enum import Enum  
from tkinter import \*  
import os  
import pickle  
from copy import copy  
  
CLIENTS\_DATABASE\_FILE\_NAME = 'clients.data'  
  
  
# Коласс для хранения информации о клиентах  
class Client:  
 def \_\_init\_\_(self, \_name: str, \_balance: int):  
 self.\_\_name = \_name  
 self.\_\_balance = \_balance  
  
 @property  
 def name(self):  
 return self.\_\_name  
  
 @property  
 def balance(self):  
 return self.\_\_balance  
  
 def increase\_balance(self, \_value: int):  
 exception\_if\_invalid\_amount(\_value)  
 self.\_\_balance += \_value  
  
 def decrease\_balance(self, \_value: int):  
 exception\_if\_invalid\_amount(\_value)  
 self.\_\_balance -= \_value  
  
  
# Проверка на корректность суммы  
def exception\_if\_invalid\_amount(\_amount):  
 if int(\_amount) < 0:  
 raise ValueError('Сумма должна быть больше 0!')  
  
  
# Статический класс для управления счетами клиентов  
class AccountsManager:  
 # Для хранения зарегистрированных клиентов.  
 \_\_clients\_by\_name = {}  
 # Для хранения бэкапа клиентов до коммита.  
 \_\_clients\_backup = {}  
 use\_database = False  
  
 def \_\_init\_\_(self):  
 raise Exception('This is a static class!')  
  
 # Проверяем, существует ли клиент с таким именем  
 @classmethod  
 def is\_client\_exists(cls, \_name: str):  
 return \_name in cls.\_\_clients\_by\_name  
  
 @classmethod  
 def load\_clients\_from\_database(cls):  
 if cls.use\_database and os.path.isfile(CLIENTS\_DATABASE\_FILE\_NAME):  
 with open(CLIENTS\_DATABASE\_FILE\_NAME, 'rb') as \_file:  
 cls.\_\_clients\_by\_name = pickle.load(\_file)  
  
 @classmethod  
 def \_\_write\_clients\_database(cls):  
 if cls.use\_database:  
 with open(CLIENTS\_DATABASE\_FILE\_NAME, 'wb') as \_file:  
 pickle.dump(cls.\_\_clients\_by\_name, \_file)  
  
 # Закрепляем изменения путем очистки словаря с бэкапом. Откатиться назад больше нельзя.  
 @classmethod  
 def commit\_if\_has\_changes(cls):  
 if cls.\_\_clients\_backup:  
 cls.\_\_clients\_backup = {}  
 print('Committed')  
  
 cls.\_\_write\_clients\_database()  
 return True  
 else:  
 print('No changes to commit')  
 return False  
  
 # Откатываем состояние всех затронутых клиентов  
 @classmethod  
 def rollback(cls):  
 for \_name, \_client in cls.\_\_clients\_backup.items():  
 cls.\_\_clients\_by\_name[\_name] = \_client  
  
 cls.\_\_clients\_backup = {}  
 print('Reverted')  
  
 # Валидируем имя пользователя и создаем для него счет, если еще не создан  
 @classmethod  
 def register\_new\_client(cls, \_name: str, \_balance: int):  
 if ' ' in \_name or '\n' in \_name:  
 return f'Имя пользователя {\_name} не должно содержать пробелы!.'  
  
 elif \_name in cls.\_\_clients\_by\_name:  
 raise Exception(f'Клиент с именем {\_name} уже зарегистрирован!')  
  
 else:  
 \_new\_client = Client(\_name, \_balance)  
 cls.\_\_clients\_by\_name[\_name] = \_new\_client  
 cls.\_\_write\_clients\_database()  
  
 print(f'Клиент {\_name} успешно зарегистрирован.')  
  
 # Пополнение баланса указанного клиента на указанную сумму  
 @classmethod  
 def deposit(cls, \_name: str, \_sum: int):  
 \_sum = int(\_sum)  
 exception\_if\_invalid\_amount(\_sum)  
  
 \_client = cls.\_\_get\_client\_register\_if\_not\_exists(\_name)  
 cls.\_\_backup\_client(\_client)  
 cls.\_\_deposit(\_client, \_sum)  
  
 return f'Зачисление {\_sum} клиенту {\_name}'  
  
 # Списание с баланса клиента указанной суммы  
 @classmethod  
 def withdraw(cls, \_name: str, \_sum: int):  
 \_sum = int(\_sum)  
 exception\_if\_invalid\_amount(\_sum)  
  
 \_client = cls.\_\_get\_client\_register\_if\_not\_exists(\_name)  
 cls.\_\_backup\_client(\_client)  
 cls.\_\_withdraw(\_client, \_sum)  
  
 return f'Списание {\_sum} у клиента {\_name}.'  
  
 # Вывод на экран баланса указанного клиента или всех клиентов  
 @classmethod  
 def balance(cls, \_name=''):  
 if \_name:  
 if \_name in cls.\_\_clients\_by\_name:  
 \_client = cls.\_\_clients\_by\_name[\_name]  
 return f'У клиента {\_client.name} баланс {\_client.balance}.'  
 else:  
 return 'NO CLIENT.'  
 else:  
 \_balances = ''  
  
 for \_client in cls.\_\_clients\_by\_name.values():  
 \_balances += f'У клиента {\_client.name} баланс {\_client.balance}.\n'  
  
 return \_balances.strip()  
  
 # Перевод средств от одного клиента другому  
 @classmethod  
 def transfer(cls, \_name1: str, \_name2: str, \_sum: int):  
 \_sum = int(\_sum)  
 exception\_if\_invalid\_amount(\_sum)  
  
 \_client1 = cls.\_\_get\_client\_register\_if\_not\_exists(\_name1)  
 \_client2 = cls.\_\_get\_client\_register\_if\_not\_exists(\_name2)  
  
 cls.\_\_backup\_client(\_client1)  
 cls.\_\_backup\_client(\_client2)  
  
 cls.\_\_withdraw(\_client1, \_sum)  
 cls.\_\_deposit(\_client2, \_sum)  
  
 return f'Перевод от клиента {\_client1.name} клиенту {\_client2.name}. Сумма {\_sum}.'  
  
 # Начисление процента от суммы на балансе всем клиентам с положительным балансом  
 @classmethod  
 def income(cls, \_percent: int):  
 \_log = ''  
 \_percent = int(\_percent)  
 exception\_if\_invalid\_amount(\_percent)  
  
 for \_client in cls.\_\_clients\_by\_name.values():  
 if \_client.balance > 0:  
 \_sum = int(\_client.balance \* (\_percent / 100))  
 print(\_client.balance \* (\_percent / 100), \_sum)  
  
 cls.\_\_backup\_client(\_client)  
 cls.\_\_withdraw(\_client, \_sum)  
 \_log += f'Зачисление {\_percent}%/{\_sum} клиенту {\_client.name}.\n'  
  
 return \_log.strip()  
  
 # Можно вызвать этот метод в любом месте транзакции командой TEST\_FAIL  
 @classmethod  
 def test\_fail(cls):  
 raise Exception("Test exception handling")  
  
 @classmethod  
 def \_\_backup\_client(cls, \_client: Client):  
 if \_client.name not in cls.\_\_clients\_backup:  
 cls.\_\_clients\_backup[\_client.name] = copy(\_client)  
  
 # Базовая операция списания без валидации  
 @classmethod  
 def \_\_withdraw(cls, \_client: Client, \_sum: int):  
 \_client.decrease\_balance(\_sum)  
  
 # Базовая операция зачисления без валидации  
 @classmethod  
 def \_\_deposit(cls, \_client: Client, \_sum: int):  
 \_client.increase\_balance(\_sum)  
  
 # Возвращаем клиента если он существует, либо регистрируем нового и возвращаем его  
 @classmethod  
 def \_\_get\_client\_register\_if\_not\_exists(cls, \_name: str):  
 if \_name in cls.\_\_clients\_by\_name:  
 return cls.\_\_clients\_by\_name[\_name]  
 else:  
 cls.register\_new\_client(\_name, 0)  
 return cls.\_\_clients\_by\_name[\_name]  
  
  
class Colors(Enum):  
 black = 'black'  
 white = 'white'  
  
  
def clear\_text\_fields(\_event):  
 global commands\_input\_field  
 global program\_output\_field  
 commands\_input\_field.delete("1.0", END)  
 write\_in\_output\_field('')  
  
  
def on\_calculate\_click(\_event):  
 global commands\_input\_field  
  
 # Собираем все команды из поля для ввода.  
 \_input\_commands = commands\_input\_field.get("1.0", END)  
 # Удаляем пустые строки и преобразуем строку в список команд с аргументами.  
 \_input\_commands = [x for x in \_input\_commands.split('\n') if x != '']  
 \_result\_messages = []  
 \_has\_error = False  
  
 # В ТЗ явно не отражено, поэтому было принято решение воспринимать последовательность команд как единую транзакцию  
 # и, в случае ошибки, откатывать все измененния, внесенные этими командами.  
  
 # Согласно ТЗ: Количество команд, которые может ввести пользователь за один раз – не более 20.  
 if len(\_input\_commands) > 20:  
 \_result\_messages = ['Максимальное количество команд - 20. Удалите некоторые команды и попробуйте снова.']  
 \_has\_error = True  
  
 else:  
 # Перебираем все команды с аргументами из списка.  
 for \_row in \_input\_commands:  
 # Получаем из строки саму команду и ее аргументы.  
 \_args = \_row.split(' ')  
 \_command = \_args[0]  
 del \_args[0]  
  
 # Согласно ТЗ: Команды должны вводится только большими буквами.  
 if \_command != \_command.upper():  
 \_error\_message = f'Команды должны вводится только большими буквами! Введено: {\_command}'  
 \_result\_messages.append(\_error\_message)  
 \_has\_error = True  
 break  
  
 else:  
 # Для вызова метода по имени  
 \_command = \_command.lower()  
  
 # Поскольку в ТЗ было четко прописано что  
 # "Предполагается, что пользователь такой системы грамотный и команды с аргументами вводит без ошибок в рамках их вышесформулированного синтаксиса."  
 # Дополнительных проверок на существование команд и соответствие аргументов реализовано не будет.  
 try:  
 # Вызываем метод по иемни и передаем аргументы.  
 \_result = getattr(AccountsManager, \_command)(\*\_args)  
  
 if isinstance(\_result, list):  
 \_result\_messages += \_result  
 else:  
 \_result\_messages.append(\_result)  
  
 # В случае, если все же кто-то ошибется, бросаем исключение прямо в лицо.  
 except Exception as e:  
 print(e.\_\_str\_\_())  
 \_result\_messages.append(e.\_\_str\_\_())  
 \_has\_error = True  
 break  
  
 if \_has\_error:  
 AccountsManager.rollback()  
 \_result\_messages.append('[!!!]Операции не были проведены из за ошибки!')  
 else:  
 # Если коммит прошел, то выдаем сообщение об успехе.  
 # Метод вернет False в случае если нет изменений для закрепления.  
 if AccountsManager.commit\_if\_has\_changes():  
 \_result\_messages.append('[V]Изменения успешно закрепелены.')  
  
 write\_in\_output\_field('\n'.join(\_result\_messages))  
  
  
def write\_in\_output\_field(\_text: str):  
 global program\_output\_field  
 print(f'Program output:\n{\_text}')  
 program\_output\_field.config(state='normal')  
 program\_output\_field.delete("1.0", END)  
 program\_output\_field.insert("1.0", \_text)  
 program\_output\_field.config(state='disabled')  
  
  
# Смена светлой темы на темную и наоборот  
def change\_theme(\_event):  
 global is\_dark\_theme  
 is\_dark\_theme = not is\_dark\_theme  
 update\_theme()  
  
  
# Установка цветов в зависимости от выбранной темы  
def update\_theme():  
 if is\_dark\_theme:  
 \_fg\_color = Colors.white.value  
 \_bg\_color = Colors.black.value  
 else:  
 \_fg\_color = Colors.black.value  
 \_bg\_color = Colors.white.value  
  
 window.configure(background=\_bg\_color)  
  
 # Перебираем все дочерние элементы окна и устанавливаем цвета для атрибутов, если эти атрибуты есть у элемента  
 for \_child in window.children.values():  
 if 'bg' in \_child.keys():  
 \_child.configure(bg=\_bg\_color)  
  
 if 'fg' in \_child.keys():  
 \_child.configure(fg=\_fg\_color)  
  
 if 'insertbackground' in \_child.keys():  
 \_child.configure(insertbackground=\_fg\_color)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 window = Tk()  
 window.title('Potapov-NA Banking')  
 window.resizable(False, False)  
  
 # Создаем интерфейс  
 input\_label = Label(window, text='Commands input:')  
 input\_label.grid(row=1, column=1, sticky='W', pady=2)  
  
 commands\_input\_field = Text(window, width=45, height=15, borderwidth=2, font=("Times New Roman", 14), wrap='no')  
 commands\_input\_field.grid(row=2, column=1, sticky='W', columnspan=3)  
 input\_scrollY = Scrollbar(window, command=commands\_input\_field.yview, orient=VERTICAL, width=12)  
 input\_scrollY.grid(row=2, column=4, sticky='NSE', pady=5)  
 input\_scrollX = Scrollbar(window, command=commands\_input\_field.xview, orient=HORIZONTAL, width=12)  
 input\_scrollX.grid(row=3, column=1, sticky='NWE', padx=5, columnspan=3)  
 commands\_input\_field.config(yscrollcommand=input\_scrollY.set, xscrollcommand=input\_scrollX.set)  
  
 output\_label = Label(window, text='Program output:')  
 output\_label.grid(row=1, column=5, sticky='W', pady=2)  
 # Поле для вывода сообщений послеь выполнения программы. Закрыто для редактирования пользователем.  
 program\_output\_field = Text(window, width=45, height=15, borderwidth=2, state='disabled',  
 font=('Times New Roman', 14),  
 wrap='word')  
 program\_output\_field.grid(row=2, column=5, sticky='W', columnspan=3)  
 output\_scrollY = Scrollbar(window, command=program\_output\_field.yview, orient=VERTICAL, width=12, troughcolor='red')  
 output\_scrollY.grid(row=2, column=8, sticky='NSE', pady=5)  
  
 calculate\_btn = Button(window, text='Calculate', width=10, height=1, borderwidth=2)  
 # Регистрируем обработчик нажатия  
 calculate\_btn.bind("<Button-1>", on\_calculate\_click)  
 calculate\_btn.grid(row=4, column=1, sticky='W', pady=4, padx=2)  
  
 clear\_btn = Button(window, text='Clear', width=10, height=1, borderwidth=2)  
 # Регистрируем обработчик нажатия  
 clear\_btn.bind("<Button-1>", clear\_text\_fields)  
 clear\_btn.grid(row=4, column=2, sticky='W', pady=4, padx=2)  
  
 change\_theme\_btn = Button(window, text='Theme', width=5, height=1, borderwidth=2)  
 # Регистрируем обработчик нажатия  
 change\_theme\_btn.bind("<Button-1>", change\_theme)  
 change\_theme\_btn.grid(row=1, column=7, sticky='E', pady=2, padx=2)  
  
 is\_dark\_theme = False  
 update\_theme()  
  
 # Если хочется чтобы клиенты не удалялись после перезапуска программы.  
 # В ТЗ не описано, наверное, банк однодневка :)  
 \_load\_clients\_database = False  
  
 if \_load\_clients\_database:  
 AccountsManager.use\_database = True  
 AccountsManager.load\_clients\_from\_database()  
  
 if not AccountsManager.is\_client\_exists('Potapov'):  
 AccountsManager.register\_new\_client('Potapov', 70165405)  
  
 else:  
 AccountsManager.register\_new\_client('Potapov', 70165405)  
  
 window.mainloop()

## Приложение 3.

# Ячеек памяти для 70165405: 9  
# Количество строк «цифрового дисплея» для 70165405: 10  
# Доп функции для П: Inv, sin, cos, tan  
  
import re  
from enum import Enum  
from tkinter import Frame, Tk, LEFT, Text, Button, Scrollbar, END, VERTICAL, Label, TOP  
import tkinter  
from math import sin, cos, tan  
  
# Флаг, указывающий находится ли калькулятор в расширенном режиме или в обычном  
is\_extended\_mode = False  
MAX\_HISTORY\_ROWS = 10  
# Компиляция регулярки для валидации возведения в степень.  
INVALID\_SQRT\_EXPRESSION\_RE = re.compile('.\*-\d{1,}sqrt.\*')  
# Все числа и знаки в поле для ввода представлены в виде списка, например ['53', '+', '1']  
current\_input\_expression = ['0']  
  
  
# Перечисление доступных типов кнопок  
class ButtonActionType(Enum):  
 save = 'MS'  
 read = 'MR'  
 clear = 'MC'  
 plus = 'M+'  
 minus = 'M-'  
  
  
# Статический класс для взаимодействия с ячейками памяти  
class Memory:  
 MEMORY\_CELLS\_COUNT = 9  
 # Ячейки памяти, MEMORY\_CELLS\_COUNT штук  
 \_\_memory\_cells\_by\_index = {x: '0' for x in range(0, MEMORY\_CELLS\_COUNT)}  
 # Списки кнопок для работы с ячейками памяти по их типу  
 \_\_memory\_buttons\_by\_action = {  
 ButtonActionType.plus: [],  
 ButtonActionType.minus: [],  
 ButtonActionType.clear: [],  
 ButtonActionType.read: [],  
 ButtonActionType.save: []  
 }  
  
 def \_\_init\_\_(self):  
 raise Exception('This is a static class!')  
  
 @classmethod  
 def append\_button(cls, \_action\_type: ButtonActionType, \_button: tkinter.Button):  
 # Кнопка будет прикреплена к ячейке памяти с тем же индеком, что в списке cls.\_\_memory\_buttons\_by\_action[\_action\_type]  
 # Здесь важна правильная последовательность добавления кнопок при создании UI.  
 cls.\_\_memory\_buttons\_by\_action[\_action\_type].append(\_button)  
  
 # Найти ячейку памяти по кнопке. Необходимо, чтобы при нажатии на кнопку понять с какой ячейкой памяти производятся дейтсвия  
 @classmethod  
 def get\_memory\_cell\_index(cls, \_search\_button: tkinter.Button):  
 \_action\_type = ButtonActionType(\_search\_button.cget("text"))  
 \_clicked\_button\_index = 0  
  
 while \_clicked\_button\_index < len(cls.\_\_memory\_buttons\_by\_action[\_action\_type]):  
 \_button = cls.\_\_memory\_buttons\_by\_action[\_action\_type][\_clicked\_button\_index]  
  
 if \_button is \_search\_button:  
 return \_clicked\_button\_index  
  
 else:  
 \_clicked\_button\_index += 1  
  
 raise Exception(f'Memory cell for button {\_search\_button.\_\_repr\_\_()} not found!')  
  
 # Записать значения в ячейку с индеском \_index  
 @classmethod  
 def write\_cell(cls, \_index: int, \_value):  
 \_value = str(\_value)  
  
 if \_value.replace('.', '').replace('-', '').isdigit():  
 cls.\_\_memory\_cells\_by\_index[\_index] = \_value  
 else:  
 raise ValueError(f'Invalid {\_value} numeric value')  
  
 # Прочитать значения из ячейки с индексом \_index  
 @classmethod  
 def read\_cell(cls, \_index: int):  
 return cls.\_\_memory\_cells\_by\_index[\_index]  
  
  
# Перечисление доступных операций  
class MathOperation(Enum):  
 sin = 'sin'  
 cos = 'cos'  
 tan = 'tan'  
  
  
# Рекурсия для подсчета суммы всех чисел пока не останется одно  
def calculate\_all\_numbers\_sum(\_value) -> int:  
 \_value = str(\_value)  
  
 if len(\_value) > 1:  
 return calculate\_all\_numbers\_sum(sum([int(x) for x in \_value]))  
 else:  
 return int(\_value)  
  
  
def calculate\_cos(\_value: float):  
 return cos(\_value)  
  
  
def calculate\_sin(\_value: float):  
 return sin(\_value)  
  
  
def calculate\_tan(\_value: float):  
 return tan(\_value)  
  
  
# Получения функции, реализующей операцию по ее типу  
FUNCTION\_BY\_OPERATION = {  
 MathOperation.sin: calculate\_sin,  
 MathOperation.cos: calculate\_cos,  
 MathOperation.tan: calculate\_tan  
}  
  
  
# region Обработчики нажатий на кнопки  
# Переключение между обычным и расширенным режимом.  
def toggle\_calculator\_mode(\_event):  
 global is\_extended\_mode  
 is\_extended\_mode = not is\_extended\_mode  
  
 if is\_extended\_mode:  
 window.title('Калькулятор. Расширенный режим.')  
 extension\_frame.pack(side=TOP)  
  
 else:  
 window.title('Калькулятор. Обычный режим.')  
 extension\_frame.pack\_forget()  
  
  
# Обработчик нажатия на кнопку CE  
def on\_ce\_pressed(\_event):  
 global current\_input\_expression  
  
 if current\_input\_expression:  
 # Удаляем из поля для ввода последнее введенное число или знак  
 del current\_input\_expression[-1]  
  
 # Если в результате поле осталось пустым, то пишем ноль.  
 if not current\_input\_expression:  
 current\_input\_expression = ['0']  
  
 update\_input\_field\_view()  
  
  
# Обработчик нажатия на кнопку C  
def on\_c\_pressed(\_event):  
 global current\_input\_expression  
 current\_input\_expression = ['0']  
 update\_input\_field\_view()  
 append\_to\_history\_field('C')  
  
  
# Обработчик нажатия кнопки =  
def on\_enter\_pressed(\_event):  
 global current\_input\_expression  
 # Получаем значения из поля для ввода, отсекая одинокие символы справа  
 \_user\_input = ''.join(current\_input\_expression).rstrip('(+-./\*')  
 # Результат вычисления в числовом представлении  
 \_digit\_result = 0  
  
 if not \_user\_input:  
 return  
  
 if INVALID\_SQRT\_EXPRESSION\_RE.match(\_user\_input):  
 # Результат вычисления в виде ошибки  
 \_result = 'Invalid sqrt expression'  
  
 else:  
 # Подготавливаем значение из поля для ввода, формируя корректное выражение для python  
 \_expression = \_user\_input.replace('^', '\*\*').replace('sqrt', '\*\*(0.5)').replace(' ', '')  
  
 try:  
 # Само вычисление значения выражения было решено сделать на основе функции eval.  
 # Оглядываясь назад, я бы сделал иначе.. В итоге подпер костылями и поехало.  
 \_result = eval(\_expression)  
  
 # Отбрасываем точку в случае если в результате тип float вида 12.0  
 if int(\_result) - \_result == 0:  
 \_result = int(\_result)  
  
 \_digit\_result = \_result  
  
 except ZeroDivisionError:  
 \_result = 'Zero division error'  
  
 except Exception as e:  
 \_result = 'Invalid operation'  
 print(e.\_\_str\_\_())  
  
 print(\_user\_input, '=', \_result)  
  
 # Добавляем в историю результат в текстовом представлении. Это может быть цифра или сообщение об ошибке.  
 append\_to\_history\_field(f'{\_user\_input}\n=\n{\_result}')  
 # В поле для ввода возвращаем результат в числовом представлении. 0 при ошибке.  
 write\_in\_input\_field(str(\_digit\_result))  
  
  
# Обработчик нажатия кнопки с символами 0-9.+-()/\*  
def on\_char\_button\_pressed(\_event):  
 append\_to\_input\_field(\_event.widget.cget("text"))  
  
  
# Обработка нажатия на любую кнопку для работы с ячейками памяти  
def on\_memory\_button\_pressed(\_event):  
 # Получаем саму кнопку, которая была ажата  
 \_pressed\_button = \_event.widget  
 # По тексту на кнопке получаем ее тип  
 \_action\_type = ButtonActionType(\_pressed\_button.cget("text"))  
 # Затем обращаемся к памяти для поиска ячейки памяти, к которой привязана эта кнопка  
 \_memory\_cell\_index = Memory.get\_memory\_cell\_index(\_pressed\_button)  
 print(f'Pressed {\_action\_type.name}/{\_action\_type.value} button in column {\_memory\_cell\_index + 1}')  
  
 # Далее действуем в зависимости от типа нажатой кнопки  
 if \_action\_type == ButtonActionType.clear:  
 Memory.write\_cell(\_memory\_cell\_index, '0')  
  
 elif \_action\_type == ButtonActionType.save:  
 on\_enter\_pressed(None)  
 \_current\_value = get\_input\_field\_value() if get\_input\_field\_value() else '0'  
 Memory.write\_cell(\_memory\_cell\_index, get\_input\_field\_value())  
  
 elif \_action\_type == ButtonActionType.plus:  
 # Записываем текущее значение из поля для ввода в ячейку памяти если в ячейке памяти 0  
 if Memory.read\_cell(\_memory\_cell\_index) == '0':  
 \_current\_value = get\_input\_field\_value() if get\_input\_field\_value() else '0'  
 on\_enter\_pressed(None)  
 Memory.write\_cell(\_memory\_cell\_index, get\_input\_field\_value())  
  
 # Подставляем значение из памяти в выражение в поле для ввода  
 append\_to\_input\_field('+')  
 append\_to\_input\_field(Memory.read\_cell(\_memory\_cell\_index))  
 # Нажимаем на кнопку = для проведения расчета и записи истории штатным образом  
 on\_enter\_pressed(None)  
  
 elif \_action\_type == ButtonActionType.read:  
 append\_to\_input\_field(Memory.read\_cell(\_memory\_cell\_index))  
  
 elif \_action\_type == ButtonActionType.minus:  
 # Записываем текущее значение из поля для ввода в ячейку памяти если в ячейке памяти 0  
 if Memory.read\_cell(\_memory\_cell\_index) == '0':  
 \_current\_value = get\_input\_field\_value() if get\_input\_field\_value() else '0'  
 on\_enter\_pressed(None)  
 Memory.write\_cell(\_memory\_cell\_index, \_current\_value)  
  
 # Подставляем значение из памяти в выражение в поле для ввода  
 append\_to\_input\_field('-')  
 append\_to\_input\_field(Memory.read\_cell(\_memory\_cell\_index))  
 # Нажимаем на кнопку = для проведения расчета и записи истории штатным образом  
 on\_enter\_pressed(None)  
  
 else:  
 raise ValueError(f'Unexpected action type {\_action\_type}')  
  
  
# Обработчик нажатия на кнопку <>. Суммирует все цифры в поле для ввода пока не останется одна  
def sum\_all\_numbers(\_event):  
 # Жмем = на случай если в поле для ввода в данной момент какое-то выражение.  
 # В таком случае операция будет произведена над результатом этого выражения.  
 on\_enter\_pressed(None)  
 \_input = get\_input\_field\_value()  
 \_input = str(int(float(\_input))).lstrip('-')  
  
 \_result = calculate\_all\_numbers\_sum(\_input)  
  
 write\_in\_input\_field(str(\_result))  
 append\_to\_history\_field(f'{"+".join(\_input)}...\n=\n{str(\_result)}')  
  
  
# Обработчик нажатия на кнопки sin, cos, tan  
def on\_sin\_cos\_tan\_pressed(\_event):  
 # Выполняемая операци определяется на основании подписи кнопки  
 \_operation = MathOperation(\_event.widget.cget("text"))  
 on\_enter\_pressed(None)  
 \_current\_value = get\_input\_field\_value()  
  
 try:  
 # Получаем нужную функцию по типу операции  
 \_result = FUNCTION\_BY\_OPERATION[\_operation](float(\_current\_value))  
 write\_in\_input\_field(str(\_result))  
  
 except ValueError as e:  
 # В случае ошибки в математической функции пишем ее в историю  
 \_result = e.\_\_str\_\_()  
 write\_in\_input\_field('0')  
  
 append\_to\_history\_field(f'{\_operation.value}({\_current\_value})\n=\n{str(\_result)}')  
  
  
# endregion  
  
# Запись текста в текстовое поле, которое заблокированно для ввода пользователем  
def write\_in\_blocked\_field(\_text: str, \_field):  
 \_field.config(state='normal')  
 \_field.delete("1.0", END)  
 # Применяем тэг, по которому происходит выравнивание текста по правому краю  
 \_field.insert("1.0", \_text, 'tag-right')  
 \_field.config(state='disabled')  
  
  
# Перезаписать историю новыми данными  
def write\_in\_history\_field(\_text: str):  
 global calculator\_history\_field  
 write\_in\_blocked\_field(\_text, calculator\_history\_field)  
  
  
# Перезаписать поле для ввода новыми данными. Запись происходит через список current\_input\_expression,  
# а UI обновляется в update\_input\_field\_view()  
def write\_in\_input\_field(\_text: str):  
 global current\_input\_expression  
  
 if \_text:  
 current\_input\_expression = [\_text]  
 else:  
 current\_input\_expression = ['0']  
  
 update\_input\_field\_view()  
  
  
# Обновление тексового поля для ввода нв UI  
def update\_input\_field\_view():  
 global current\_input\_expression  
 write\_in\_blocked\_field(''.join(current\_input\_expression), calculator\_input\_field)  
 print(current\_input\_expression)  
  
  
# Добавить значение к уже существующим в поле для ввода  
def append\_to\_input\_field(\_text: str):  
 global current\_input\_expression  
  
 # Если поле для ввода не пустое и хотят добавить точку, то добавляем  
 if current\_input\_expression and \_text == '.':  
 current\_input\_expression[-1] += \_text  
  
 # Если поле для ввода не пустое и хотят добавить цифру, в предыдущим знаком была тоже цифра, то...  
 elif current\_input\_expression and \  
 \_text.isdigit() and \  
 current\_input\_expression[-1].replace('-', '').replace('.', '').isdigit():  
  
 # Если последнее введенное число было 0, то заменяем его, чтобы числа не начинались с нуля.  
 # Например: ['8', '-', '0'] после добавления 2 будет ['8', '-', '2'], а вот ['8', '-', '20'] под условие не попадет.  
 if current\_input\_expression[-1] == '0':  
 current\_input\_expression[-1] = \_text  
  
 # Иначе добавляем к последнему введенному числу еще одну цифру.  
 # Например: ['8', '-', '20'] после добавления 2 будет ['8', '-', '202']  
 else:  
 current\_input\_expression[-1] += \_text  
  
 # Иначе просто добавляем как новый элемент списка.  
 # В данному случае у нас либо рядом два знака, либо число и знак.  
 # Либо список пустой, но такого быть не должно, должен подставляться 0.  
 else:  
 current\_input\_expression.append(\_text)  
  
 update\_input\_field\_view()  
  
  
# Добавляем к существующей истории новые строки.  
def append\_to\_history\_field(\_text: str):  
 global calculator\_history\_field  
 \_current\_history = calculator\_history\_field.get("1.0", END)  
 \_new\_history = f'{\_text}\n{\_current\_history}'  
 \_history\_rows = \_new\_history.split('\n')  
  
 # Если строк больше допустимого, то обрезаем лишнее.  
 if len(\_history\_rows) > MAX\_HISTORY\_ROWS:  
 \_history\_rows = \_history\_rows[0:MAX\_HISTORY\_ROWS]  
 \_new\_history = '\n'.join(\_history\_rows)  
  
 write\_in\_history\_field(\_new\_history)  
  
  
# Получаем текст из поля для ввода выражения.  
def get\_input\_field\_value():  
 return calculator\_input\_field.get("1.0", END).strip().replace(' ', '')  
  
  
# region Создание GUI.  
if \_\_name\_\_ == '\_\_main\_\_':  
 window = Tk()  
 window.title('Калькулятор. Обычный режим.')  
 window.resizable(False, False)  
  
 base\_frame = Frame(window)  
 base\_frame.pack(side=LEFT)  
  
 # Поле для ввода.  
 calculator\_input\_field = Text(base\_frame, width=30, height=1, font=("Times New Roman", 14), wrap='no',  
 state='disabled')  
 calculator\_input\_field.tag\_configure('tag-right', justify='right')  
 calculator\_input\_field.grid(row=1, column=1, sticky='WN', columnspan=4)  
  
 # Экран истории со скролл баром.  
 calculator\_history\_field = Text(base\_frame, width=30, height=8, font=("Times New Roman", 14), wrap='word',  
 state='disabled')  
 calculator\_history\_field.tag\_configure('tag-right', justify='right')  
 calculator\_history\_field.grid(row=2, column=1, sticky='WN', columnspan=4, rowspan=5)  
 input\_scrollY = Scrollbar(base\_frame, command=calculator\_history\_field.yview, orient=VERTICAL, width=12)  
 input\_scrollY.grid(row=2, column=5, sticky='NSE', pady=5, rowspan=5)  
 calculator\_history\_field.config(yscrollcommand=input\_scrollY.set)  
  
 # Кнопки базовых символов для ввода.  
 \_row = 8  
 \_column = 1  
 \_i = 0  
 # Итерируемся по списку кнопок. Список представляет из себя развернутую панель ввода калькулятора.  
 for \_char in ['(', ')', 'sqrt', '^', '7', '8', '9', '\*', '4', '5', '6', '/', '1', '2', '3', '+', '.', '0', '', '-']:  
  
 # В списке могут быть пустые строки для того чтобы оставить пропуски между кнопкми  
 if \_char:  
 char\_button = Button(base\_frame, text=\_char, width=3, height=1)  
 char\_button.bind("<Button-1>", on\_char\_button\_pressed)  
 char\_button.grid(row=\_row, column=\_column, sticky='W', pady=2, padx=2)  
  
 \_i += 1  
 # Смещаемся на ряд вниз каждые 4 итерации.  
 if not \_i % 4:  
 \_row += 1  
 \_column = 1  
 else:  
 \_column += 1  
  
 # Создаем специфические кнопки с отдельными обработчиками.  
 # Кнопка равно.  
 enter\_button = Button(base\_frame, text='=', width=3, height=1, bg='green')  
 enter\_button.bind("<Button-1>", on\_enter\_pressed)  
 enter\_button.grid(row=12, column=3, sticky='W', pady=2, padx=2)  
  
 # Кнопка <> для вычисления суммы всех цифр числа до последнего.  
 delete\_last\_symbol = Button(base\_frame, text='<>', width=3, height=1, bg='red')  
 delete\_last\_symbol.bind("<Button-1>", sum\_all\_numbers)  
 delete\_last\_symbol.grid(row=7, column=3, sticky='W', pady=2, padx=2)  
  
 # Кнопка CE для удаления последнего введенного символа или числа.  
 delete\_last\_symbol = Button(base\_frame, text='CE', width=3, height=1, bg='red')  
 delete\_last\_symbol.bind("<Button-1>", on\_ce\_pressed)  
 delete\_last\_symbol.grid(row=7, column=2, sticky='W', pady=2, padx=2)  
  
 # Кнопка C для очистки поля ввода.  
 delete\_all\_symbols = Button(base\_frame, text='C', width=3, height=1, bg='red')  
 delete\_all\_symbols.bind("<Button-1>", on\_c\_pressed)  
 delete\_all\_symbols.grid(row=7, column=1, sticky='W', pady=2, padx=2)  
  
 # Кнопка для переключения между обычным и расширенным режимом калькулятора.  
 change\_mode\_btn = Button(base\_frame, text='===', width=3, height=1, bg='yellow')  
 change\_mode\_btn.bind("<Button-1>", toggle\_calculator\_mode)  
 change\_mode\_btn.grid(row=7, column=4, sticky='W', pady=2, padx=2)  
  
 # Рамку для расширенного режима не упаковываем pack(), чтобы она не отображалась сразу при запуске.  
 extension\_frame = Frame(window)  
  
 # Построчно создаем кнопки для ячеек памяти  
 for \_i in range(1, Memory.MEMORY\_CELLS\_COUNT + 1):  
 buttons\_row = []  
 buttons\_frame = Frame(extension\_frame, bg='blue')  
 buttons\_frame.grid(row=\_i, column=1, padx=2, pady=2, columnspan=6)  
  
 buttons\_row.append(Button(buttons\_frame, text=ButtonActionType.save.value, width=3, height=1))  
 buttons\_row.append(Button(buttons\_frame, text=ButtonActionType.plus.value, width=3, height=1))  
 buttons\_row.append(Button(buttons\_frame, text=ButtonActionType.minus.value, width=3, height=1))  
 buttons\_row.append(Button(buttons\_frame, text=ButtonActionType.clear.value, width=3, height=1))  
 buttons\_row.append(Button(buttons\_frame, text=ButtonActionType.read.value, width=3, height=1))  
  
 # Проходимся по только созданной строке кнопок и биндим эвент для каждой кнопки  
 # А так же регистрируем каждую кнопку в классе Memory  
 for \_col in range(1, 6):  
 \_button = buttons\_row[\_col - 1]  
 \_button.bind("<Button-1>", on\_memory\_button\_pressed)  
 buttons\_row[\_col - 1].grid(row=\_i, column=\_col, sticky='W', pady=2, padx=2)  
  
 \_action\_type = ButtonActionType(\_button.cget("text"))  
 Memory.append\_button(\_action\_type, \_button)  
  
 label = Label(buttons\_frame, text=\_i, width=1, height=1, bg='blue', font=("Times New Roman", 14), fg='white')  
 label.grid(row=\_i, column=6, sticky='W', pady=2, padx=2)  
  
 sin\_button = Button(extension\_frame, text='sin', width=5, height=2, bg='yellow')  
 sin\_button.bind("<Button-1>", on\_sin\_cos\_tan\_pressed)  
 sin\_button.grid(row=13, column=1, sticky='W', pady=2)  
  
 cos\_button = Button(extension\_frame, text='cos', width=5, height=2, bg='yellow')  
 cos\_button.bind("<Button-1>", on\_sin\_cos\_tan\_pressed)  
 cos\_button.grid(row=13, column=2, sticky='W', pady=2)  
  
 tan\_button = Button(extension\_frame, text='tan', width=5, height=2, bg='yellow')  
 tan\_button.bind("<Button-1>", on\_sin\_cos\_tan\_pressed)  
 tan\_button.grid(row=13, column=3, sticky='W', pady=2)  
  
 label = Label(extension\_frame, text='\*measured in radians', height=1,  
 font=("Times New Roman", 10), fg='black')  
 label.grid(row=14, column=1, sticky='W', pady=2, padx=2, columnspan=3)  
  
 # В первую ячейку памяти заранее записываю свой ID для удобства  
 Memory.write\_cell(0, 70165405)  
 update\_input\_field\_view()  
 window.mainloop()  
# endregion

## Приложение 4.

import datetime  
import random  
from collections import namedtuple  
from tkinter import Tk, Entry, Button, Canvas, Label, StringVar  
import copy  
  
# Константы  
ID = '70165405'  
TOTAL\_DISKS\_COUNT = sum([int(x) for x in ID])  
DEFAULT\_PROGRESS\_PERCENTS = (ID[0:2], ID[2:4], ID[4:6], ID[6:8])  
  
# Именованный кортеж для хранения информации о диске  
Disk = namedtuple('Disk', ('color', 'size'))  
DISK\_HEIGHT = 15  
PILLARS\_COUNT = 8  
PILLAR\_WIDTH = 6  
PILLAR\_HEIGHT = 350  
STAND\_HEIGHT = 15  
CANVAS\_SIZE = (1200, 500)  
PILLARS\_OFFSET = (CANVAS\_SIZE[0] - CANVAS\_SIZE[0] / PILLARS\_COUNT) / PILLARS\_COUNT  
  
# Подсчет количества итераций алгоритма ханойских башен  
total\_disks\_moves = 0  
  
  
# Статический класс для переключения и записи итераций алгоритма ханойских башен  
class IterationsManager:  
 # В историю сохраняется положение всех дисков на каждой итерации.  
 # Выглядит одна запись в истории так: {1: [Disk], 2:[Disk].... 8[Disk]} где цифра - номер шпинделя  
 \_\_disks\_positions\_history = [{}]  
 \_\_current\_iteration = 0  
 \_\_total\_iterations\_count = 0  
  
 def \_\_init\_\_(self):  
 raise Exception('This is a static class!')  
  
 # Общее количество итерация  
 @classmethod  
 def total\_iterations\_count(cls) -> int:  
 return len(cls.\_\_disks\_positions\_history) - 1  
  
 # Текущая выбранная итерация  
 @classmethod  
 def current\_iteration\_number(cls) -> int:  
 return cls.\_\_current\_iteration  
  
 # Текущее расположение дисков на шпинделях на текущей выбранной итерации  
 @classmethod  
 def current\_disks\_positions\_by\_pillar(cls) -> dict:  
 return cls.\_\_disks\_positions\_history[cls.current\_iteration\_number()]  
  
 # Выбрать итерация с номером \_iteration\_number  
 @classmethod  
 def select\_iteration(cls, \_iteration\_number: int):  
 if 0 > \_iteration\_number < cls.\_\_total\_iterations\_count:  
 raise Exception(f"Iteration {\_iteration\_number} not present")  
  
 cls.\_\_current\_iteration = \_iteration\_number  
  
 # Выбрать самую последнюю итерацию из истории  
 @classmethod  
 def select\_last\_iteration(cls):  
 cls.\_\_current\_iteration = len(cls.\_\_disks\_positions\_history) - 1  
  
 # Выбрать следующую по счетку итерацию в истории  
 @classmethod  
 def select\_next\_iteration(cls):  
 if cls.\_\_current\_iteration + 1 >= len(cls.\_\_disks\_positions\_history):  
 raise StopIteration  
  
 cls.\_\_current\_iteration += 1  
  
 # Выбрать предыдущую по счетку итерацию в истории  
 @classmethod  
 def select\_previous\_iteration(cls):  
 if cls.\_\_current\_iteration - 1 < 0:  
 raise StopIteration  
  
 cls.\_\_current\_iteration -= 1  
  
 # Добавить еще одну итерацию в историю. Необходимо в момент расчета алгоритма ханойских башен для сохранения истории  
 @classmethod  
 def create\_and\_select\_new\_iteration(cls):  
 cls.\_\_current\_iteration = cls.total\_iterations\_count() + 1  
 \_previous\_iteration = cls.\_\_disks\_positions\_history[cls.\_\_current\_iteration - 1]  
 # cls.\_\_disks\_positions\_history.append(copy.deepcopy(\_previous\_iteration)) # работало дольше в 10 раз  
 \_previous\_iteration = {x: copy.copy(y) for x, y in  
 \_previous\_iteration.items()} # \_previous\_iteration = {1: [Disk], 2:[Disk], 3[Disk]....} где цифра - номер шпинделя  
 cls.\_\_disks\_positions\_history.append(\_previous\_iteration)  
  
  
# Необходимо вычислить, за какое минимальное количество итераций переместятся все диски на шпиндель номер 1 по следующим  
# правилам:  
# а) За одну итерацию можно переместить не более одного диска  
# б) Диски можно класть только с большего на меньший  
# в) Со шпинделя номер 8 можно перекладывать диски только на  
# шпиндели 7 и 6  
# г) Со шпинделя номер 1 можно перекладывать диски только на  
# шпиндели номер 2 и 3  
# д) Со шпинделей от 2 по 7 можно перекладывать диски только на  
# два соседних шпинделя  
def move\_all\_disks\_and\_write\_history():  
 # Рекурсивная функция для переноса дисков с одного шпинделя на другой через третий  
 def move\_disks(\_disks\_count, \_from, \_to, \_via):  
 global total\_disks\_moves  
  
 # Одна итерация движения диска  
 def move\_iteration():  
 global total\_disks\_moves  
 if \_disks\_count <= 0:  
 return  
  
 if \_from == \_to or \_from == \_via or \_to == \_via:  
 raise Exception(f"Такой перенос невозможен! from:{\_from} to:{\_to} via:{\_via}")  
  
 if 1 > \_from > PILLARS\_COUNT or 1 > \_to > PILLARS\_COUNT or 1 > \_via > PILLARS\_COUNT:  
 raise Exception(f"Выход за диапазон допустимых значений! from:{\_from} to:{\_to} via:{\_via}")  
  
 if \_from == 1:  
 if (\_from + 2) < \_to or (\_from + 2) < \_via:  
 raise Exception(  
 f"Со шпинделя номер 1 можно перекладывать диски только на шпиндели 2 и 3! from:{\_from} to:{\_to} via:{\_via}")  
 elif \_from == 8:  
 if \_to < (\_from - 2) or \_via < (\_from - 2):  
 raise Exception(  
 f"Со шпинделя номер 8 можно перекладывать диски только на шпиндели 7 и 6! from:{\_from} to:{\_to} via:{\_via}")  
  
 # Создаем новую итерацию в истории  
 IterationsManager.create\_and\_select\_new\_iteration()  
 # Снимаем диск со шпидлея  
 \_disk\_to\_move = IterationsManager.current\_disks\_positions\_by\_pillar()[\_from].pop(-1)  
  
 if IterationsManager.current\_disks\_positions\_by\_pillar()[\_to]:  
 if \_disk\_to\_move.size > IterationsManager.current\_disks\_positions\_by\_pillar()[\_to][-1].size:  
 raise Exception("Большой диск нельзя класть на маленький!")  
  
 # Надеваем диск на другой шпиндель  
 IterationsManager.current\_disks\_positions\_by\_pillar()[\_to].append(\_disk\_to\_move)  
 total\_disks\_moves += 1  
  
 if \_disks\_count == 0:  
 return  
  
 if \_disks\_count == 1:  
 move\_iteration()  
 return  
  
 move\_disks(\_disks\_count - 1, \_from, \_via, \_to)  
 move\_iteration()  
 move\_disks(\_disks\_count - 1, \_via, \_to, \_from)  
  
 def move\_from\_left\_to\_right():  
 \_len = len(IterationsManager.current\_disks\_positions\_by\_pillar()[3])  
 move\_disks(\_len, 3, 4, 2)  
 move\_disks(\_len, 4, 5, 3)  
 move\_disks(\_len, 5, 6, 4)  
  
 \_len = len(IterationsManager.current\_disks\_positions\_by\_pillar()[4])  
 move\_disks(\_len, 4, 3, 5)  
 move\_disks(\_len, 3, 2, 4)  
 move\_disks(\_len, 2, 1, 3)  
  
 \_len = len(IterationsManager.current\_disks\_positions\_by\_pillar()[6])  
 move\_disks(\_len, 6, 5, 7)  
 move\_disks(\_len, 5, 4, 6)  
 move\_disks(\_len, 4, 3, 5)  
  
 print('Processing disk movement..')  
 # 10919 итераций.  
 # С помощью инструкции return можно остановиться в любом месте и посмотреть как выглядят столбы с дисками.  
 # Собираем диски в несколько кучек и сдвигаем их в правую сторону, ближе к первому столбу.  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[1]), 1, 2, 3)  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[2]), 2, 1, 3)  
  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[3]), 3, 4, 2)  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[4]), 4, 3, 5)  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[3]), 3, 2, 4)  
  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[5]), 5, 6, 4)  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[6]), 6, 5, 7)  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[5]), 5, 4, 6)  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[4]), 4, 3, 5)  
  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[7]), 7, 8, 6)  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[8]), 8, 6, 7)  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[6]), 6, 5, 7)  
 move\_disks(len(IterationsManager.current\_disks\_positions\_by\_pillar()[5]), 5, 4, 6)  
  
 # Начинаем перебрасывать самые маленькие диски за самые большие  
 \_len = len(IterationsManager.current\_disks\_positions\_by\_pillar()[1])  
 move\_disks(\_len, 1, 3, 2)  
 move\_disks(\_len, 3, 4, 2)  
 move\_disks(\_len, 4, 5, 3)  
 move\_disks(\_len, 5, 6, 4)  
 move\_disks(\_len, 6, 7, 5)  
 move\_disks(\_len, 7, 8, 6)  
  
 \_len = len(IterationsManager.current\_disks\_positions\_by\_pillar()[2])  
 move\_disks(\_len, 2, 3, 1)  
 move\_disks(\_len, 3, 4, 2)  
 move\_disks(\_len, 4, 5, 3)  
 move\_disks(\_len, 5, 6, 4)  
 move\_disks(\_len, 6, 7, 5)  
  
 \_len = len(IterationsManager.current\_disks\_positions\_by\_pillar()[3])  
 move\_disks(\_len, 3, 4, 2)  
 move\_disks(\_len, 4, 5, 3)  
 move\_disks(\_len, 5, 6, 4)  
  
 # Перекладываем самые большие диски на первый столб.  
 \_len = len(IterationsManager.current\_disks\_positions\_by\_pillar()[4])  
 move\_disks(\_len, 4, 3, 5)  
 move\_disks(\_len, 3, 2, 4)  
 move\_disks(\_len, 2, 1, 3)  
  
 # Освобождаем место для перемещения остальных 3х кучек слева направо.  
 \_len = len(IterationsManager.current\_disks\_positions\_by\_pillar()[8])  
 move\_disks(\_len, 8, 6, 7)  
 move\_disks(\_len, 6, 5, 7)  
 move\_disks(\_len, 5, 4, 6)  
 move\_disks(\_len, 4, 3, 5)  
  
 \_len = len(IterationsManager.current\_disks\_positions\_by\_pillar()[7])  
 move\_disks(\_len, 7, 8, 6)  
  
 \_len = len(IterationsManager.current\_disks\_positions\_by\_pillar()[6])  
 move\_disks(\_len, 6, 5, 7)  
 move\_disks(\_len, 5, 4, 6)  
  
 # Переносим кучки направо  
 move\_from\_left\_to\_right()  
  
 \_len = len(IterationsManager.current\_disks\_positions\_by\_pillar()[8])  
 move\_disks(\_len, 8, 6, 7)  
 move\_disks(\_len, 6, 5, 7)  
 move\_disks(\_len, 5, 4, 6)  
  
 move\_from\_left\_to\_right()  
  
 move\_disks(\_len, 3, 2, 4)  
 move\_disks(\_len, 2, 1, 3)  
 print("Done!")  
  
  
def show\_next\_iteration(\_event):  
 try:  
 IterationsManager.select\_next\_iteration()  
 redraw\_disks()  
 except StopIteration:  
 print('This is the last iteration!')  
 error\_label\_text.set('This is the last iteration!')  
  
  
def show\_previous\_iteration(\_event):  
 try:  
 IterationsManager.select\_previous\_iteration()  
 redraw\_disks()  
 except StopIteration:  
 print('This is the first iteration!')  
 error\_label\_text.set('This is the first iteration!')  
  
  
def show\_end\_of\_iterations(\_event):  
 IterationsManager.select\_last\_iteration()  
 redraw\_disks()  
  
  
def show\_start\_of\_iterations(\_event):  
 IterationsManager.select\_iteration(0)  
 redraw\_disks()  
  
  
# Показываем промежуточную терацию по проценту  
def show\_intermediate\_iteration(\_event, \_percent):  
 \_percent = \_percent.get()  
 print('Percent', \_percent)  
 if \_percent.replace('.', '').isdigit():  
 \_percent = float(\_percent)  
 if \_percent > 100:  
 error\_label\_text.set(f'Invalid percent value! Percent must be between 0 and 100, got {\_percent}')  
 print(f'-Invalid percent value! Percent must be between 0 and 100, got {\_percent}')  
 else:  
 \_requested\_iteration = (IterationsManager.total\_iterations\_count() / 100) \* \_percent  
 \_requested\_iteration = round(\_requested\_iteration, 3)  
  
 if int(\_requested\_iteration) - \_requested\_iteration == 0:  
 \_requested\_iteration = int(\_requested\_iteration)  
  
 elif int(\_requested\_iteration.\_\_ceil\_\_()) > IterationsManager.total\_iterations\_count():  
 \_requested\_iteration = int(\_requested\_iteration)  
  
 print('-Requested iteration', \_requested\_iteration)  
  
 IterationsManager.select\_iteration(\_requested\_iteration)  
 redraw\_disks()  
  
 else:  
 print(f'-Invalid percent value! {\_percent}')  
 error\_label\_text.set(f'Invalid percent value! Expected digits, got {\_percent}')  
  
  
# Перерисовываем все диски на экране в соответствии с их положением на выбранной итерации  
def redraw\_disks():  
 # Удаляем все нарисованные диски  
 canvas.delete('disks')  
 error\_label\_text.set('')  
 # Выводим номер выбранной итерации на экран  
 current\_iteration\_text.set(IterationsManager.current\_iteration\_number())  
  
 # Если выбрана промежуточная итерация, то тут будет кортеж: перемещаемый диск и две позиции, откуда и куда он перемещается  
 \_flying\_disk\_and\_position = None  
  
 if isinstance(IterationsManager.current\_iteration\_number(), float):  
 # Получаем номера соседних целых итераций  
 \_first\_iteration = IterationsManager.current\_iteration\_number().\_\_floor\_\_()  
 \_second\_iteration = IterationsManager.current\_iteration\_number().\_\_ceil\_\_()  
  
 # Выбираем первую итерацию и получаем расположение дисков на ней  
 IterationsManager.select\_iteration(\_first\_iteration)  
 \_fist\_iteration\_disks\_state = IterationsManager.current\_disks\_positions\_by\_pillar()  
  
 # Выбираем вторую итерацию и получаем расположение дисков на ней  
 IterationsManager.select\_iteration(\_second\_iteration)  
 \_second\_iteration\_disks\_state = IterationsManager.current\_disks\_positions\_by\_pillar()  
 \_positions = []  
 \_flying\_disk = None  
  
 # Проходимся циклом по первой итерации и сравниваем ее со второй. Ищем диск, который был перемещен  
 \_i = 1  
 while \_i < len(\_fist\_iteration\_disks\_state):  
  
 # Ищем шпиндели, диски на которых откличаются. Таких должно быть два  
 if \_fist\_iteration\_disks\_state[\_i] != \_second\_iteration\_disks\_state[\_i]:  
 if not \_flying\_disk:  
 # Диск, который был перемещен, будет в разнице между шпинделем откуда и шпинделем куда  
 \_flying\_disk = list(  
 set(\_fist\_iteration\_disks\_state[\_i]).difference(set(\_second\_iteration\_disks\_state[\_i])))  
 \_positions.append(\_i)  
  
 \_i += 1  
  
 if not \_flying\_disk:  
 raise Exception(  
 f"Не удалось найти перемещенный диск на итерации {IterationsManager.current\_iteration\_number()}")  
  
 else:  
 # Наконец создаем кортеж  
 \_flying\_disk\_and\_position = (\_flying\_disk[0], \*\_positions)  
  
 # Проходимся по всем шпинделям на выбранной целой итерации, не важно floor или ceil  
 for \_pillar, \_disks in IterationsManager.current\_disks\_positions\_by\_pillar().items():  
  
 # Проходимся по всем дискам на шпинделе  
 \_i = 1  
 while \_i <= len(\_disks):  
 \_disk = \_disks[\_i - 1]  
  
 # Если диск соответствует диску, который был перемещен на дробной итерации, то изображаем его в воздухе  
 # Сравнение производим по рамеру т.к он уникален  
 if \_flying\_disk\_and\_position and \_disk.size == \_flying\_disk\_and\_position[0].size:  
 \_first\_position = \_flying\_disk\_and\_position[1]  
 \_second\_position = \_flying\_disk\_and\_position[2]  
  
 # Высчитываем где должен находиться летающий диск  
 \_flying\_disk\_center = PILLARS\_OFFSET \* (  
 PILLARS\_COUNT - ((\_first\_position + \_second\_position) / 2) + 1)  
 \_flying\_disk\_height = CANVAS\_SIZE[1] - 400 - DISK\_HEIGHT  
  
 # Рисуем линии от диска до шпинделей  
 canvas.create\_line(\_flying\_disk\_center, \_flying\_disk\_height,  
 PILLARS\_OFFSET \* (PILLARS\_COUNT - \_first\_position + 1) + PILLAR\_WIDTH / 2,  
 CANVAS\_SIZE[1] - STAND\_HEIGHT - DISK\_HEIGHT,  
 fill=\_disk.color, tags='disks', dash=2, width=5)  
  
 canvas.create\_line(\_flying\_disk\_center, \_flying\_disk\_height,  
 PILLARS\_OFFSET \* (PILLARS\_COUNT - \_second\_position + 1) + PILLAR\_WIDTH / 2,  
 CANVAS\_SIZE[1] - STAND\_HEIGHT - DISK\_HEIGHT,  
 fill=\_disk.color, tags='disks', dash=2, width=5)  
  
 # Задаем параметры для рисования дискаы  
 \_pillar\_center = \_flying\_disk\_center  
 \_disk\_height = \_flying\_disk\_height  
  
 else:  
 # Задаем параметры для рисования дискаы  
 \_pillar\_center = PILLARS\_OFFSET \* (PILLARS\_COUNT - \_pillar + 1) + PILLAR\_WIDTH / 2  
 \_disk\_height = CANVAS\_SIZE[1] - STAND\_HEIGHT - DISK\_HEIGHT \* \_i  
  
 # Рисуем диск  
 canvas.create\_rectangle(\_pillar\_center - \_disk.size / 2,  
 \_disk\_height,  
 \_pillar\_center + \_disk.size / 2,  
 \_disk\_height + DISK\_HEIGHT,  
 fill=\_disk.color, tags='disks')  
  
 # Пишем размер диска на самом диске  
 canvas.create\_text((\_pillar\_center, \_disk\_height + DISK\_HEIGHT / 2),  
 text=\_disk.size, font=("Times New Roman", 10), fill='white', tags='disks')  
 \_i += 1  
  
  
# region Создание GUI.  
if \_\_name\_\_ == '\_\_main\_\_':  
 window = Tk()  
 window.title('Ханойские башни на максималках.')  
  
 # Создаем канвас  
 canvas = Canvas(window, width=CANVAS\_SIZE[0], height=CANVAS\_SIZE[1], bg='pink')  
  
 # Рисуем шпиндели  
 for \_pillar\_position in range(1, PILLARS\_COUNT + 1):  
 \_pillar\_number = PILLARS\_COUNT + 1 - \_pillar\_position  
 # Задаем начальное количество дисков на шпинделе по ID  
 \_disks\_on\_pillar\_count = int(ID[\_pillar\_number - 1])  
 IterationsManager.current\_disks\_positions\_by\_pillar()[\_pillar\_number] = []  
  
 # Создаем диски и сохраняем их текущую позицию как начальную  
 for \_i in range(0, \_disks\_on\_pillar\_count):  
 \_disk\_width = \_pillar\_number \* 10 + (\_disks\_on\_pillar\_count - \_i)  
  
 # Создаем конкретный диск и добавляем в историю. Цвет диска выбирается случайно  
 IterationsManager.current\_disks\_positions\_by\_pillar()[\_pillar\_number].append(  
 Disk("#" + ("%06x" % random.randint(0, 16777215)), \_disk\_width))  
  
 \_canvas\_width = PILLARS\_OFFSET \* \_pillar\_position  
  
 # Сам шпиндель  
 canvas.create\_rectangle(\_canvas\_width,  
 CANVAS\_SIZE[1] - PILLAR\_HEIGHT,  
 \_canvas\_width + PILLAR\_WIDTH,  
 CANVAS\_SIZE[1],  
 fill="brown")  
  
 # Подложка под шпинделями  
 canvas.create\_rectangle(0, CANVAS\_SIZE[1] - STAND\_HEIGHT, CANVAS\_SIZE[0], CANVAS\_SIZE[1], fill="brown", )  
  
 # Нумеруем шпиндели от 8 до 1 слева направо  
 for \_pillar\_position in range(1, PILLARS\_COUNT + 1):  
 \_pillar\_number = PILLARS\_COUNT + 1 - \_pillar\_position  
 \_canvas\_width = PILLARS\_OFFSET \* \_pillar\_position  
 canvas.create\_text((\_canvas\_width + PILLAR\_WIDTH / 2, CANVAS\_SIZE[1] - 7),  
 text=\_pillar\_number, font=("Times New Roman", 12, 'bold'), fill='yellow')  
  
 canvas.grid(row=2, column=1, padx=4, pady=4, rowspan=7, columnspan=8)  
  
 error\_label\_text = StringVar(value='')  
 error\_label = Label(window, width=5, font=("Times New Roman", 14), fg='red', textvariable=error\_label\_text)  
 error\_label.grid(row=1, column=1, sticky='EW', padx=10, pady=20, columnspan=8)  
  
 # Создаем поля для ввода промежуточных итераций и кнопки под ними  
 \_iterations = 0  
 for \_i in range(3, 7):  
 \_text = StringVar(value=DEFAULT\_PROGRESS\_PERCENTS[\_iterations])  
 \_iterations += 1  
 \_progress\_input = Entry(window, width=5, font=("Times New Roman", 14), textvariable=\_text)  
 \_progress\_input.grid(row=9, column=\_i, sticky='EW', padx=10, pady=2)  
  
 \_progress\_select\_bth = Button(window, text=f'П.{\_iterations}', width=5)  
 # При нажатии на кнопку вызываем функцию show\_intermediate\_iteration, передавая в нее ячейку с введенным прогрессом  
 \_progress\_select\_bth.bind("<Button-1>",  
 lambda event, arg=\_progress\_input: show\_intermediate\_iteration(event, arg))  
 \_progress\_select\_bth.grid(row=10, column=\_i, sticky='EW', padx=10, pady=2)  
  
 to\_start\_button = Button(window, text='Начало', width=15)  
 to\_start\_button.bind("<Button-1>", show\_start\_of\_iterations)  
 to\_start\_button.grid(row=9, column=2, sticky='EW', padx=10, pady=2, rowspan=2)  
  
 to\_end\_button = Button(window, text='Окончание', width=15)  
 to\_end\_button.bind("<Button-1>", show\_end\_of\_iterations)  
 to\_end\_button.grid(row=9, column=7, sticky='EW', padx=10, pady=2, rowspan=2)  
  
 previous\_button = Button(window, text='<<', width=5)  
 previous\_button.bind("<Button-1>", show\_previous\_iteration)  
 previous\_button.grid(row=11, column=3, sticky='EW', padx=10, pady=20)  
  
 next\_button = Button(window, text='>>', width=5)  
 next\_button.bind("<Button-1>", show\_next\_iteration)  
 next\_button.grid(row=11, column=4, sticky='EW', padx=10, pady=20)  
  
 current\_iteration\_text = StringVar(value=0)  
 \_current\_iteration\_label = Label(window, width=10, font=("Times New Roman", 14),  
 textvariable=current\_iteration\_text)  
 \_current\_iteration\_label.grid(row=1, column=2, sticky='N')  
 \_current\_iteration\_label\_text = Label(window, text='Итерация:', font=("Times New Roman", 14))  
 \_current\_iteration\_label\_text.grid(row=1, column=1, sticky='N')  
  
 \_start\_time = datetime.datetime.now()  
  
 # Запускаем алгоритм для расчета и сохранения всех движений дисков из начального состояния в конечное  
 move\_all\_disks\_and\_write\_history()  
  
 # Тест 1: 0.171038 sec  
 # Тест 2: 0.083018 sec  
 # Тест 3: 0.08602 sec  
 print('Elapsed seconds on loading', (datetime.datetime.now() - \_start\_time).total\_seconds())  
  
 # Показываем начальное состояние дисков при старте программы  
 show\_start\_of\_iterations(None)  
  
 print('Total iterations', total\_disks\_moves)  
 window.mainloop()