Тренажер решения уравнения основной школы

Разработала:

ученица 5"Э" класса

ГБОУ Школа № 1504  
**Сойчик Аливия**

Москва

2025 г.

**СОДЕРЖАНИЕ**

Введение

1. Уравнения в математике
2. Алгоритм решения уравнений
3. Тренажер решения уравнений и его польза
4. Разработка тренажера решения уравнений
   1. Выбор среды разработки
   2. Этапы разработки и разбитие задачи на части
   3. Описание алгоритмов
   4. Тестирование приложения

Заключение

Список источников информации

Приложение 1. Листинг модуля main.py

Приложение 2. Листинг модуля function.py

**Введение**

Мой проект — это практическая и исследовательская работа на тему решения уравнений основной школы.

Летом, подготавливаясь к пятому классу по Петерсон я встретила множество уравнений, решение которых вызвало у меня интерес. Мне стало любопытно можно ли разработать общий алгоритм их решения. Так же было бы не плохо иметь под рукой тренажер решения уравнений, который подскажет новичкам какие шаги нужно выполнить.

Пятиклассникам только начинающим осваивать уравнения будет полезным иметь под рукой тренажер для решения уравнений. Задумка в том, что он не решит за них уравнения, а только подскажет шаги его решения (алгоритм).

**1. Уравнения в математике**

Уравнение — это математическое равенство, в котором неизвестна некая величина, которая обозначается любой латинской буквой. Решить уравнение — это значит найти все его корни, или убедиться, что их нет. Корень уравнения — это число при подстановке которого на место неизвестной величины равенство соблюдается. В рамках данного проекта мы рассмотрим решение простых линейных уравнений.

Для чего же нужны нам уравнения? В школе, они нам полезны прежде всего для решения задач. Составив уравнение по задаче, мы легко найдём что нам требуется. Например, у нас есть задача:

*В двух коробках 28 кг яблок. Во второй коробке на 4 кг больше, чем в первой. Сколько кг яблок в каждой коробке?*

Она легко решается с по мощью уравнения. Предположим, что ***x*** это кг в первой коробке (так как о ней нечего не известно).Про кг во второй нам известно что в ней на 4 кг больше, чем в первой, значит ***x+4***. Так как в двух коробках 28 кг, то получаем уравнение:

***x + (x + 4) = 28***

Решим его:

*2х + 4 = 28*

*2х = 28-4*

*2х = 24*

*x = 24:2*

*x = 12*

Значит в первой коробке 12 кг яблок, а во второй 12+4 = 16 кг .

Так же задача легко решалась бы с помощью уравнения если бы в ней говорилось про большее количество коробок, например три или четыре, мы бы просто составили длинное уравнение и решили его.

Таким образом уравнения нужны для того, чтобы механизировать процесс вычислений. Научившись быстро (по алгоритму) решать уравнения и составив уравнение к какой-либо задаче, мы легко её решим.

**2. Алгоритм решения уравнений**

Алгоритм — это последовательность чётко определённых действий для решения какой либо задачи. Продумав алгоритм решения линейных уравнений мы сможем легко решать их в будущем.

В начальных классах и в начале пятого класса нас учили решать простые уравнения по правилам нахождения неизвестных компонентов. Например: *чтобы найти неизвестное слагаемое нужно из суммы вычесть известное слагаемое.*

Выглядит это так:

*X+5=7*

*X=7-5*

*X=2*

Но для более сложных уравнений приходится хитрить и применять правила нахождения неизвестных несколько раз. Например уравнение:

*2(х+1) = 8*

*x+1 = 8:2*

*x+1 = 4*

*x = 4-1*

*x = 3*

Тут нам сначала приходится находить неизвестный множитель, а потом неизвестное слагаемое. В общем выглядит не сильно сложно, но если мы возьмём уравнение более сложного вида, например такое: *5 - 2(x - 1) = 4 - x,*  то тут нам пришлось бы применять правила нахождения неизвестных более пяти раз и выполнять кучу промежуточных вычислений. Поэтому алгоритм решения уравнений с использованием правил нахождения неизвестных мне показался слишком долгим для сложных уравнений.

Исследуя литературу по решению уравнений и поиск информации в интернете я остановилась на следующем алгоритме.

1) Раскрыть скобки (если они есть)

2) Перенести в левую часть компоненты с неизвестным, а в правую числовые компоненты по правилам переноса.

3) Упростить (вычислить) обе части.

4) Вычислить неизвестное, если есть необходимость

Решим наше предыдущее уравнение:

*5-2(х-1) = 4-х*

*1) 5-2х+2=4-х*

*2) -2х+х=4 - 5 - 2*

*3) -х=-3*

*4) х=3*

Мы видим что решение простое и компактное. Значит для нашего приложения — тренажера я воспользуюсь этим алгоритмом. Ниже приведена блок схема этого алгоритма.

Уравнение содержит скобки?

Да

Раскрыть скобки

В левой части только неизвеные а в правой только числа?

Нет

Нет

Да

Выполнить перенос

Требуется упрощение?

Упростить

Да

Нет

Неизвестное найдено?

Вычислить неизвестное

Да

Да

Нет

Рис.1. Блок схема алгоритма решения уравнений

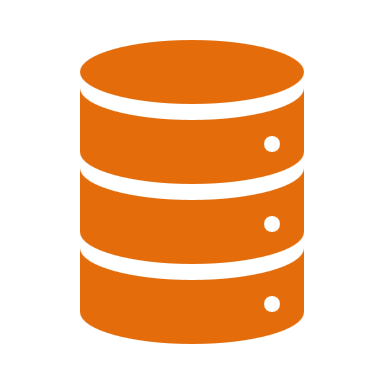
**3. Тренажер решения уравнений и его польза**

Продуктом моего проекта является тренажер решения уравнений. Это Web – приложение, которое помогает школьникам потренироваться решать линейные уравнения. По задумке, моё приложение не будет решать за школьника уравнение, а в место этого разделит процесс решения на этапы и выдаст подсказку по решению для каждого этапа.

Я выбрала разработку Web – приложения, так как его можно запустить практически на любом устройстве (телефон, компьютер, планшет) где есть интернет, что будет очень удобно для школьников. Web – приложение запускается в браузере. Браузер – это программа для отображения Web сайтов. Наше Web-приложение это, по сути, маленький сайт. Хранятся сайты на серверах, значит в процессе разработки нам необходимо найти место (сервер), где мы будем хранить наше Web-приложение. Для хранения приложения я остановилась на сервисе хранения исходных кодов <https://github.com/>. Данный сервис предоставляет нам возможность хранить исходный код нашего Web – приложения, и будет служить сервером для его загрузки.



Ресурсы Web приложения



Ресурсы Web приложения

Пользователи

Интернет

Ресурсы Web приложения

Сервер

Рис. 2. Схема работы Web-приложений

Наше­­ приложение может быть полезным для учеников пятых классов. В этом периоде мы только начинаем решать уравнения и нам может быть сложно разобраться в них. По себе я знаю, что, разделив любую задачу на части (этапы), в ней становится легче разобраться и решить её. Поэтому, в нашем приложении я использую поэтапное решение уравнений. Так же может быть полезным то, что приложение не решает за ученика уравнение, а проверяет только его ответ на каждый этап, заставляя его подумать над своими ошибками.

**4. Разработка тренажера решения уравнений**

**4.1 Выбор среды разработки**

Знакомясь с языками программирования, я пока что остановилась на изучении языка Python. Клиентскую часть Web приложения в большинстве случаев разрабатывают на языке JavaScript, так как именно на этом языке выполняет активное содержимое браузер. На данном этапе мне показался это язык более сложным чем Python. Поэтому я решила использовать для нашего Web-приложения Python с использованием промежуточной среды Brython. Библиотека Brython переводит программы написанные на Python в JavaScript прямо в браузере. Все что нам нужно – это подключить библиотеку Brython и загрузить на странницу наше приложение на Python.

Также для быстрой разработки интерфейса я использовала уже готовые модули для среды Brython. Вот они:

1. Модуль SimpleEquation – этот модуль содержит компонент - поле ввода простого уравнения. Этот компонент удобен тем, что он проверят правильность ввода уравнения за нас. Также используя его очень удобно организовать обход по элементам уравнения и выделения цветом частей уравнения.

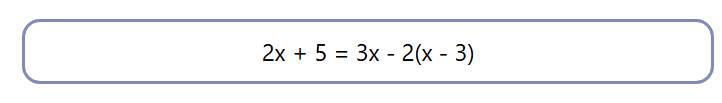


Рис.3. Компонент SimpleEquation на странице Web приложения

1. Модуль Controls – из этого модуля мы будем использовать компоненты Button и Label. Button – это кнопка, при нажатии на которую мы запустим нужные нам действия. Label – это текстовое поле, куда мы будем выводить информацию про тот или иной этап.



Рис.3. Компоненты Button на странице Web приложения

**4.2. Этапы разработки и разбитие задачи на части**

При разработке приложения я прошла через следующие этапы.

1. Сформулировать общую логику работы приложения
2. Разбить задачу на части
3. Разработка частей, алгоритмов их работы и написание их кода
4. Тестирование частей и приложения целиком
5. По результатам тестирования - сбор информации о доработках для следующих версий.

Общую логику задачи можно описать так. Должен быть интерфейс, где пользователь сможет ввести уравнение. Должна быть кнопка, после нажатия которой, выполняется проверка введенных данных на каждом этапе, и переход к следующему этапу. А также кнопка “Заново” для сброса приложение в начальное состояние. Пользователь вводит уравнение, и после нажатия на кнопку “Начать” (для первого этапа) или “Далее” (для остальных) приложение проверят необходим ли текущий этап. Если необходим - то выводит подсказку по этому этапу, ждет результат от пользователя, проверяет результат и, если верно переходим к следующему этапу. Если же этап не нужен, то сразу переходим к следующему этапу.

Задачу можно разбить на следующие части:

1. Для первого этапа (Раскрыть скобки)
   1. Определить, нужно ли раскрывать скобки в уравнении
   2. Выделить скобки в уравнении
   3. Раскрыть скобки в уравнении, чтобы проверить наш результат с результатом пользователя
2. Для второго этапа (Перенести все неизвестные компоненты влево, а известные вправо)
   1. Определить, нужно ли выполнять перенос в уравнении
   2. Выделить в уравнении красным то, что нужно переносить влево, а то, что вправо выделить синим
   3. Выполнить перенос в уравнении, чтобы проверить наш результат с результатом пользователя
3. Для третьего этапа (Упростить обе части уравнения)
   1. Определить, нужно ли упрощать уравнение
   2. Выделить знаки, с помощью которых мы будем упрощать уравнение
   3. Выполнить упрощение уравнения, чтобы проверить наш результат с результатом пользователя
4. Для четвертого этапа (Вычислить неизвестное)
   1. Определить, нужно ли вычислять неизвестное в уравнении
   2. Вычислить неизвестное, чтобы проверить наш результат с результатом пользователя

Код Web приложения разбит на три части - три файла.

* Main.py – это основной файл приложения. В нем мы объявляем компоненты нашего интерфейса, описываем действия при событиях (нажатие кнопок), и описываем основную логики этапов решения уравнения.
* Functions.py – это файл с функциями в которых выполняются алгоритмы для каждой части задачи (например определение нужен ли тот или иной этап, алгоритмы выделения частей уравнения для этапов, расчет результата для каждого этапа).
* Tips.py – файл с текстами для пояснений к каждому этапу.

В следующей главе рассмотрим упрощенные схемы алгоритмов для реализации каждой части

* 1. **Описание алгоритмов**

**Алгоритм определения нужно ли раскрывать скобки в уравнении.**

Тут нам нужно перебрать элементы уравнения и если мы наткнулись на элемент с типом «скобка» вернуть True, если не наткнулись, вернуть False.

ЕстьСкобки(Уравнение)

Взять Элемент из Уравнение

Нет

Элемент.Тип = Скобка

Да

False

True

Рис. 4. Схема алгоритма определения нужно ли раскрывать скобки в уравнении

**Алгоритм** **раскрытий скобок в уравнении**

Тут нам нужно перебрать элементы уравнения и если некий элемент стоит перед скобкой, то сохранить его дня дальнейшего перемножения с тем, что внутри скобок. Также нужно сохранить знак перед скобкой (плюс там или минус). После того как элемент равен открывающейся скобке, мы выставляем флаг что мы внутри скобки и последующие элементы нужно перемножать с тем, что перед скобкой. После закрывающейся скобки, мы снимаем этот флаг. Если элемент не перед скобкой и не внутри скобки, то мы просто переносим его в результат.

РаскрытьСкобки(Уравнение)

Да

Элемент.Значение == ‘(‘

Взять Элемент из Уравнение

Результат = “”

Нет

Да

Элемент.Значение == ‘)‘

Выставить флаг что мы внутри скобок

Нет

Снять флаг что мы внутри скобок

Мы внутри скобок?

Нет

Умножить ЭлементПередСкобкой на Элемент и поместить в Результат

Элемент стоит перед скобкой?

Нет

Да

Результат

ЭлементПередСкобкой = Элемент

Поместить Элемент в Результат

Рис. 5. Схема алгоритма раскрытия скобок в уравнении

**Алгоритм проверки нужен ли перенос элементов в уравнении**

Это алгоритм должен проверить есть ли неизвестные в правой части уравнения (чтобы перенести потом их влево), либо же есть числа без неизвестного в левой части уравнения(чтобы перенести потом их вправо),. Для этого переберем элементы левой и правой части и проверим их тип.

НуженПеренос(Уравнение)

True

Да

Нет

Элемент.Тип = Число

Элемент.Тип = Неизвестное

Взять Элемент из Уравнение.ЛеваяЧасть

Взять Элемент из Уравнение.ПраваяЧасть

Нет

Да

False

True

Рис. 6. Схема алгоритма проверки нужен ли перенос элементов в уравнении

**Алгоритм переноса**

Тут действуем в три этапа. Сначала записываем в результат для левой части неизвестные из левой части уравнения (оставляем неизвестные слева). Затем дополняем результат левой части неизвестными из правой части уравнения и меняем знак перед ними на противоположный, а в результат правой части записываем числа из правой части уравнения (оставляем числа справа). И в конце записываем в результат правой части числа из левой части уравнения и меняем знак перед ними на противоположный.

Сохранят результаты будем в переменные для левой и правой части. В конце соберем уравнение из этих переменных и знака «=» между левой и правой частью.

Да

Добавить Элемент в РезультатПравый

Нет

Элемент.Тип == Число

Элемент.Тип == Неизвестное

Взять Элемент из Уравнение.ПраваяЧасть

Добавить Элемент в РезультатПравый

Нет

Нет

Да

Добавить Элемент в РезультатЛевый с противоположным знаком

Да

Элемент.Тип == Число

Взять Элемент из Уравнение.ЛеваяЧасть

Добавить Элемент в РезультатЛевый

Да

Нет

Элемент.Тип = Неизвестное

РезультатПравый = «»

РезультатЛевый = «»

Перенос(Уравнение)

Взять Элемент из Уравнение.ЛеваяЧасть

РезультатЛевый + «=» + РезультатПравый

Рис. 7. Схема алгоритма проверки нужен ли перенос элементов в уравнении

**Алгоритм проверки нужно** **ли упрощение уравнения**

Упрощение уравнения нужно в случае, когда хотя бы в одной части уравнения есть два значения и операция между ними. То есть нам нужно проверить количество компонентов в левой и правой части и если компонентов больше трех (два значения и операция к примеру 2+2), то упрощение нужно, возвращаем True.

Да

True

Колво(Уравнение.ЛеваяЧасть) >=3

НуженоУпрощение(Уравнение)

False

True

Колво(Уравнение.ПраваяЧасть) >=3

Нет

Да

Нет

Рис. 8. Схема алгоритма проверки нужен ли упрощение уравнения

**Алгоритм упрощения уравнения**

Тут мы должны для каждой части уравнения выполнить суммирование всех значений компонентов со знаком перед ними.

Сохранят результаты будем в переменные для левой и правой части. В конце соберем упрощенное уравнение из этих переменных, буквы “неизвестного” (хранится в переменной Уравнение) и знака «=» между левой и правой частью

РезультатПравый = 0

РезультатЛевый = 0

Нет

Да

Взять Элемент из Уравнение.ЛеваяЧасть

Упрощение(Уравнение)

Элемент.Тип == Неизвестное

Добавить Элемент.ЗначениеПередНеизвестным в РезультатЛевый учитывая знак перед ним

Взять Элемент из Уравнение.ПраваяЧасть

Элемент.Тип == Число

Да

Нет

Добавить Элемент.Значение в РезультатПравый учитывая знак перед ним

КакСтрока(РезультатЛевый) + Уравнение.Неизвестное + «=» + КакСтрока(РезультатПравый)

Рис. 9. Схема алгоритма упрощения уравнения

**Алгоритм проверки нужно ли вычисление неизвестного**

Вычислять неизвестное нужно в том случае, если перед ним стоит минус или число. Например -x = -2, или 2x=4. Значит мы должны проверить первый компонент левой части на значение «-» или на значение перед неизвестным более единицы.

НуженоВычислениеНеизвестного(Уравнение)

Нет

Уравнение.ЛеваяЧасть[0]. Значение == «-»

Да

True

Нет

Уравнение.ПраваяЧасть[0]. ЗначениеПередНеизвестным > 1

Да

False

True

Рис. 10. Схема алгоритма проверки нужно ли вычисление неизвестного

**Алгоритм вычисления неизвестного**

Тут нам нужно будет разделить правую часть н левую часть. При этом если они не делятся нацело, нам нужно пометить в правую часть дробь. Для этого выполним три действия. Первым действием определим число правой части с учетом знака перед ним. Вторым действие определим число правой части с учетом знака перед ним. И третьим действием определим делятся ли части нацело, если да, то вернем корень уравнения, если нет, то в результате вернем корень уравнения с дробью.

Нет

Да

ЗначениеПравойЧасти % ЗначениеЛевойЧасти == 0

ЗначениеПравойЧасти= Уравнение.ЛеваяЧасть[0].Значение

ЗначениеПравойЧасти = -Уравнение.ЛеваяЧасть[1].Значение

Нет

Да

Уравнение.ПраваяЧасть[0]. Значение == «-»

ЗначениеЛевойЧасти= Уравнение.ЛеваяЧасть[0].ЗначениеПередНеизвестным

ЗначениеЛевойЧасти = -Уравнение.ЛеваяЧасть[1].ЗначениеПередНеизвестным

ВычислениеНеизвестного(Уравнение)

Нет

Да

Уравнение.ЛеваяЧасть[0]. Значение == «-»

Уравнение.Неизвестное + «=» + КакСтрока(ЗначениеПравойЧасти) + «/» КакСтрока (ЗначениеЛевойЧасти)

Уравнение.Неизвестное + «=» + КакСтрока(ЗначениеПравойЧасти / ЗначениеЛевойЧасти)

Рис. 11. Схема алгоритма вычисления неизвестного

* 1. **Тестирование проекта**

Чтобы протестировать наше приложение необходимо проверить как очень простые, так и сложные варианты уравнений.

Самый простой вариант уравнений *x = 1*. В данном случае все этапы должны пропускаться, и нажимая кнопку «далее» мы придём к ответу.

Следующие уравнения, которые мы должны проверить, это такие при которых нужно будет выполнить только последний этап. Например: *2x = 4*. Так же нам следует проверить уравнения с минусом *-x = -1*. Ещё проверим уравнение, в котором результатом будет дробь *2x = 1.*

Теперь проверим уравнения, где необходимо выполнить два последних этапа: упрощение уравнения и вычисление неизвестного. Сначала проверим уравнения, в которых упрощать нужно только левую часть: *2x + x = 6*. Потом те, где нужно упрощать правую часть: *x = 4 – 3.* И наконец те, где нужно упрощать обе части: *x + 4x = 6 – 1.*

Далее проверим уравнения, где нужно выполнить три последних этапа: перенос, упрощение уравнения и вычисление неизвестного. Здесь мы тоже разобьём уравнения на три вида которые нам необходимо протестировать. Первый, где нужен перенос только слева вправо: 2*x – 2 = 2*. Второй, где нужен перенос только справа влево: *3x = x + 4.* И третий, где нужен перенос и слева вправо, и справа влево: *3 – x = x + 1.*

И наконец, уравнения, где нужно проверить все этапы: раскрытие скобок, перенос, упрощение уравнения и вычисление неизвестного. И тут мы тоже проверим три варианта. Первый, где раскрытие скобок нужно только в левой части и в этом случае разделим проверку ещё на четыре варианта:

* + - 1. Когда перед скобками ничего нет: (*x + x) – 2 = 2*.
      2. Когда перед скобками минус: *-(x – 3x) – 2 = 2.*
      3. Когда перед скобками число: *2(x – 1) = 2.*
      4. Когда перед скобками неизвестное: *x(1 + 1) – 2 = 2.*

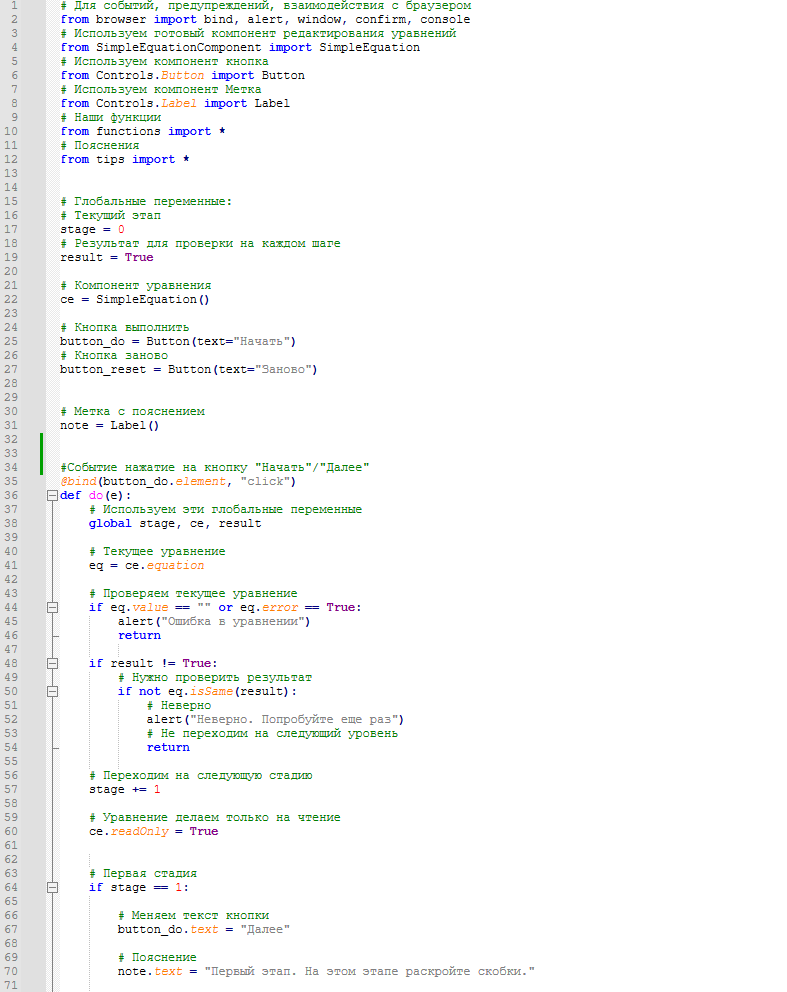
Второй, когда раскрытие скобок нужно только в правой части: *4x = 2(x + 2).* И третий, когда раскрытие скобок нужно в обоих частях: *x(3 + 1) = 2(x + 2).* Также проверим вариант с несколькими скобками в оной из частей: *4x = 2(x + 2) – (x + 1).*

Все представленные варианты уравнений были проверены в нашем Web приложении, и все они дали ожидаемый результат.

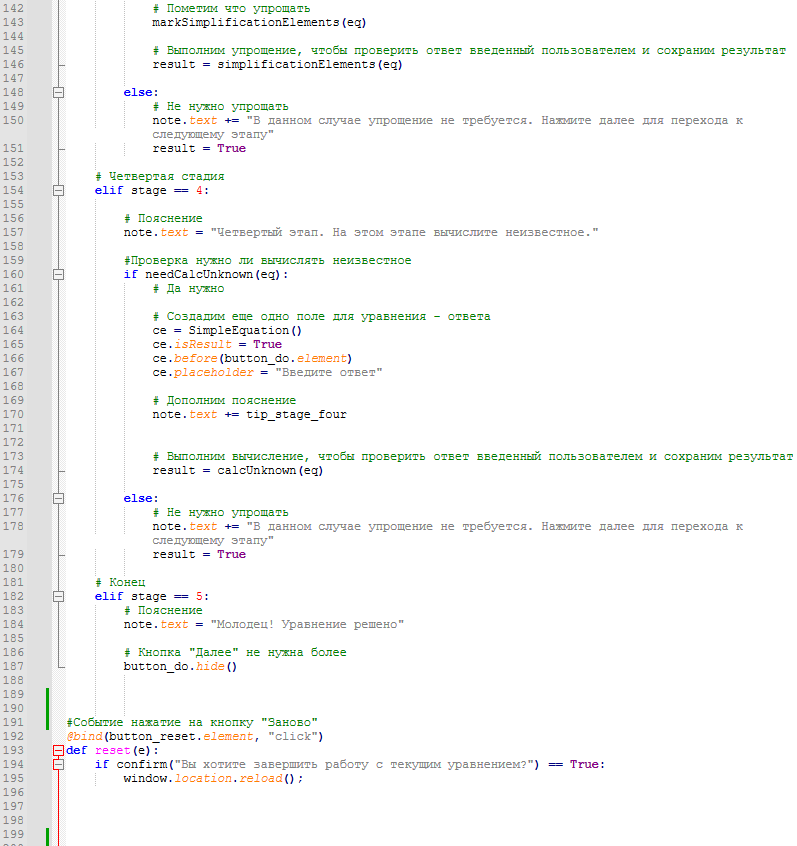
**Заключение**

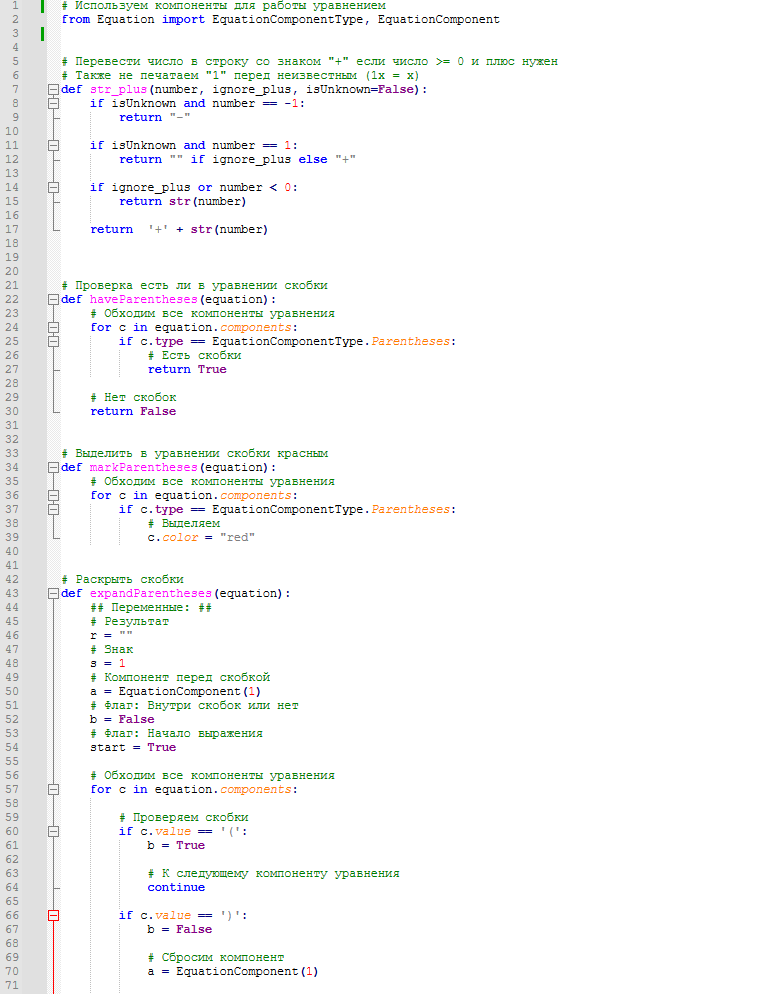
**Список источников информации**

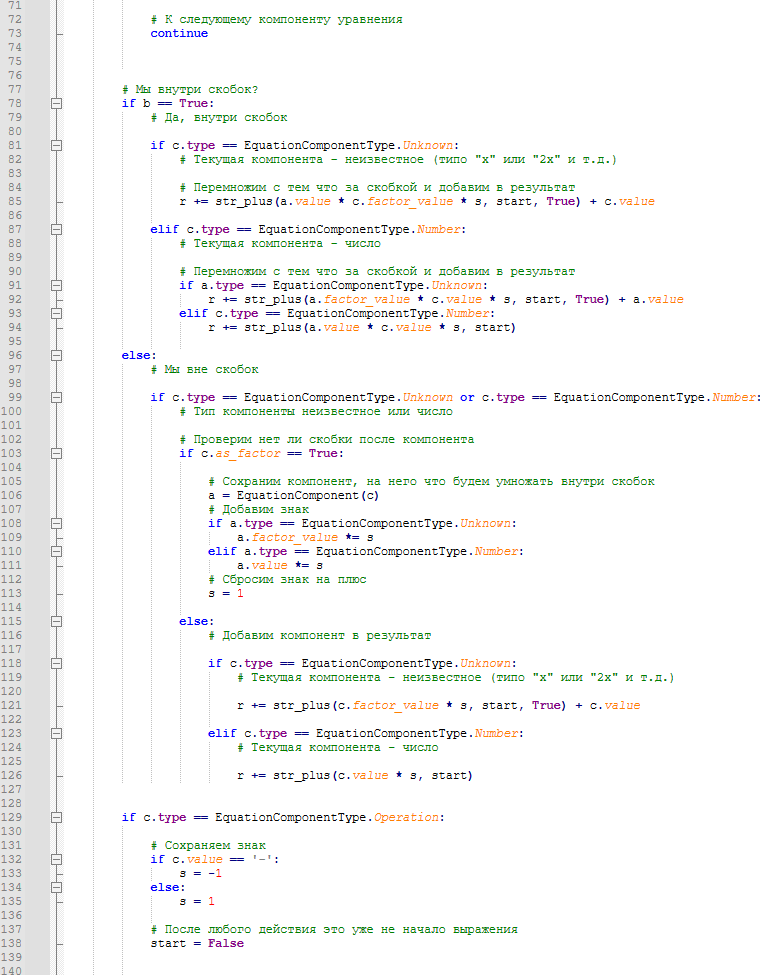
* + - 1. Джейсон Бриггс, Python для детей
      2. Brython: <https://brython.info>
      3. Как решать линейные уравнения: <https://math-prosto.ru/ru/pages/linear_equations/solution_linear_equations>
      4. Как решать линейные уравнения — формулы и примеры решения простейших уравнений: <https://skysmart.ru/articles/mathematic/reshenie-prostyh-linejnyh-uravnenij>

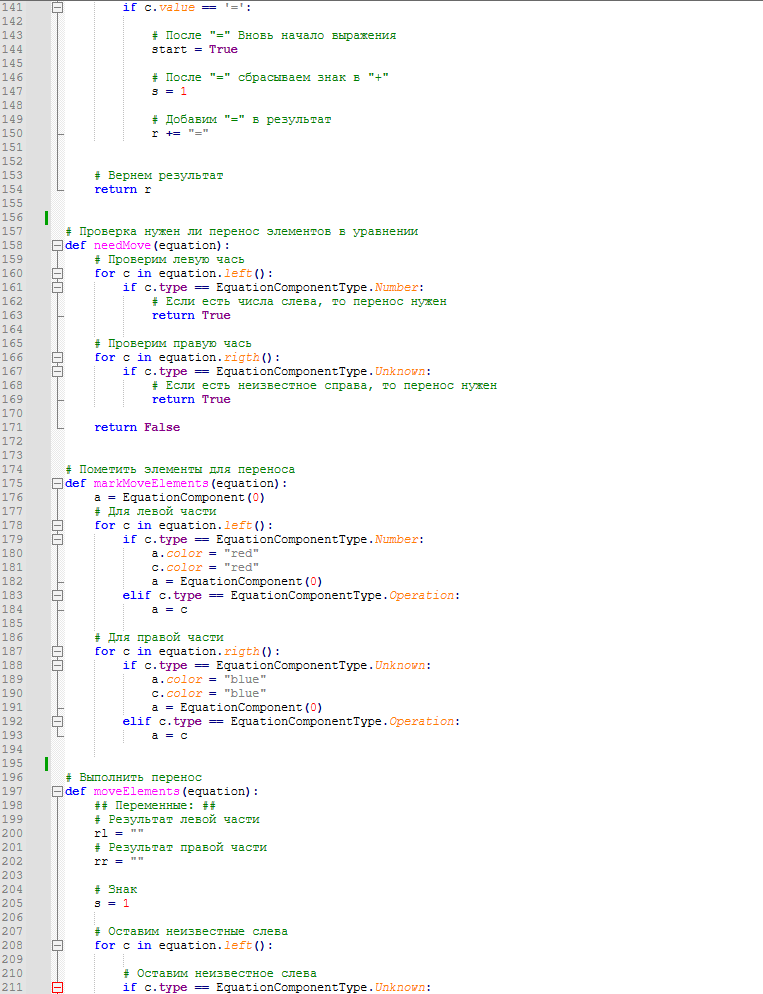
**Приложение 1. Листинг модуля main.py**

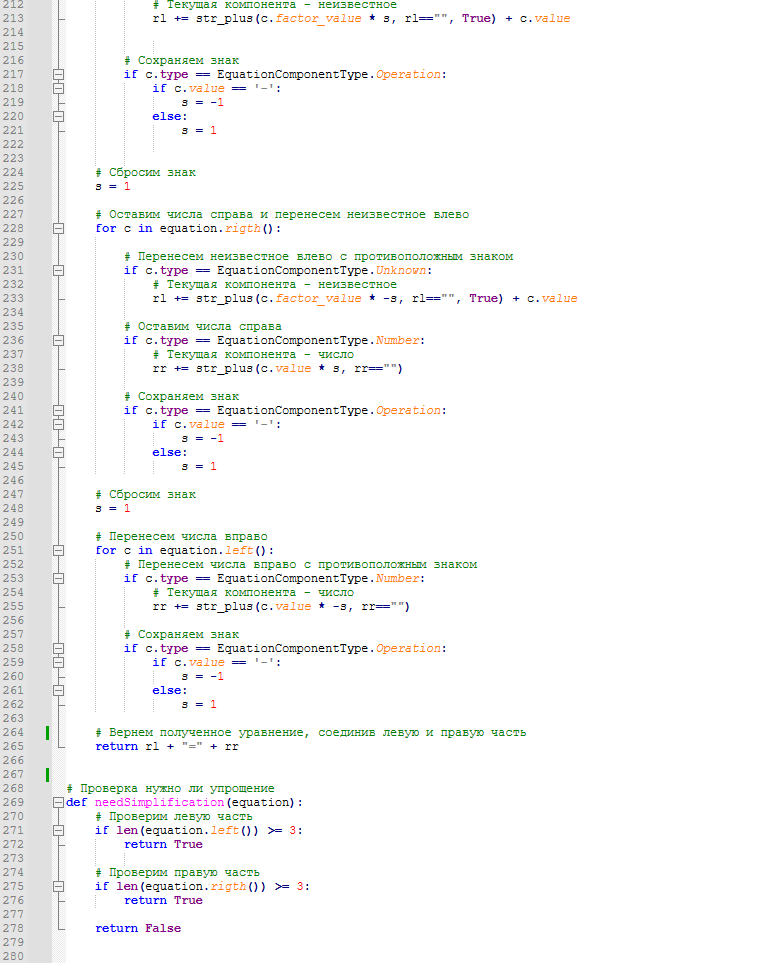
****

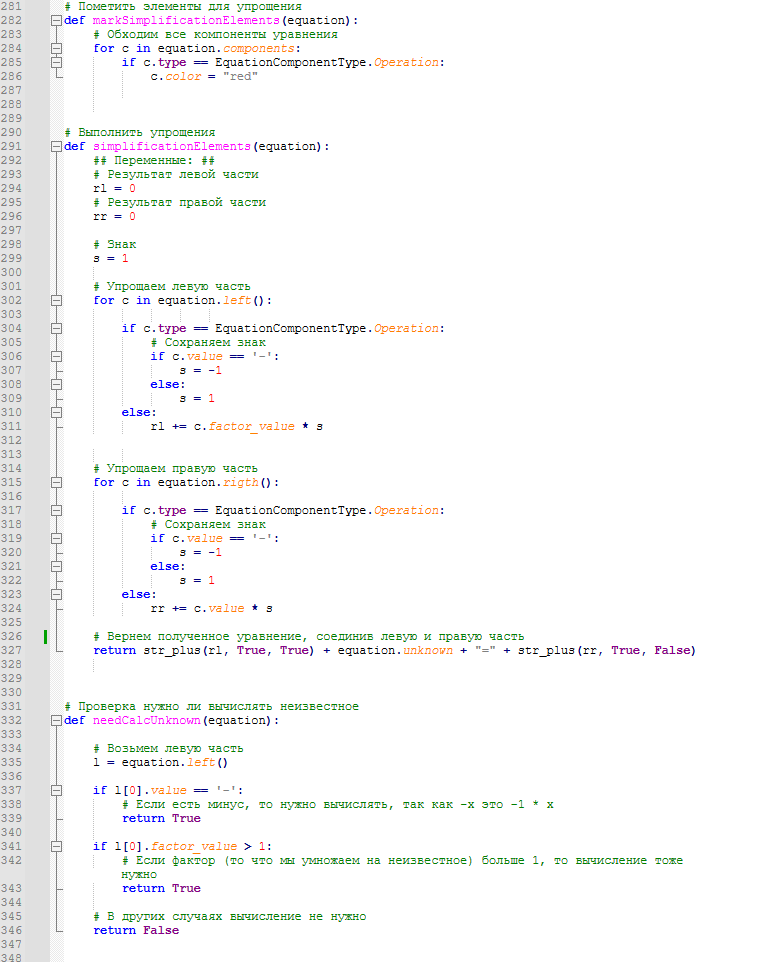
****

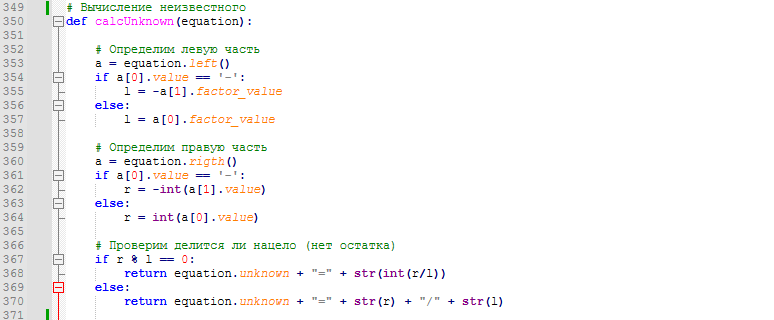
**Приложение 2. Листинг модуля function.py**

****

****

****

****

****