МУНИЦИПАЛЬНОЕ БЮДЖЕТНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

**ГИМНАЗИЯ ИМ. АКАДЕМИКА Н.Г. БАСОВА ПРИ ВОРОНЕЖСКОМ ГОСУДАРСТВЕННОМ УНИВЕРСИТЕТЕ**

**ИТОГОВЫЙ ИНДИВИДУАЛЬНЫЙ ПРОЕКТ**

ПО ИНФОРМАТИКЕ

«**РЕШЕНИЕ МАТЕМАТИЧЕСКИХ УРАВНЕНИЙ (GUI-ПРИЛОЖЕНИЕ)**»

**Выполнил:** Карташов Глеб

Борисович,

ученик 10 класса «Г»

**Руководитель:** Гончарова Марина

Алексеевна,

учитель информатики

Воронеж

**2021-2022 учебный год**

Оглавление

[ВВЕДЕНИЕ 3](#_Toc103810305)

[ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 4](#_Toc103810306)

[Выбранный инструментал 4](#_Toc103810307)

[Реализуемые функции 4](#_Toc103810308)

[Положительные аспекты реализации проекта 4](#_Toc103810308)

[ПРАКТИЧЕСКАЯ ЧАСТЬ 5](#_Toc103810310)

[Описание технологического процесса 5](#_Toc103810311)

[Описание этапов работы над проектом 5](#_Toc103810312)

[Особенности разработки и разработанного ПО 5](#_Toc103810312)

[ЗАКЛЮЧЕНИЕ 7](#_Toc103810313)

[ЛИТЕРАТУРА 8](#_Toc103810314)

[ПРИЛОЖЕНИЕ 9](#_Toc103810314)

ВВЕДЕНИЕ

В сети имеется множество различных калькуляторов и программ для решения различных уравнений и для работы с математическими сущностями. Для работы с ними нужен постоянный доступ к интернету, к тому же, чтобы их найти нужно некоторое время. Ко всему этому добавляется факт того, что это не очень практично и не очень удобно. Во время работы за ПК, да и при решении различных задач приходится постоянно прибегать к различной сложности математическим вычислениям. Данное приложение поможет сэкономить время и облегчить жизнь решающему задачи, например, по математике, физике или информатике.

Таким образом, предпосылками для данного проекта явились повышение удобства пользования математическим аппаратом при решении задач по физике и другим предметам.

Цели работы: 1) Разработка приложения для быстрой работы с математическими объектами (многофункционального калькулятора). 2) Приобретение практических навыков в области разработки ПО, а также его тестирования и отладки.

Задачи:

1. Разработать техническое задание, пользовательский интерфейс приложения, его структуру и список реализуемых функций.
2. Выбрать язык программирования, среду и необходимые для реализации программы технологии.
3. Провести тестирование и отладку компонентов программы.
4. Собрать программу в исполняемый файл.
5. Оформить презентационную часть работы.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Выбранный инструментал

Для реализации продукта был выбран язык программирования Python, из-за быстроты написания ПО на нём. В качестве среды программирования была выбрана PyCharm IDE, из-за удобства написания, хранения и версионирования кода, тестирования, отладки и запуска программы в ней. Математические и другие функции разрабатывались с использованием библиотек sympy, matplotlib, math. GUI или пользовательский интерфейс был реализован при помощи фреймворка PyQt. Версионирование велось при помощи Git, хранился код на моём локальном ПК и в репозитории на GitHub.

## Реализуемые функции

В продукте были реализованы следующие функции: вычисление НОД и НОК чисел, проверка числа на простоту и его факторизация (разложение на множители), возведение в степень, извлечение корня, сумма натуральных чисел на заданном отрезке (от m до n), решение квадратных, биквадратных и линейных уравнений, вычисление количества перестановок, сочетаний и размещений, сумма арифметических и геометрических прогрессий.

## Положительные аспекты реализации проекта

Были применены на практике знания в математике, навыки функционального и объектно-ориентированного программирования, учтены некоторые особенности применения математического аппарата.

ПРАКТИЧЕСКАЯ ЧАСТЬ

## Описание технологического процесса

Наш технологический процесс можно обобщить до четырёх этапов:

1. Выбор идеи проекта, обоснование его целей, задач и актуальности
2. Сбор необходимой информации и материалов
3. Разработка ПО
4. Оформление работы

## Описание этапов работы над проектом

Я долго думал над выбором идеи для проекта. Когда тема была выбрана, были собраны материалы (см. далее), которые я принялся изучать. Далее был продуман план реализации проекта и обозначены сроки. Затем было продумано построение приложения. После, начата разработка. Сначала прописывались общие классы (main.py, mainwindow.py, algebra.py), затем реализовывались классы функций, каждый в своём файле. После было проведено тестирование программы на ошибки. Потом я собрал всё в исполняемый .exe файл, что стало далеко не самой простой задачей. Уже после была начата работа над презентационной частью проекта.

**Особенности разработки и разработанного ПО**

1. Архитектурная проработанность (я изучил документацию PyQt и нашёл способ разнесения классов по файлам). Также с целью сокращения написанного кода и повышения читабельности был реализован один метод открывающий родительское и дочернее окно в каждом классе. Чтобы этот метод каждый раз не повторялся он был вынесен в базовый класс от которого наследовались другие. Данные архитектурные решения были необходимы, т.к. проект, по объёму занимаемого кода довольно большой и будет дальше расширяться.
2. Для работы с математическими объектами, а не только числами была выбрана библиотека sympy, однако где-то применялись и функции стандартной библиотеки math
3. С помощью библиотеки matplotlib можно преобразовывать математическую ТЕХ-нотацию сгенерированную с помощью метода sympy.latex в красивое изображение с расширением .svg. Т.к. потребность в генерации этого изображения есть практически для каждой функции, то преобразование ТEX-нотации в картинку было вынесено в отдельную функцию classes.text2svg
4. Для реализации GUI был выбран фреймворк PyQt, т.к. по нему есть много материала в интернете и отличная документация по очень обширному его функционалу.
5. Чтобы постоянно не переписывать названия требуемых модулей, вынес их в modules.py.
6. Данный многофункциональный калькулятор будет поддерживаться и дальше, в связи с чем использование принципов ООП для повышения удобства написания кода, удобного добавления нового функционала и более понятной архитектуры проекта, очень кстати.
7. Калькулятор имеет немалое количество функций, каждая со своими особенностями и ограничениями. Важно чтобы они работали для всех вводимых значений и не выдавали ошибок. Поэтому на первом месте для меня стояла реализация необходимого функционала в пренебрежение красоте интерфейса.
8. Изначально планировалось сделать функционал по решению уравнений, но затем было принято решение расширить проект и добавить другие алгебраические функции.
9. Ввиду ограниченности времени геометрические, физические, тригонометрические, информатические и другие функции реализованы не были, однако, благодаря архитектуре приложения они легко могут быть добавлены в проект позже.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данного проекта:

1. Изучили информацию по необходимым библиотекам в результате чего расширили знания в области программирования GUI приложений и возможностей в реализации математического функционала.
2. Разработали приложение (многофункциональный калькулятор) для быстрой работы с математическими объектами.
3. Приобрели практические навыки в области разработки ПО, а также его тестирования и отладки.

Также стоит сказать, что данный проект будет дальше дополняться различным функционалом, т.к. для меня лично это оказалось очень интересным занятием.

ЛИТЕРАТУРА

1. <https://doc.qt.io/qtforpython/> - документация фреймворка PyQt
2. <https://docs.sympy.org/latest/index.html> - документация библиотеки sympy
3. <https://matplotlib.org/stable/index.html> документация библиотеки matplotlib
4. <https://docs.python.org/3> - документация по языку Python
5. <https://stackoverflow.com/questions/32035251/displaying-latex-in-pyqt-pyside-qtablewidget> - отображение ТЕХ-нотации в PyQt
6. <https://habr.com/ru/post/325626/> - статья о том, как собрать проект на Рython и PyQt под Windows
7. <https://github.com/pyinstaller/pyinstaller/issues/2919> - исправление ошибки возникающей при сборке проекта

**ПРИЛОЖЕНИЕ**

Структура кода

main.py – запускаемый файл

mainwindow.py – главное окно

modules.py – необходимые модули и библиотеки

algebra.py – алгебраические функции

images/ – папка с изображениями

classes/

|---tex2svg.py

|---algebra/

|---arithmetic.py

|---average\_values.py

|---combinatorics.py

|---equations.py

|---progressions.py

|---arithmetic\_\_/

|---exponentiation.py

|---gcd\_and\_lmn.py

|---root\_extraction.py

|---simplicity\_and\_factorization.py

|---sum\_m\_n.py

|---average\_values\_\_/

|---arithm.py

|---geom.py

|---harmon.py

|---quadr.py

|---combinatorics\_\_/

|---combinations.py

|---permutations.py

|---placements.py

|---equations\_\_/

|---biquadratic.py

|---cubic\_equation.py

|---inequalities.py

|---linear\_equation.py

|---quadratic\_equations.py

|---progressions\_\_/

|---arithmetic.py

|---geometric.py

**Код**

**main.py**

import sys

from mainwindow import MainWindow

from PyQt5.QtWidgets import QApplication

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec\_())

**modules.py**

from PyQt5.QtWidgets import \*

from PyQt5.QtGui import \*

from PyQt5.QtCore import \*

import sympy, math

from PyQt5.QtSvg import QSvgWidget

from sympy.printing.latex import latex

from classes.tex2svg import tex2svg

**mainwindow.py**

from modules import \*

from algebra import Algebra

from geometry import Geometry

from trigonometry import Trigonometry

from informatics import Informatics

from physics import Physics

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setWindowTitle("Многофункциональный калькулятор")

self.resize(910, 850)

self.setWindowIcon(QIcon('images/background.jpg'))

algebra\_button = QPushButton("Алгебра", self)

algebra\_button.setGeometry(QRect(250, 130, 280, 100))

algebra\_button.clicked.connect(lambda: self.open\_sub\_window(Algebra))

geometry\_button = QPushButton("Геометрия", self)

geometry\_button.setGeometry(QRect(250, 230, 280, 100))

geometry\_button.clicked.connect(lambda: self.open\_sub\_window(Geometry))

trigonometry\_button = QPushButton("Тригонометрические функции", self)

trigonometry\_button.setGeometry(QRect(250, 330, 280, 100))

trigonometry\_button.clicked.connect(lambda: self.open\_sub\_window(Trigonometry))

informatics\_button = QPushButton("Информатика", self)

informatics\_button.setGeometry(QRect(250, 430, 280, 100))

informatics\_button.clicked.connect(lambda: self.open\_sub\_window(Informatics))

physics\_button = QPushButton("Физика", self)

physics\_button.setGeometry(QRect(250, 530, 280, 100))

physics\_button.clicked.connect(lambda: self.open\_sub\_window(Physics))

menu\_label = QLabel("Меню", self)

menu\_label.setGeometry(QRect(250, 60, 50, 100))

font = QFont()

font.setPointSize(13)

self.setFont(font)

def open\_sub\_window(self, window\_type):

self.sub\_window = window\_type(self)

self.sub\_window.show()

self.close()

**geometry.py**

from modules import \*

class Geometry(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Геометрия")

self.resize(910, 850)

menu\_button = QPushButton("В меню", self)

menu\_button.setGeometry(QRect(250, 20, 280, 100))

menu\_button.clicked.connect(self.open\_parent\_window)

squares\_of\_figures\_button = QPushButton("Площади фигур", self)

squares\_of\_figures\_button.setGeometry(QRect(250, 130, 280, 100))

squares\_of\_figures\_button.clicked.connect(lambda: self.open\_sub\_window(Squares\_Of\_Figures))

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

def open\_sub\_window(self, window\_type):

self.sub\_window = window\_type(self)

self.sub\_window.show()

self.close()

class Squares\_Of\_Figures(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Площади фигур")

self.resize(910, 850)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

**algebra.py**

from modules import \*

from classes.algebra.average\_values import Average\_Values

from classes.algebra.arithmetic import Arithmetic

from classes.algebra.equations import Equations

from classes.algebra.progressions import Progressions

from classes.algebra.combinatorics import Combinatorics

from classes.algebra.calculus import Calculus

from classes.algebra.graphic import Graphic

class Algebra(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Алгебра")

self.resize(910, 850)

menu\_button = QPushButton("Меню", self)

menu\_button.setGeometry(QRect(250, 20, 280, 100))

menu\_button.clicked.connect(self.open\_parent\_window)

average\_values\_button = QPushButton("Средние величины", self)

average\_values\_button.setGeometry(QRect(250, 120, 280, 100))

average\_values\_button.clicked.connect(lambda: self.open\_sub\_window(Average\_Values))

arithmetic\_button = QPushButton("Арифметика", self)

arithmetic\_button.setGeometry(QRect(250, 220, 280, 100))

arithmetic\_button.clicked.connect(lambda: self.open\_sub\_window(Arithmetic))

equations\_button = QPushButton("Уравнения и неравенства", self)

equations\_button.setGeometry(QRect(250, 320, 280, 100))

equations\_button.clicked.connect(lambda: self.open\_sub\_window(Equations))

progressions\_button = QPushButton("Прогрессии", self)

progressions\_button.setGeometry(QRect(250, 420, 280, 100))

progressions\_button.clicked.connect(lambda: self.open\_sub\_window(Progressions))

combinatorics\_button = QPushButton("Комбинаторика", self)

combinatorics\_button.setGeometry(QRect(250, 520, 280, 100))

combinatorics\_button.clicked.connect(lambda: self.open\_sub\_window(Combinatorics))

calculus\_button = QPushButton("Немного из матанализа", self)

calculus\_button.setGeometry(QRect(530, 120, 280, 100))

calculus\_button.clicked.connect(lambda: self.open\_sub\_window(Calculus))

graphic\_button = QPushButton("Построение графика функции", self)

graphic\_button.setGeometry(QRect(530, 220, 280, 100))

graphic\_button.clicked.connect(lambda: self.open\_sub\_window(Graphic))

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

def open\_sub\_window(self, window\_type):

self.sub\_window = window\_type(self)

self.sub\_window.show()

self.close()

**trigonometry.py**

from modules import \*

class Trigonometry(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Тригонометрические функции")

self.resize(910, 850)

menu\_button = QPushButton("В меню", self)

menu\_button.setGeometry(QRect(250, 20, 280, 100))

menu\_button.clicked.connect(self.open\_parent\_window)

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

def open\_sub\_window(self, window\_type):

self.sub\_window = window\_type(self)

self.sub\_window.show()

self.close()

**-сlasses-text2svg.py**

from io import BytesIO

import matplotlib.pyplot as plt

plt.rc('mathtext', fontset='cm')

def tex2svg(formula, fontsize=12, dpi=300):

fig = plt.figure(figsize=(0.01, 0.01))

fig.text(0, 0, r'${}$'.format(formula), fontsize=fontsize)

output = BytesIO()

fig.savefig(output, dpi=dpi, transparent=True, format='svg',

bbox\_inches='tight', pad\_inches=0.0)

plt.close(fig)

output.seek(0)

return output.read()

**classes.algebra**

from modules import \*

from classes.algebra.progressions\_\_.arithmetic import Arithmetic

from classes.algebra.progressions\_\_.geometric import Geometric

class Progressions(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Прогрессии")

self.resize(910, 850)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

button = QPushButton("Сумма арифметической прогрессии", self)

button.setGeometry(QRect(250, 120, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Arithmetic))

button = QPushButton("Сумма геометрической прогрессии", self)

button.setGeometry(QRect(250, 220, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Geometric))

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

def open\_sub\_window(self, window\_type):

self.sub\_window = window\_type(self)

self.sub\_window.show()

self.close()

from modules import \*

from classes.algebra.equations\_\_.quadratic\_equations import Quadratic\_Equations

from classes.algebra.equations\_\_.biquadratic import Biquadratic

from classes.algebra.equations\_\_.linear\_equation import Linear

from classes.algebra.equations\_\_.cubic\_equation import Cubic\_Equation

from classes.algebra.equations\_\_.system\_of\_linear\_equations import System

from classes.algebra.equations\_\_.inequalities import Inequalities

class Equations(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Уравнения")

self.resize(910, 850)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

quadratic\_equations\_button = QPushButton("Квадратные уравнения", self)

quadratic\_equations\_button.setGeometry(QRect(250, 120, 280, 100))

quadratic\_equations\_button.clicked.connect(lambda: self.open\_sub\_window(Quadratic\_Equations))

button = QPushButton("Биквадратные уравнения", self)

button.setGeometry(QRect(250, 220, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Biquadratic))

button = QPushButton("Линейные уравнения", self)

button.setGeometry(QRect(250, 320, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Linear))

"""button = QPushButton("Кубические уравнения", self)

button.setGeometry(QRect(250, 420, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Cubic\_Equation))"""

"""button = QPushButton("Система линейных уравнений", self)

button.setGeometry(QRect(250, 520, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(System))

button = QPushButton("Неравенства", self)

button.setGeometry(QRect(530, 120, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Inequalities))"""

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

def open\_sub\_window(self, window\_type):

self.sub\_window = window\_type(self)

self.sub\_window.show()

self.close()

from modules import \*

from classes.algebra.combinatorics\_\_.placements import Placements

from classes.algebra.combinatorics\_\_.combinations import Combinations

from classes.algebra.combinatorics\_\_.permutations import Permutations

class Combinatorics(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Комбинаторика")

self.resize(910, 850)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

button = QPushButton("Перестановки", self)

button.setGeometry(QRect(250, 120, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Permutations))

button = QPushButton("Размещения", self)

button.setGeometry(QRect(250, 220, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Placements))

button = QPushButton("Сочетания", self)

button.setGeometry(QRect(250, 320, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Combinations))

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

def open\_sub\_window(self, window\_type):

self.sub\_window = window\_type(self)

self.sub\_window.show()

self.close()

from modules import \*

from classes.algebra.arithmetic\_\_.gcd\_and\_lmn import GCD\_and\_LMN

from classes.algebra.arithmetic\_\_.simplicity\_and\_factorization import Simplicity\_And\_Factorization

from classes.algebra.arithmetic\_\_.exponentiation import Exponentiation

from classes.algebra.arithmetic\_\_.root\_extraction import Root\_Extraction

from classes.algebra.arithmetic\_\_.sum\_m\_n import Sum\_M\_N

class Arithmetic(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Арифметика")

self.resize(910, 850)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

button = QPushButton("НОД и НОК", self)

button.setGeometry(QRect(250, 120, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(GCD\_and\_LMN))

button = QPushButton("Проверка числа на простоту и его факторизация", self)

button.setGeometry(QRect(250, 220, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Simplicity\_And\_Factorization))

button = QPushButton("Возведение в степень", self)

button.setGeometry(QRect(250, 320, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Exponentiation))

button = QPushButton("Извлечение корня", self)

button.setGeometry(QRect(250, 420, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Root\_Extraction))

button = QPushButton("Сумма натуральных чисел от m до n", self)

button.setGeometry(QRect(250, 520, 280, 100))

button.clicked.connect(lambda: self.open\_sub\_window(Sum\_M\_N))

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

def open\_sub\_window(self, window\_type):

self.sub\_window = window\_type(self)

self.sub\_window.show()

self.close()

from modules import \*

class Exponentiation(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Возведение в степень")

self.resize(910, 850)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

self.a = QLineEdit(self)

self.a.setGeometry(QRect(25, 190, 50, 50))

self.a.setValidator(QDoubleValidator(0.0, 10000.0, 6))

self.b = QLineEdit(self)

self.b.setGeometry(QRect(75, 190, 50, 50))

self.b.setValidator(QDoubleValidator(0.0, 100.0, 6))

root\_button = QPushButton("Показать степень (сначала возводимое число, потом степень)", self)

root\_button.setGeometry(QRect(250, 130, 280, 100))

root\_button.clicked.connect(self.root)

self.svg\_root = QSvgWidget(self)

self.svg\_root.setGeometry(QRect(530, 150, 100, 20))

def root(self):

if self.a.text() != '' and self.b.text() != '':

a = self.a.text()

b = self.b.text()

a = a.replace(",", ".")

b = b.replace(",", ".")

a = float(a)

b = float(b)

self.svg\_root.load(tex2svg(latex(a\*\*b)))

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

from modules import \*

class GCD\_and\_LMN(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("НОД и НОК")

self.resize(910, 850)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

self.a\_argument = QLineEdit(self)

self.a\_argument.setGeometry(QRect(25, 190, 50, 50))

self.a\_argument.setValidator(QIntValidator(1, 1000000000, self))

self.b\_argument = QLineEdit(self)

self.b\_argument.setGeometry(QRect(75, 190, 50, 50))

self.b\_argument.setValidator(QIntValidator(1, 1000000000, self))

"""self.n = QLineEdit(self)

self.n.setGeometry(QRect(75, 350, 50, 50))

self.n.setValidator(QIntValidator(1, 20, self))

n\_button = QPushButton("Генерация окон", self)

n\_button.setGeometry(QRect(125, 350, 280, 100))

n\_button.clicked.connect(self.nn)"""

gcd\_button = QPushButton("НОД", self)

gcd\_button.setGeometry(QRect(250, 130, 280, 100))

gcd\_button.clicked.connect(self.gcd)

lmn\_button = QPushButton("НОК", self)

lmn\_button.setGeometry(QRect(250, 230, 280, 100))

lmn\_button.clicked.connect(self.lmn)

self.svg\_gcd = QSvgWidget(self)

self.svg\_gcd.setGeometry(QRect(530, 150, 100, 50))

self.svg\_lmn = QSvgWidget(self)

self.svg\_lmn.setGeometry(QRect(530, 250, 100, 50))

"""нод и нок для нескольких чисел, считывание с окошек итд"""

"""def nn(self):

for i in range(1, int(self.n.text()) + 1):

s = 's' + str(i)

self.s = QLineEdit(self)

self.s.setGeometry(QRect(75 + (i-1)\*50, 450, 50, 50))

self.s.setValidator(QIntValidator(1, 10, self))

self.s.show()

"""

def gcd(self):

if self.a\_argument.text() != '' and self.b\_argument.text() != '':

a = int(self.a\_argument.text())

b = int(self.b\_argument.text())

self.svg\_gcd.load(tex2svg(latex(sympy.gcd(a, b))))

def lmn(self):

if self.a\_argument.text() != '' and self.b\_argument.text() != '':

a = int(self.a\_argument.text())

b = int(self.b\_argument.text())

self.svg\_lmn.load(tex2svg(latex(a\*b/sympy.gcd(a, b))))

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

from modules import \*

class Root\_Extraction(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Извлечение корня")

self.resize(910, 850)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

self.a\_argument = QLineEdit(self)

self.a\_argument.setGeometry(QRect(25, 190, 50, 50))

self.a\_argument.setValidator(QDoubleValidator(0.0, 1000000000.0, 6))

"""notation=QDoubleValidator.StandardNotation"""

root\_button = QPushButton("Показать корень", self)

root\_button.setGeometry(QRect(250, 130, 280, 100))

root\_button.clicked.connect(self.root)

self.svg\_root = QSvgWidget(self)

self.svg\_root.setGeometry(QRect(530, 150, 100, 20))

def root(self):

if self.a\_argument.text() != '':

a = self.a\_argument.text()

a = a.replace(",", ".")

a = float(a)

self.svg\_root.load(tex2svg(latex(sympy.sqrt(a))))

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

from modules import \*

from sympy import expand, symbols, factor

class Simplicity\_And\_Factorization(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Проверка числа на простоту и его факторизация")

self.resize(910, 850)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

self.number = QLineEdit(self)

self.number.setGeometry(QRect(25, 190, 50, 50))

self.number.setValidator(QIntValidator(1, 1000000000, self))

prime\_btn = QPushButton("Число простое?", self)

prime\_btn.setGeometry(QRect(250, 130, 280, 100))

prime\_btn.clicked.connect(self.IsPrime)

fact\_btn = QPushButton("Факторизация числа", self)

fact\_btn.setGeometry(QRect(250, 230, 280, 100))

fact\_btn.clicked.connect(self.Factor)

self.svg\_fact = QSvgWidget(self)

self.svg\_fact.setGeometry(QRect(530, 270, 100, 50))

self.prime\_label = QLabel(self)

self.prime\_label.setGeometry(QRect(530, 150, 100, 50))

def Factor(self):

if self.number.text() != "":

n = int(self.number.text())

Ans = []

d = 2

while d \* d <= n:

if n % d == 0:

Ans.append(d)

n //= d

else:

d += 1

if n > 1:

Ans.append(n)

listik = set(Ans)

s = []

for i in listik:

s.append(latex(factor(expand(symbols(str(i)) \*\* (Ans.count(i))))))

self.svg\_fact.load(tex2svg("\cdot".join(s)))

def IsPrime(self):

if self.number.text() != "":

n = int(self.number.text())

d = 2

while d \* d <= n and n % d != 0:

d += 1

if d \* d > n:

self.prime\_label.setText(" ДА")

else:

self.prime\_label.setText(" Нет, оно составное")

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

from modules import \*

class Sum\_M\_N(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Сумма натуральных чисел от m до n")

self.resize(910, 850)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

self.m = QLineEdit(self)

self.m.setGeometry(QRect(25, 190, 50, 50))

self.m.setValidator(QIntValidator(1, 1000000000))

self.n = QLineEdit(self)

self.n.setGeometry(QRect(75, 190, 50, 50))

self.n.setValidator(QIntValidator(1, 1000000000))

root\_button = QPushButton("Сумма чисел от m до n", self)

root\_button.setGeometry(QRect(250, 130, 280, 100))

root\_button.clicked.connect(self.root)

self.svg\_root = QSvgWidget(self)

self.svg\_root.setGeometry(QRect(530, 150, 100, 20))

def root(self):

if self.m.text() != '' and self.n.text() != '':

m = int(self.m.text())

n = int(self.n.text())

if m <= n:

self.svg\_root.load(tex2svg(latex(int((m+n)\*(n-m+1)/2))))

else:

self.svg\_root.load(tex2svg(latex(int((m + n) \* (m - n + 1) / 2))))

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

import math

from modules import \*

class Combinations(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Сочетания")

self.resize(910, 850)

x\_2\_pixmap = QPixmap("./images/soch.jpg")

x\_pixmap = QPixmap("./images/sochet.jpg")

x\_2 = QLabel(self)

x = QLabel(self)

plus1 = QLabel(self)

plus2 = QLabel(self)

x\_2.setGeometry(QRect(50, 190, 1000, 500))

x.setGeometry(QRect(400, 100, 500, 200))

plus1.setGeometry(QRect(70, 190, 50, 50))

plus2.setGeometry(QRect(150, 190, 50, 50))

x\_2.setPixmap(x\_2\_pixmap)

x.setPixmap(x\_pixmap)

plus1.setText("n: ")

plus2.setText("k: ")

display\_solution\_button = QPushButton("Показать количество сочетаний из n элементов по k", self)

display\_solution\_button.setGeometry(QRect(530, 20, 280, 100))

display\_solution\_button.clicked.connect(self.display\_solution)

self.b\_argument = QLineEdit(self)

self.b\_argument.setGeometry(QRect(90, 190, 50, 50))

self.b\_argument.setValidator(QIntValidator(1, 1000000000, self))

self.c\_argument = QLineEdit(self)

self.c\_argument.setGeometry(QRect(170, 190, 50, 50))

self.c\_argument.setValidator(QIntValidator(1, 1000000000, self))

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

self.svg = QSvgWidget(self)

self.svg.setGeometry(QRect(10, 10, 200, 100))

def display\_solution(self):

if self.b\_argument.text() != '' and int(self.b\_argument.text()) != 0 and self.c\_argument.text() != '':

n = int(self.b\_argument.text())

k = int(self.c\_argument.text())

self.svg.load(tex2svg(latex(math.factorial(n)//(math.factorial(k)\*math.factorial(n-k)))))

self.svg.show()

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

from modules import \*

class Permutations(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Перестановки")

self.resize(910, 850)

plus1 = QLabel(self)

plus1.setGeometry(QRect(70, 190, 50, 50))

plus1.setText("n: ")

x\_2\_pixmap = QPixmap("./images/perest.jpg")

x\_2 = QLabel(self)

x\_2.setGeometry(QRect(50, 190, 1000, 500))

x\_2.setPixmap(x\_2\_pixmap)

display\_solution\_button = QPushButton("Показать количество перестановок из n элементов", self)

display\_solution\_button.setGeometry(QRect(530, 20, 280, 100))

display\_solution\_button.clicked.connect(self.display\_solution)

self.b\_argument = QLineEdit(self)

self.b\_argument.setGeometry(QRect(90, 190, 50, 50))

self.b\_argument.setValidator(QIntValidator(1, 1000000000, self))

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

self.svg = QSvgWidget(self)

self.svg.setGeometry(QRect(10, 10, 200, 100))

def display\_solution(self):

if self.b\_argument.text() != '':

n = int(self.b\_argument.text())

self.svg.load(tex2svg(latex(math.factorial(n))))

self.svg.show()

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

from modules import \*

class Placements(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Размещения")

self.resize(910, 850)

plus1 = QLabel(self)

plus2 = QLabel(self)

plus1.setGeometry(QRect(70, 190, 50, 50))

plus2.setGeometry(QRect(150, 190, 50, 50))

plus1.setText("n: ")

plus2.setText("k: ")

x\_2\_pixmap = QPixmap("./images/razm.jpg")

x\_2 = QLabel(self)

x\_2.setGeometry(QRect(50, 190, 1000, 500))

x\_2.setPixmap(x\_2\_pixmap)

display\_solution\_button = QPushButton("Показать количество различных размещений из n элементов по k", self)

display\_solution\_button.setGeometry(QRect(530, 20, 350, 100))

display\_solution\_button.clicked.connect(self.display\_solution)

self.b\_argument = QLineEdit(self)

self.b\_argument.setGeometry(QRect(90, 190, 50, 50))

self.b\_argument.setValidator(QIntValidator(1, 1000000000, self))

self.c\_argument = QLineEdit(self)

self.c\_argument.setGeometry(QRect(170, 190, 50, 50))

self.c\_argument.setValidator(QIntValidator(1, 1000000000, self))

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

self.svg = QSvgWidget(self)

self.svg.setGeometry(QRect(10, 10, 200, 100))

def display\_solution(self):

if self.b\_argument.text() != '' and int(self.b\_argument.text()) != 0 and self.c\_argument.text() != '':

n = int(self.b\_argument.text())

k = int(self.c\_argument.text())

self.svg.load(tex2svg(latex(math.factorial(n)//math.factorial(n-k))))

self.svg.show()

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

from modules import \*

class Biquadratic(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Биквадратные уравнения")

self.resize(910, 850)

x\_4\_pixmap = QPixmap("./images/x^4.svg")

x\_2\_pixmap = QPixmap("./images/x^2.svg")

plus1\_pixmap = QPixmap("./images/plus.svg")

plus2\_pixmap = QPixmap("./images/plus.svg")

eq0\_pixmap = QPixmap("./images/eq0.svg")

x\_4 = QLabel(self)

x\_2 = QLabel(self)

plus1 = QLabel(self)

plus2 = QLabel(self)

eq0 = QLabel(self)

x\_4.setGeometry(QRect(50, 190, 50, 50))

x\_2.setGeometry(QRect(140, 190, 50, 50))

plus1.setGeometry(QRect(70, 190, 50, 50))

plus2.setGeometry(QRect(160, 190, 50, 50))

eq0.setGeometry(QRect(220, 190, 50, 50))

x\_4.setPixmap(x\_4\_pixmap)

x\_2.setPixmap(x\_2\_pixmap)

plus1.setPixmap(plus1\_pixmap)

plus2.setPixmap(plus2\_pixmap)

eq0.setPixmap(eq0\_pixmap)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

display\_solution\_button = QPushButton("Показать решение", self)

display\_solution\_button.setGeometry(QRect(250, 130, 280, 100))

display\_solution\_button.clicked.connect(self.display\_solution)

self.solution\_label = QLabel(self)

self.solution\_label.setGeometry(QRect(50, 50, 500, 100))

self.a\_argument = QLineEdit(self)

self.a\_argument.setGeometry(QRect(0, 190, 50, 50))

self.a\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.b\_argument = QLineEdit(self)

self.b\_argument.setGeometry(QRect(90, 190, 50, 50))

self.b\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.c\_argument = QLineEdit(self)

self.c\_argument.setGeometry(QRect(175, 190, 50, 50))

self.c\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.svg\_1 = QSvgWidget(self)

self.svg\_1.setGeometry(QRect(10, 10, 100, 50))

self.svg\_2 = QSvgWidget(self)

self.svg\_2.setGeometry(QRect(10, 60, 100, 50))

self.svg\_3 = QSvgWidget(self)

self.svg\_3.setGeometry(QRect(110, 10, 100, 50))

self.svg\_4 = QSvgWidget(self)

self.svg\_4.setGeometry(QRect(110, 60, 100, 50))

def display\_solution(self):

if self.a\_argument.text() != '' and self.a\_argument.text() != '0' and self.b\_argument.text() != '' and self.c\_argument.text() != '':

a = int(self.a\_argument.text())

b = int(self.b\_argument.text())

c = int(self.c\_argument.text())

D = b \*\* 2 - 4 \* a \* c

if D < 0:

self.svg\_1.close()

self.svg\_2.close()

self.svg\_3.close()

self.svg\_4.close()

self.solution\_label.setText("РЕШЕНИЙ НЕТ")

self.solution\_label.show()

elif D == 0:

"""solution = sympy.simplify(-b / (2 \* a))"""

t = -b/(2\*a)

self.svg\_3.close()

self.svg\_4.close()

if t >= 0:

x1 = -sympy.sqrt(t)

x2 = sympy.sqrt(t)

self.solution\_label.close()

self.svg\_1.load(tex2svg(latex(x1)))

self.svg\_2.load(tex2svg(latex(x2)))

self.svg\_1.show()

self.svg\_2.show()

else:

self.svg\_1.close()

self.svg\_2.close()

self.solution\_label.setText("РЕШЕНИЙ НЕТ")

self.solution\_label.show()

else:

t\_11 = sympy.simplify((-b + sympy.sqrt(D)) / (2 \* a))

t\_22 = sympy.simplify((-b - sympy.sqrt(D)) / (2 \* a))

"""t\_11 = (-b + math.sqrt(D)) / (2 \* a)

t\_22 = (-b - math.sqrt(D)) / (2 \* a)"""

if t\_11 >= 0 and t\_22 >= 0:

x1 = -sympy.sqrt(t\_11)

x2 = sympy.sqrt(t\_11)

x3 = -sympy.sqrt(t\_22)

x4 = sympy.sqrt(t\_22)

self.solution\_label.close()

self.svg\_1.load(tex2svg(latex(x1)))

self.svg\_2.load(tex2svg(latex(x2)))

self.svg\_3.load(tex2svg(latex(x3)))

self.svg\_4.load(tex2svg(latex(x4)))

self.svg\_1.show()

self.svg\_2.show()

self.svg\_3.show()

self.svg\_4.show()

if t\_22 >= 0 and t\_11 < 0:

x1 = -sympy.sqrt(t\_22)

x2 = sympy.sqrt(t\_22)

self.solution\_label.close()

self.svg\_1.load(tex2svg(latex(x1)))

self.svg\_2.load(tex2svg(latex(x2)))

self.svg\_3.close()

self.svg\_4.close()

self.svg\_1.show()

self.svg\_2.show()

if t\_11 >= 0 and t\_22 < 0:

x1 = -sympy.sqrt(t\_11)

x2 = sympy.sqrt(t\_11)

self.solution\_label.close()

self.svg\_1.load(tex2svg(latex(x1)))

self.svg\_2.load(tex2svg(latex(x2)))

self.svg\_3.close()

self.svg\_4.close()

self.svg\_1.show()

self.svg\_2.show()

if t\_22 < 0 and t\_11 < 0:

self.svg\_1.close()

self.svg\_2.close()

self.svg\_3.close()

self.svg\_4.close()

self.solution\_label.setText("РЕШЕНИЙ НЕТ")

self.solution\_label.show()

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

from modules import \*

class Linear(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Линейные уравнения")

self.resize(910, 850)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

x\_pixmap = QPixmap("./images/x.svg")

plus\_pixmap = QPixmap("./images/plus.svg")

eq\_pixmap = QPixmap("./images/eq.svg")

x = QLabel(self)

plus = QLabel(self)

eq = QLabel(self)

x.setGeometry(QRect(50, 190, 50, 50))

eq.setGeometry(QRect(140, 190, 50, 50))

plus.setGeometry(QRect(70, 190, 50, 50))

x.setPixmap(x\_pixmap)

plus.setPixmap(plus\_pixmap)

eq.setPixmap(eq\_pixmap)

display\_solution\_button = QPushButton("Показать решение", self)

display\_solution\_button.setGeometry(QRect(250, 130, 280, 100))

display\_solution\_button.clicked.connect(self.display\_solution)

self.solution\_label = QLabel(self)

self.solution\_label.setGeometry(QRect(50, 50, 500, 100))

self.a\_argument = QLineEdit(self)

self.a\_argument.setGeometry(QRect(0, 190, 50, 50))

self.a\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.b\_argument = QLineEdit(self)

self.b\_argument.setGeometry(QRect(90, 190, 50, 50))

self.b\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.c\_argument = QLineEdit(self)

self.c\_argument.setGeometry(QRect(170, 190, 50, 50))

self.c\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.svg = QSvgWidget(self)

self.svg.setGeometry(QRect(10, 10, 100, 50))

def display\_solution(self):

if self.a\_argument.text() != '' and int(self.a\_argument.text()) != 0 and self.b\_argument.text() != '' and self.c\_argument.text() != '':

a = int(self.a\_argument.text())

b = int(self.b\_argument.text())

c = int(self.c\_argument.text())

self.svg.load(tex2svg(latex((c-b)/a)))

self.svg.show()

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

from modules import \*

class Quadratic\_Equations(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Квадратные уравнения")

self.resize(910, 850)

x\_2\_pixmap = QPixmap("./images/x^2.svg")

x\_pixmap = QPixmap("./images/x.svg")

plus1\_pixmap = QPixmap("./images/plus.svg")

plus2\_pixmap = QPixmap("./images/plus.svg")

eq0\_pixmap = QPixmap("./images/eq0.svg")

x\_2 = QLabel(self)

x = QLabel(self)

plus1 = QLabel(self)

plus2 = QLabel(self)

eq0 = QLabel(self)

x\_2.setGeometry(QRect(50, 190, 50, 50))

x.setGeometry(QRect(140, 190, 50, 50))

plus1.setGeometry(QRect(70, 190, 50, 50))

plus2.setGeometry(QRect(150, 190, 50, 50))

eq0.setGeometry(QRect(237, 190, 50, 50))

x\_2.setPixmap(x\_2\_pixmap)

x.setPixmap(x\_pixmap)

plus1.setPixmap(plus1\_pixmap)

plus2.setPixmap(plus2\_pixmap)

eq0.setPixmap(eq0\_pixmap)

display\_solution\_button = QPushButton("Показать решение", self)

display\_solution\_button.setGeometry(QRect(250, 130, 280, 100))

display\_solution\_button.clicked.connect(self.display\_solution)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

"""a\_label = QLabel("a", self)

a\_label.setGeometry(QRect(50, 130, 50, 100))

b\_label = QLabel("b", self)

b\_label.setGeometry(QRect(100, 130, 50, 100))

c\_label = QLabel("c", self)

c\_label.setGeometry(QRect(150, 130, 50, 100))"""

self.solution\_label = QLabel(self)

self.solution\_label.setGeometry(QRect(50, 50, 500, 100))

self.a\_argument = QLineEdit(self)

self.a\_argument.setGeometry(QRect(0, 190, 50, 50))

self.a\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.b\_argument = QLineEdit(self)

self.b\_argument.setGeometry(QRect(90, 190, 50, 50))

self.b\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.c\_argument = QLineEdit(self)

self.c\_argument.setGeometry(QRect(170, 190, 50, 50))

self.c\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.svg\_1 = QSvgWidget(self)

self.svg\_1.setGeometry(QRect(10, 10, 100, 50))

self.svg\_2 = QSvgWidget(self)

self.svg\_2.setGeometry(QRect(10, 110, 100, 50))

def display\_solution(self):

if self.a\_argument.text() != '' and self.a\_argument.text() != '0' and self.b\_argument.text() != '' and self.c\_argument.text() != '':

a = int(self.a\_argument.text())

b = int(self.b\_argument.text())

c = int(self.c\_argument.text())

D = b \*\* 2 - 4 \* a \* c

if D < 0:

self.svg\_1.close()

self.svg\_2.close()

self.solution\_label.setText("РЕШЕНИЙ НЕТ")

self.solution\_label.show()

elif D == 0:

solution = sympy.simplify(-b / (2 \* a))

self.svg\_2.close()

self.solution\_label.close()

self.svg\_1.load(tex2svg(latex(solution)))

self.svg\_1.show()

else:

solution\_1 = sympy.simplify((-b + sympy.sqrt(D)) / (2 \* a))

solution\_2 = sympy.simplify((-b - sympy.sqrt(D)) / (2 \* a))

self.solution\_label.close()

self.svg\_1.load(tex2svg(latex(solution\_1)))

self.svg\_1.show()

self.svg\_2.load(tex2svg(latex(solution\_2)))

self.svg\_2.show()

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

from modules import \*

class Arithmetic(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Cумма n первых членов арифметической прогрессии")

self.resize(910, 850)

self.a\_argument = QLineEdit(self)

self.a\_argument.setGeometry(QRect(0, 190, 50, 50))

self.a\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.b\_argument = QLineEdit(self)

self.b\_argument.setGeometry(QRect(90, 190, 50, 50))

self.b\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.c\_argument = QLineEdit(self)

self.c\_argument.setGeometry(QRect(170, 190, 50, 50))

self.c\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.svg\_1 = QSvgWidget(self)

self.svg\_1.setGeometry(QRect(10, 10, 200, 100))

a\_label = QLabel("a1", self)

a\_label.setGeometry(QRect(10, 130, 50, 100))

b\_label = QLabel("d", self)

b\_label.setGeometry(QRect(100, 130, 50, 100))

c\_label = QLabel("n", self)

c\_label.setGeometry(QRect(180, 130, 50, 100))

display\_solution\_button = QPushButton("Показать сумму", self)

display\_solution\_button.setGeometry(QRect(250, 130, 280, 100))

display\_solution\_button.clicked.connect(self.display\_solution)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

def display\_solution(self):

if self.a\_argument.text() != '' and self.b\_argument.text() != '' and self.c\_argument.text() != '':

a1 = int(self.a\_argument.text())

d = int(self.b\_argument.text())

n = int(self.c\_argument.text())

self.svg\_1.load(tex2svg(latex(a1\*n + (d\*n\*(n-1))/2)))

self.svg\_1.show()

def open\_parent\_window(self):

self.parent\_window.show()

self.close()

from modules import \*

class Geometric(QWidget):

def \_\_init\_\_(self, parent\_window):

super().\_\_init\_\_()

self.initUI()

self.parent\_window = parent\_window

def initUI(self):

self.setWindowTitle("Cумма n первых членов геометрической прогрессии")

self.resize(910, 850)

self.a\_argument = QLineEdit(self)

self.a\_argument.setGeometry(QRect(0, 190, 50, 50))

self.a\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.b\_argument = QLineEdit(self)

self.b\_argument.setGeometry(QRect(90, 190, 50, 50))

self.b\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.c\_argument = QLineEdit(self)

self.c\_argument.setGeometry(QRect(170, 190, 50, 50))

self.c\_argument.setValidator(QIntValidator(-1000000000, 1000000000, self))

self.svg\_1 = QSvgWidget(self)

self.svg\_1.setGeometry(QRect(10, 10, 200, 100))

a\_label = QLabel("b1", self)

a\_label.setGeometry(QRect(10, 130, 50, 100))

b\_label = QLabel("q", self)

b\_label.setGeometry(QRect(100, 130, 50, 100))

c\_label = QLabel("n", self)

c\_label.setGeometry(QRect(180, 130, 50, 100))

display\_solution\_button = QPushButton("Показать сумму", self)

display\_solution\_button.setGeometry(QRect(250, 130, 280, 100))

display\_solution\_button.clicked.connect(self.display\_solution)

back\_button = QPushButton("Назад", self)

back\_button.setGeometry(QRect(250, 20, 280, 100))

back\_button.clicked.connect(self.open\_parent\_window)

def display\_solution(self):

if self.a\_argument.text() != '' and self.b\_argument.text() != '' and self.c\_argument.text() != '':

b1 = int(self.a\_argument.text())

q = int(self.b\_argument.text())

n = int(self.c\_argument.text())

if q >= 1 or q <= -1:

self.svg\_1.load(tex2svg(latex(b1\*(q\*\*n - 1)/(q-1))))

else:

self.svg\_1.load(tex2svg(latex(b1/(1-q))))

self.svg\_1.show()

def open\_parent\_window(self):

self.parent\_window.show()

self.close()