Google Summer of Code 2020

Revamping Math Interactions

Prayush Dawda

About You

Why are you interested in working with Oppia, and on your chosen project?

I have always relied on online resources to fill the gaps in my understanding of concepts. It's a great feeling to find a website that offers a variety of topics that are taught with clarity and brevity. So, when I found Oppia, I was genuinely impressed with its unique style of teaching by storytelling. I was also excited to find out that this project is open-sourced. This meant that I could use my skills to give back to the community of online education, especially since everyone in this organization is so inviting to newcomers.

Oppia provides an intuitive experience for learners. I tried out playing through a couple of explorations and I felt like I was talking to a friend. Apart from that, what I find unique is that Oppia provides a versatile platform for the creators. Here they can illustrate their ideas in the form of interactive lessons. Thus, having features with sufficient functionality is imperative in order to ensure the high quality of these interactive lessons.
Unfortunately, the current state of one of the most important features, the Math Expression Input Interaction, is not great. A large number of interactive lessons depend on this math interaction, which is why it is important for it to be bug-free. Some of the bugs found are listed here.

This project aims to add three new math-based interactions that will replace the current math interaction. This would end up solving all the issues that the current interaction has, and provide a more versatile rule-set which can detect a variety of learner answers.
I'm really excited to work on this project as its success will ensure that the math-experience of learners and creators is smooth and fruitful.

Prior experience

I am sufficiently familiar with the technologies that Oppia uses. I have been using Python for more than two years now. I have also been using JavaScript for about a year now. In particular, I have been using the framework Oppia is built on, Angular, for more than five months now. Some of my open-source projects that showcase my technical skills are here.
I have been participating in competitive coding competitions for more than two years now, which is why I will be able to write efficient algorithms that this project requires. My profiles: CodeChef. CodeForces.

Apart from this, I have been actively contributing to Oppia for more than three months now, by creating PRs, filing issues and reviewing other's code. I am a member of the Learner and Creator Experience team and I have gained sufficient knowledge regarding the parts of the codebase that are relevant to this project. I have worked with the frontend as well as the backend and a lot of my work has been related to enhancing or fixing bugs related to interactions. I have also been part of the release testing team which has shown me the user's point of view and I can make design decisions keeping the user as a focal point.

Some of the major PRs I've made till now are:

- Added an audit job to find explorations with MathExpressionInput. (#8669)
- Customizing "Add Item" button text. (#8638)
- Fixed fraction input errors. (#8454)
- Changed setup.py to rename yarn folder. (#8405)
- Added a callout to classroom page. (#8511)

I will continue my contributions to Oppia. Here's the full list of PRs and the list of issues I've filed.

Contact info and timezone

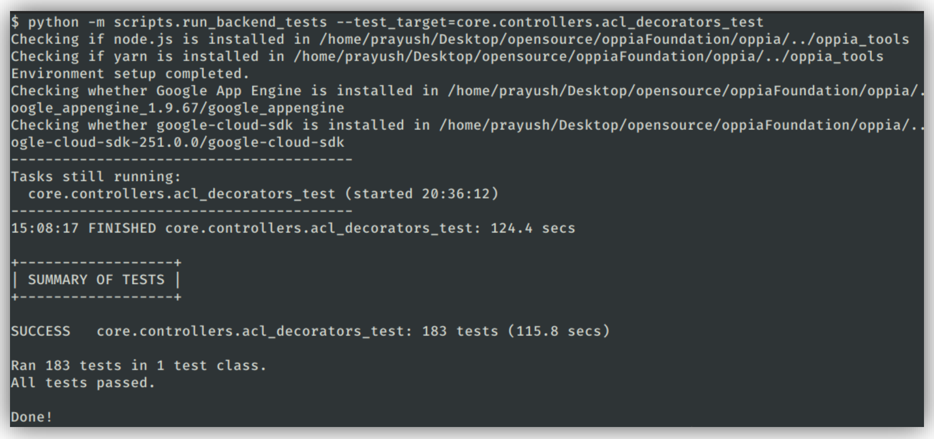
Email: iamprayush@gmail.com
GitHub profile: iamprayush
I will stay in India throughout the summer. The time zone will be Indian Standard Time (GMT+5:30)

Time commitment

I plan on dedicating 7-8 hours a day to this project.
Due to COVID-19, I have no idea about the dates of my final exams. They will most likely last one week and my work hours would reduce to 5-6 hours a day during that week. In order to cover-up, I intend on increasing my work rate before that week to 8-9 hours a day.

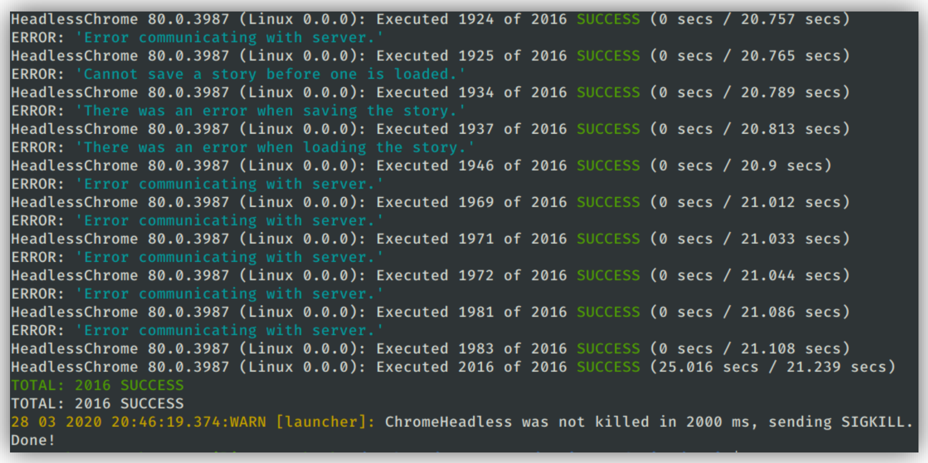
Essential Prerequisites

- I am able to run a single backend test target on my machine.



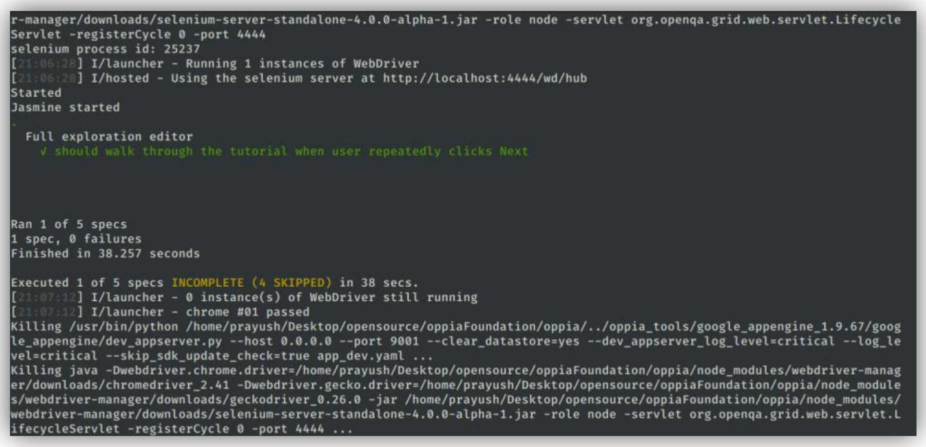
```
$ python -m scripts.run_backend_tests --test_target=core.controllers.acl_decorators_test
Checking if node.js is installed in /home/prayush/Desktop/opensource/oppiaFoundation/oppia/../oppia_tools
Checking if yarn is installed in /home/prayush/Desktop/opensource/oppiaFoundation/oppia/../oppia_tools
Environment setup completed.
Checking whether Google App Engine is installed in /home/prayush/Desktop/opensource/oppiaFoundation/oppia/.
oogle_appengine_1.9.67/google_appengine
Checking whether google-cloud-sdk is installed in /home/prayush/Desktop/opensource/oppiaFoundation/oppia/..
ogle-cloud-sdk-251.0.0/google-cloud-sdk
---------------------------------------
Tasks still running:
  core.controllers.acl_decorators_test (started 20:36:12)
---------------------------------------
15:08:17 FINISHED core.controllers.acl_decorators_test: 124.4 secs

+------------------+
| SUMMARY OF TESTS |
+------------------+

SUCCESS   core.controllers.acl_decorators_test: 183 tests (115.8 secs)

Ran 183 tests in 1 test class.
All tests passed.

Done!
```

- I am able to run all the frontend tests at once on my machine.



```
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1924 of 2016 SUCCESS (0 secs / 20.757 secs)
ERROR: 'Error communicating with server.'
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1925 of 2016 SUCCESS (0 secs / 20.765 secs)
ERROR: 'Cannot save a story before one is loaded.'
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1934 of 2016 SUCCESS (0 secs / 20.789 secs)
ERROR: 'There was an error when saving the story.'
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1937 of 2016 SUCCESS (0 secs / 20.813 secs)
ERROR: 'There was an error when loading the story.'
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1946 of 2016 SUCCESS (0 secs / 20.9 secs)
ERROR: 'Error communicating with server.'
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1969 of 2016 SUCCESS (0 secs / 21.012 secs)
ERROR: 'Error communicating with server.'
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1971 of 2016 SUCCESS (0 secs / 21.033 secs)
ERROR: 'Error communicating with server.'
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1972 of 2016 SUCCESS (0 secs / 21.044 secs)
ERROR: 'Error communicating with server.'
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1981 of 2016 SUCCESS (0 secs / 21.086 secs)
ERROR: 'Error communicating with server.'
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 1983 of 2016 SUCCESS (0 secs / 21.108 secs)
HeadlessChrome 80.0.3987 (Linux 0.0.0): Executed 2016 of 2016 SUCCESS (25.016 secs / 21.239 secs)
TOTAL: 2016 SUCCESS
TOTAL: 2016 SUCCESS
28 03 2020 20:46:19.374:WARN [launcher]: ChromeHeadless was not killed in 2000 ms, sending SIGKILL.
Done!
```

- I am able to run one suite of e2e tests on my machine.



```
r-manager/downloads/selenium-server-standalone-4.0.0-alpha-1.jar -role node -servlet org.openqa.grid.web.servlet.Lifecycle
Servlet -registerCycle 0 -port 4444
selenium process id: 25237
[21:06:28] I/launcher - Running 1 instances of WebDriver
[21:06:28] I/hosted - Using the selenium server at http://localhost:4444/wd/hub
Started
Jasmine started
.
  Full exploration editor
    √ should walk through the tutorial when user repeatedly clicks Next




Ran 1 of 5 specs
1 spec, 0 failures
Finished in 38.257 seconds

Executed 1 of 5 specs INCOMPLETE (4 SKIPPED) in 38 secs.
[21:07:12] I/launcher - 0 instance(s) of WebDriver still running
[21:07:12] I/launcher - chrome #01 passed
Killing /usr/bin/python /home/prayush/Desktop/opensource/oppiaFoundation/oppia/../oppia_tools/google_appengine_1.9.67/goog
le_appengine/dev_appserver.py --host 0.0.0.0 --port 9001 --clear_datastore=yes --dev_appserver_log_level=critical --log_le
vel=critical --skip_sdk_update_check=true app_dev.yaml ...
Killing java -Dwebdriver.chrome.driver=/home/prayush/Desktop/opensource/oppiaFoundation/oppia/node_modules/webdriver-manag
er/downloads/chromedriver_2.41 -Dwebdriver.gecko.driver=/home/prayush/Desktop/opensource/oppiaFoundation/oppia/node_module
s/webdriver-manager/downloads/geckodriver_0.26.0 -jar /home/prayush/Desktop/opensource/oppiaFoundation/oppia/node_modules/
webdriver-manager/downloads/selenium-server-standalone-4.0.0-alpha-1.jar -role node -servlet org.openqa.grid.web.servlet.L
ifecycleServlet -registerCycle 0 -port 4444 ...
```

Other summer obligations

I have no other jobs this summer and I plan to spend my time working on this project solely.

Communication channels

I will mostly be active via email, Hangouts, and Gitter. Any other method of communication is also fine for me.
I plan to communicate with my mentor every other day to update them on the project's progress as well as to ask them doubts. I also plan to have weekly meetings to talk about the tasks I will be doing the following week.

Project Details

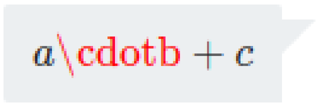
Product Design

This project aims to introduce three new interactions, **Algebraic Expression Input, Numeric Expression Input and Math Equation Input**. These interactions are meant to improve the lesson creator's capabilities, by providing versatility and variety in the rules for creating mathematical explorations. This would enable the creators to provide insightful feedback based on the learner's answer.
These interactions would also provide an intuitive experience for the learners as it aims to increase the range of answers that the learners can give and get relevant feedback on.

Currently, there exists a Math Expression Input Interaction that only supports algebraic expressions and not numeric expressions or math equations. It provides only one rule, "is mathematically equivalent to (LaTeX)", and the interaction is quite buggy. Some of the major bugs found are:

- If the exploration uses an equation, only the LHS of the learner's answer gets compared with the LHS of the creator's answer. This makes it practically impossible to create explorations that use equations.
- Simplification errors while comparing. For eg, if the given expression, to compare the learner's answer with, is specified as $a + b$.
    - Input that gets validated as correct even though it shouldn't:
        - $a + b * 2$
    - Inputs that *don't* get validated as correct even though they should:
        - $a * 1 + b$

        - $1 * a + b$
    - Another case: If the correct expression is x^2, entering $x * x$ as an answer gets validated as incorrect.
- The submit button gets disabled when an expression with symbols like $\sqrt{}$ is entered.
- The submit button gets disabled when x^{3*4} is entered.
- Rendering of LaTeX expressions is buggy. Submitting $a * b + c$ gets displayed as:



a\cdotb + c

Due to these bugs and the lack of variety in rules supported, a lot of explorations that probably wanted to use a math-based interaction, have had to resort to using the text input interaction, instead. For instance, the negative numbers collection uses the text input interaction. Using text input imposes annoying restrictions on the learner since something as minor as an extra space also leads to a wrong answer verdict. Not to mention, it would be very unintuitive to type in math expressions/equations in a normal text input box. This is especially true for young learners who comprise a significant portion of the Oppia user base.
This is why an upgrade is imperative and urgent.

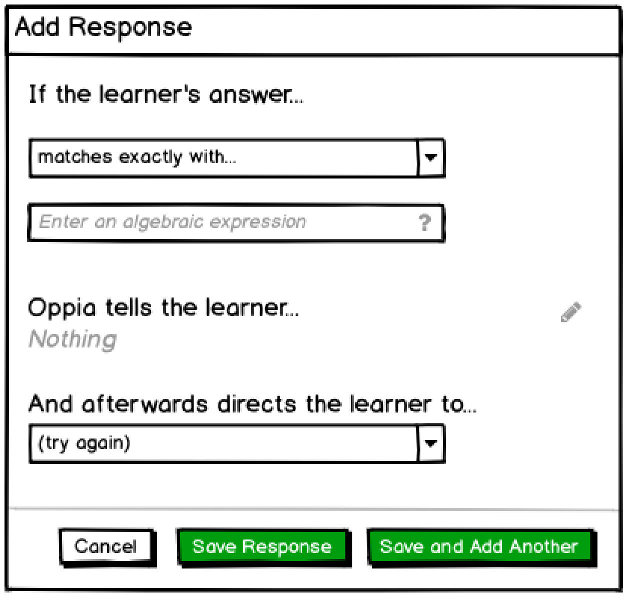
Considering the amount and severity of the bugs in the current interaction, I've proposed a ground-up rebuild with additional rules and customizations.
I'll try to illustrate my vision of the new interactions via User(learner and creator) Journeys.

Algebraic Expression Input Interaction

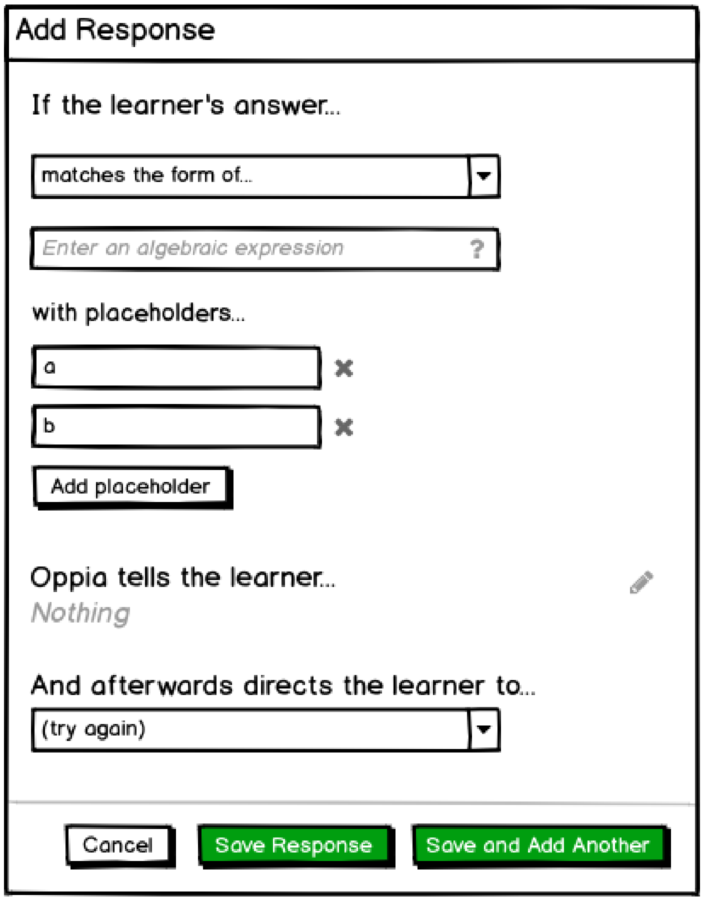
Creator's Journey

Upon choosing the *Algebraic Expression Input* interaction from the Math tab in the *Choose Interaction* modal, the creator will be directly taken to the *Add Response* modal since this interaction needs no customization args. Here the creator can provide insightful feedback by anticipating various kinds of learner answers. The modal would look something like this:

Rules: MatchesExactlyWith, ContainsSomeOf, OmitsSomeOf, IsEquivalentTo

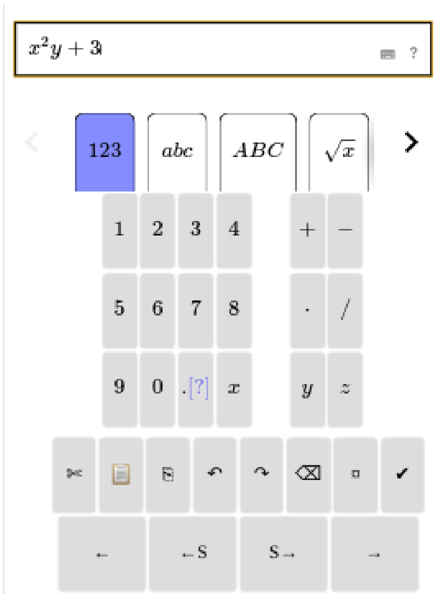

Add Response
If the learner's answer...
matches exactly with...
Enter an algebraic expression
?
Oppia tells the learner...
Nothing
And afterwards directs the learner to...
(try again)
Cancel
Save Response
Save and Add Another

Rules: MatchesWithGeneralForm

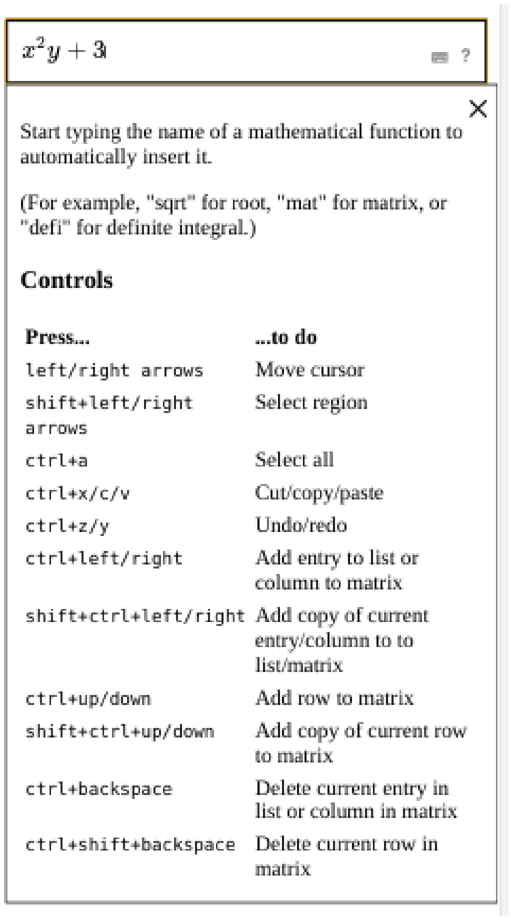


Mobile View Mocks (These would look same for desktop view)

On-screen keyboard



Help modal


x^2y+3

Start typing the name of a mathematical function to automatically insert it.

(For example, "sqrt" for root, "mat" for matrix, or "defi" for definite integral.)

Controls

Press...	**...to do**
`left/right arrows`	Move cursor
`shift+left/right arrows`	Select region
`ctrl+a`	Select all
`ctrl+x/c/v`	Cut/copy/paste
`ctrl+z/y`	Undo/redo
`ctrl+left/right`	Add entry to list or column to matrix
`shift+ctrl+left/right`	Add copy of current entry/column to to list/matrix
`ctrl+up/down`	Add row to matrix
`shift+ctrl+up/down`	Add copy of current row to matrix
`ctrl+backspace`	Delete current entry in list or column in matrix
`ctrl+shift+backspace`	Delete current row in matrix

The creator would then choose the desired rule out of the given options and then set the response to be shown to the learner if the answer gets validated by the specified rule. Finally, the creator would mention the path where the card redirects.

This interaction would not allow purely numeric expressions. There must be at least one variable in the expression. This interaction would support the following rules which can detect a wide range of learner answers:

- **MatchesExactlyWith:** “matches exactly with {{x|AlgebraicExpression}}”
  This rule would expect one argument, AlgebraicExpression, that the learner’s answer, which is also an AlgebraicExpression, can be compared with.
  The learner’s answer would be checked for mathematical equivalence with the creator’s input, *without* expanding out the parentheses present in either of the expressions.

Note that any redundant parentheses would be ignored. That means, any parentheses that don't change the syntax of the expression upon removal, would be ignored. For instance, $(x+y)+z$ would be exactly the same as $x+y+z$.
Also, any parentheses structure other than this would *not* be expanded before comparison takes place. For instance, $(x+y)(x-y)$ would *not* be exactly the same as x^2-y^2. Lastly, any constant values would be simplified if possible. For instance, $30/40+x+1/4$ would be considered the same as $x+1$.

- **IsEquivalentTo:** "is equivalent to {{x|AlgebraicExpression}}"
  This rule would expect one argument, AlgebraicExpression, that the learner's answer, which is also an AlgebraicExpression, can be compared with.
  Before comparison, both expressions would be expanded out completely. Thereafter, the learner's answer would be checked for mathematical equivalence with the creator's input. Note that there is practically no limit on how deep the expansion of parentheses would be done. For instance, $3((x-y)(x-y)\ -\ y)$ would be expanded out to $-3y+3y^2-6xy+3x^2$. All other simplifications mentioned in the MatchesExactlyWith rule would apply here as well.

- **ContainsSomeOf**: "contains at least one of the terms present in {{x|AlgebraicExpression}}"
  This rule would expect one argument, AlgebraicExpression.
  Two sets of terms would be formed by splitting the learner's answer and the creator's input. The splitting would be done according to the definition of algebraic terms.
  After splitting, we would get two lists of terms. For each term in the learner's list, it would be compared with every term in the creator's list and if even one of them matches, this rule would return true. This "match" between terms would be an exact match as mentioned above.

- **OmitsSomeOf**: "omits at least one of the terms present in {{x|AlgebraicExpression}}"
  This rule would expect one argument, AlgebraicExpression.
  Similar to the previous rule, here also, the learner's answer and the creator's input would be split into separate terms based on the definition of algebraic terms.
  For each term in the creator's list, it would be compared with every term in the learner's list and if at least one of them *doesn't* match, this rule would return true. This would imply that there is at least one term in the creator's list that is absent in the learner's list, which means that the learner has omitted that term.
  Here also, the "match" between terms would be an exact match (without expansion).

- **MatchesWithGeneralForm**: “matches the form of {{x|AlgebraicExpression}} with placeholders {{y|SetOfPlaceholders}}”
  This rule would expect two arguments, AlgebraicExpression and SetOfPlaceholders.
  In order to get validated by this rule, the learner’s answer must contain constant value replacements for the placeholders provided by the creator. For instance, if the general form is $ax + b$, this rule would return true if the learner answer is something like $2x + 3$.
  This will be validated by following these steps:
    - Firstly, it will be checked whether the learner’s answer contains any placeholders. If it does, this rule would return false, else we continue.
    - The expressions would be split by $+, \ -, \ *, \ /$. Now we’ll have two lists, answerTerms and generalTerms.
    - For each answer term, it will be tested for a “term match” with all of the general terms. Upon finding the first match, both the terms will be removed from their respective lists and the process will continue until both lists are empty. If there exists even a single term that doesn’t get matched with any of the other terms, this rule would return false.

  Term match:
  Two terms will match iff upon dividing or subtracting them, the resultant only contains constants and/or placeholders. For instance, ax will match with $2x$ since upon diving them, $a/2$ will be left which contains only placeholders and constants.

  Note:
  According to the proposed algorithm, if the general form is $abx/c \ or \ ax/b \ or \ ax * b/c$, they would all be considered the same since this is a rudimentary algorithm. It would collate multiple coefficients into a single one. So, x would be accepted as an answer to all of these. Also, $3x \ or \ x/3 \ or \ 5x/2$ would be accepted but $3x^2 \ or \ 3 \ or \ 3y \ or \ x + 2$ would not be accepted. In order to make things more apparent for the creator, a warning would be displayed if the creator enters a general expression/equation that contains multiple coefficients in a single term. This would be checked by splitting the given expression and checking for the number of variables that are also present in the placeholder set in that term. If it’s more than 1 a warning would be displayed. Also, a warning would be displayed if the creator enters an expression with placeholders in powers.

The following table aims to provide an overview of the kind of validations each rule would support with various examples. These examples would be extremely helpful while writing the unit tests for the rules.

RULE	VALUES OF THE PARAMETERS	ANSWERS	
		ACCEPTED	REJECTED
MatchesExactlyWith matches exactly with {{**x**|AlgebraicExpression}}	**x:** $(x^2-x)/3-4y$	$(x^2-x)/3-4y$ $-4y+(x^2-x)/3$ $(x^2-x)*3^{-1}-4y$	$x(x-1)/3-4y$ $x^2/3-x/3-4y$ $x^2/3-(x/3+4y)$
	x: $a^2+b^2+c^2+2ab$ $+2ac+2bc$	$a^2+b^2+c^2+2ab$ $+2ac+2bc$ $a*a+b*b+c^3/c+$ $2ab+2ac+2bc$	$a^2+b^2+c^2+$ $2(ab+ac+bc)$ $(a+b)^2+c^2+2ac+2bc$
	x: $x^{(2t+3)}+x^4$	$x^{(2t+3)}+x^4$ $x^{2t}*x^3+(x^2)^2$ $x^{2t}/x^{-3}+x^8/x^4$ $x^{2t+5}/x^2-(-x^4)$	$x^{(3t+3)}+x^4$ $x^3(x^{2t}+x)$ $x^4(x^{2t-1}+1)$ $(x^{2t+4}+x^5)/x$
	x: $9x^2-6x+1$	$9x^2-6x+1$ $3(3x^2-2x)+1$ $(3x)^2-6x+1$	$(3x-1)^2$ $3x(3x-2)+1$
IsEquivalentTo Is equivalent to {{**x**|AlgebraicExpression}}	**x:** $(x^2-x)/3-4y$	$(x^2-x)/3-4y$ $x(x-1)/3-4y$ $x^2/3-x/3-4y$ $x^2/3-(x/3+4y)$	$x(x^2-x)/3-4y$ $(x^2-x)/3+4y$ $(x^2+x)/3-4y$ $(x^2-x)*0.33-4y$
	x: $a^2+b^2+c^2+2ab$ $+2ac+2bc$	$a^2+b^2+c^2+$ $2(ab+ac+bc)$ $(a+b)^2+c^2+2ac+2bc$ $(a+b+c)^2,\ (-a-b-c)^2$	$(a+b+c)^3$ $a^2+b^2+c^2$ $a^2+b^2+c^2-2ab$ $-2ac-2bc$
	x: $x^{(2t+3)}+x^4$	$x^{(2t+3)}+x^4$ $x^3(x^{2t}+x)$ $x^4(x^{2t-1}+1)$ $(x^{2t+4}+x^5)/x$	$x^{(2t+3)}+x^3+x$ $x^{2t}+x^7$ $x^{(2t+3)}-x^4$

	x: $9x^2 - 6x + 1$	$(3x - 1)^2$ $\sqrt{(3x - 1)^4}$ $3x(3x - 2) + 1$ $3(3x^2 - 2x) + 1$	$(1 - 3x)^2$ $9x^2 - 6x - 1$ $(3x - 1)^4$
ContainsSomeOf contains at least one of the terms present in {{**x**|AlgebraicExpression}}	**x:** $(x^2 - x)/3 - 4y$	$(x^2 - x)/3 - 4y$ $(x^2 - x)/3$ $-4y$ $(x^2 - x) * 3^{-1} - 8y/2$	$(x - x^2)/3 + 4y$ $x(x - 1)/3$ $x^2/3 - x/3$
	x: $a^2 + b^2 + c^2 + 2ab$ $+ 2ac + 2bc$	$a^2 + b^2 + c^2$ $2ab + 2ac + 2bc$ $(a + b)^2 + c^2 + 2ac + 2bc$	$(a + b + c)^2$ $(a + b)^2$ $a + b + c$
	x: $x^{(2t+3)} + x^4$	$x^{(2t+3)}$ $(x^2)^2$ x^{2t}/x^{-3}	$x^3(x^{2t} + x)$ $(x^{2t+4} + x^5)/x$
	x: $9x^2 - 6x + 1$	$3x(3x - 2) + 1$ $(3x)^2 - 6x - 1$	$(3x - 1)^2$ $(3x - 1)^4$
OmitsSomeOf omits at least one of the terms present in {{**x**|AlgebraicExpression}}	**x:** $(x^2 - x)/3 - 4y$	$x(x - 1)/3 - 4y$ $x^2/3 - x/3 - 4y$	$(x^2 - x)/3 - 4y$ $(x^2 - x) * 3^{-1} - 4y$
	x: $a^2 + b^2 + c^2 + 2ab$ $+ 2ac + 2bc$	$(a + b + c)^2$ $a^2 + b^2 + c^2$ $a^2 + b^2 + c^2 +$ $2(ab + ac + bc)$	$a^2 + b^2 + c^2 + 2ab$ $+ 2ac + 2bc$ $a * a + b * b + c^3/c +$ $2ab + 2ac + 2bc$
	x: $x^{(2t+3)} + x^4$	$(x^{2t+4} + x^5)/x$ $x^3(x^{2t} + x)$	$x^{(2t+3)} + x^4$ $x^{2t}/x^{-3} + x^8/x^4$
	x: $9x^2 - 6x + 1$	$(3x - 1)^2$ $9x^2 - 6x$	$9x^2 - 6x + 1$ $(3x)^2 - 6x + 1$
MatchesWithGeneralForm matches the form of {{**x**|AlgebraicExpression}} with placeholders {{**y**|SetOfPlaceholders}}	**x:** $x/a + y/b$ **y:** $[a,\ b]$	$x/2 + y/8$ $y - 8x$ $y/8 + x/2$ $x + y$ $x/1.33 + y/4$ $x/3 + y/(8/22)$	$x/(1 + x) + y/8$ $x/2 + x/8$ $x/2$ $x/2 + y^2/8$ $x/2 + y - 5$ $x/2y + y/8$

		$x/(1 \cdot 3) + y/8$ $x/2 - y/8$	$x/2 + y/b$

Algebraic Term: Each part of an expression separated by a $+ \ or \ -$ sign is considered to be an algebraic term. If present, a parenthesized portion of the expression is considered to be a single term unless in a case where removing/expanding out the parentheses has no effect on the syntax of the expression (this would imply that the parentheses are redundant).
These would be considered as single algebraic terms: $2x,\ x^2,\ 5x/4,\ 3(4\ +\ 2a),\ (a\ +\ b)/(a\ -\ b)$.
These would not be considered as single algebraic terms: $x\ +\ y,\ 3x\ -\ 2,\ (2x - 3)$.
Examples of splitting an expression:

- $x\ +\ y\ +\ z \Rightarrow [x,\ y,\ z]$
- $(3x\ +\ 2y)\ -\ z \Rightarrow [3x,\ \ 2y,\ \ -z]$
- $x * y\ -\ z \Rightarrow [x * y,\ \ -z]$
- $x/(y\ -\ z)\ +\ 4 \Rightarrow [x/(y\ -\ z),\ 4]$
- $4 * (x\ +\ 3y) \Rightarrow [4 * (x\ +\ 3y)]$

Example Walkthrough 1
Question: Complete the square in $x^2 + 4x + 3$

- If LA "matches exactly with" $(x + 2)^2 - 1$
  - Feedback: "That is Correct!"
- Else if LA "contains at least one of the terms present in" $(x + 2)^2$
  - Feedback: "Your square term is right, but there is some mistake with the remaining constant."
- Else if LA "is equivalent to" $(x + 2)^2 - 1$
  - Feedback: "Your answer is correct but not in the required format. Please make sure your answer contains a squared term."
- Else if LA "matches the form of $(x + a)^2 + b$ with placeholders $\{a,\ b\}$
  - Feedback: "Your answer is in the correct format but the values are wrong. Please recheck for arithmetic errors."
- Else (none of the conditions match)
  - Feedback: "Answer is incorrect."

Example Walkthrough 2
Question: Expand $(a + b + c)^2$

- If LA "matches exactly with" $a^2 + b^2 + c^2 + 2ab + 2bc + 2ac$
  - Feedback: "That is Correct!"
- Else if LA "contains at least one of the terms present in" $a^2 + b^2 + c^2 + 2ab + 2bc + 2ac$

    - Feedback: "Your expansion is partially correct. Please recheck for any arithmetic errors."
- Else if LA "contains at least one of the terms present in" $(a+b)^2 + (a+c)^2 + (b+c)^2$
    - Feedback: "Your expansion looks incomplete. Please make sure all parentheses are expanded out."
- Else (none of the conditions match)
    - Feedback: "Answer is incorrect."

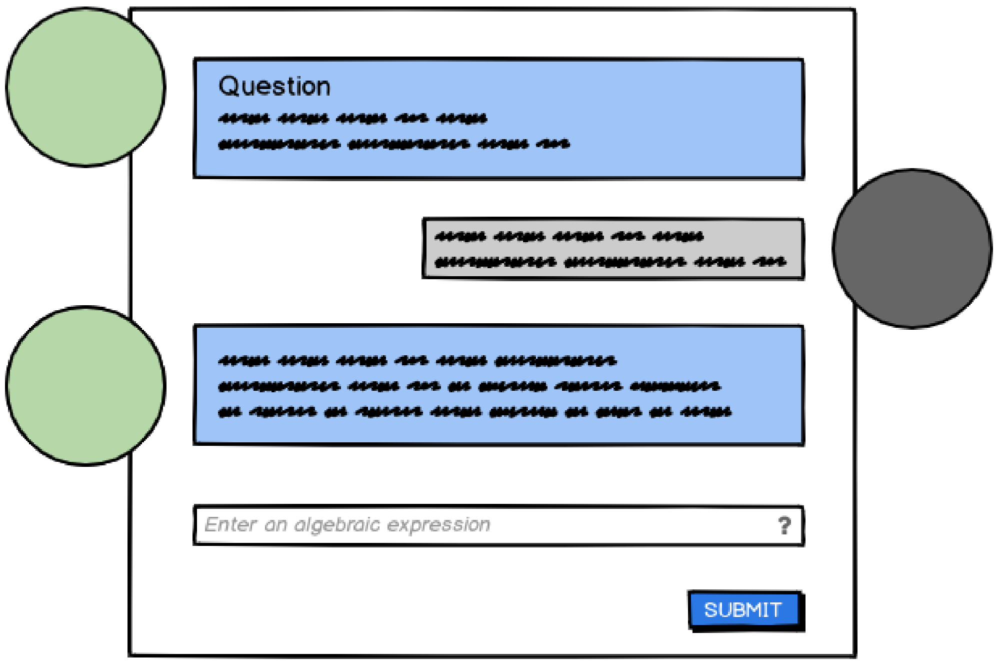
Example Walkthrough 3

Question: Collect common powers of x in the expression $x^4 + x^3y + xz - 3x^3 - x$

- If LA "matches exactly with" $x^4 + x^3(y-3) + x(z-1)$
    - Feedback: "That is Correct!"
- Else if LA "omits at least one of the terms present in" $x(z-1)$
    - Feedback: "You seem to have missed collecting the x terms."
- Else if LA "omits at least one of the terms present in" $x^3(y-3)$
    - Feedback: "You seem to have missed collecting the x^3 terms."
- Else (none of the conditions match)
    - Feedback: "Answer is incorrect."

Learner's Journey

This is how an exploration that uses the Algebraic Expression Input Interaction would look like:

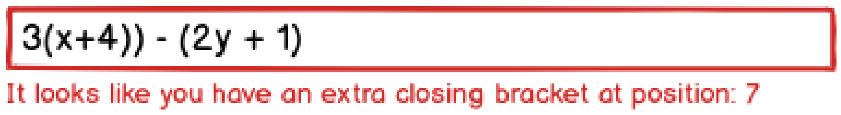


The input for the learner, here, and for the creator in the Add Response modal, would be a WYSIWYG Math Editor which will support intuitive typing of math expressions/equations. The current interaction uses guppy as the math editor and so will the new ones. Guppy provides an on-screen keyboard, with buttons for a variety of symbols. This leads to better user experience for mobile users. It also provides buttons that open up a modal to show comprehensive instructions related to symbols and other controls. These would be particularly helpful to young learners who may not have much experience in typing math expressions/equations through a keyboard.

There will also be on-submit checks for when the learner enters the expression. A warning will be shown if the learner enters an invalid algebraic expression.

IF LEARNER'S ANSWER	WARNING DISPLAYED
Contains no variables(latin letters).	It looks like you have entered only numbers. Make sure to include the necessary variables mentioned in the question.
Contains variables that are not present in the expression provided by the creator.	It looks like you have entered the following variables which are wrong: {{list of invalid variables}}.
Contains any two operators ($+, -, /, \cdot, ^\wedge, .$) consecutively	It looks like you have entered these operators incorrectly: {{first instance of invalid operator sequence}}
Contains an unbalanced bracket sequence.	It looks like you have an extra closing/opening bracket at position: {{1-based index of the first invalid bracket pair}}
Contains any of ($=, <, >$)	It looks like you have entered an equation/inequality. Please enter a math expression instead.
Contains a term that has a denominator that equates to zero.	It looks like you've divided by zero in your expression.

This is an example of a warning being displayed:


3(x+4)) - (2y + 1)

It looks like you have an extra closing bracket at position: 7

Numeric Expression Input Interaction

Creator's Journey

Upon choosing the *Numeric Expression Input* Interaction from the Math tab in the *Choose Interaction* modal, the *Add Response* modal will directly open up, since this interaction needs no customization args. This will allow the creator to provide insightful feedback by anticipating various kinds of learner answers. The modal would look exactly like it's counterpart in the Algebraic Expression Input interaction, as shown here.
The creator would then choose the desired rule out of the given options and then set the response to be shown to the learner if the answer gets validated by the specified rule. Finally, the creator would mention the path where the card redirects.

This interaction would allow purely numeric expressions only. There must be no variables in the expression. This interaction would support the following rules which can detect a wide range of learner answers:

- **MatchesExactlyWith:** "matches exactly with {{x|NumericExpression}}"
  This rule would expect one argument, NumericExpression, that the learner's answer, which is also an NumericExpression, can be compared with.
  The learner's answer would be split into terms by considering $+,-$ as delimiters. This would produce a list of terms. Now, each term in the learner's list would be compared with the creator's list and upon finding the first match, they both would be removed from their respective lists. If eventually, both lists become empty, this rule would return true. Two terms would match if there is an exact string match between parts of the term separated by $*,/$. For instance, $2*3+5$ would match exactly with $5+3*2$.

- **IsEquivalentTo:** "is equivalent to {{x|NumericExpression}}"
  This rule would expect one argument, NumericExpression, that the learner's answer, which is also an NumericExpression, can be compared with.
  The expressions would be evaluated and the mathematical value of the learner's and creator's answer would be compared. If they are equal, this rule would return true.

- **ContainsSomeOf**: "contains at least one of the terms present in {{x|NumericExpression}}"
  This rule would expect one argument, NumericExpression.
  Two sets of terms would be formed by splitting the learner's answer and the creator's input. The splitting would be done according to the definition of numeric terms.
  After splitting, we would get two lists of terms. For each term in the learner's list, it would be compared with every term in the creator's list and if even one of them matches, this

rule would return true. This “match” between terms would be an exact match as mentioned above.

- **OmitsSomeOf**: “omits at least one of the terms present in {{x|NumericExpression}}”
  This rule would expect one argument, NumericExpression.
  Similar to the previous rule, here also, the learner’s answer and the creator’s input would be split into separate terms based on the definition of numeric terms.
  For each term in the creator’s list, it would be compared with every term in the learner’s list and if at least one of them *doesn’t* match, this rule would return true. This would imply that there is at least one term in the creator’s list that is absent in the learner’s list, which means that the learner has omitted that term.
  Here also, the “match” between terms would be an exact match (without expansion).

The following table aims to provide an overview of the kind of validations each rule would support with various examples. These examples would be extremely helpful while writing the unit tests for the rules.

RULE	VALUES OF THE PARAMETERS	ANSWERS	
		ACCEPTED	REJECTED
MatchesExactlyWith matches exactly with {{**x**|NumericExpression}}	**x:** $6-(-4)$	$6-(-4)$ $-(-4)+6$	10 $6+4$ $3*2-(-4)$
	x: $3*10^{-5}$	$3*10^{-5}$ $10^{-5}*3$ $3/10^{5}$	$30*10^{-6}$ 0.00003 $3/(10*10^{4})$
	x: $1000+200+30+4+0.5+0.06$	$1000+200+30+4+0.5+0.06$ $0.06+0.5+4+30+200+1000$	1234.56 $1234+56/10$ $1230+4.56$
	x: $2*2*3*3$	$2*2*3*3$ $2*3*2*3$	$2*2*3*3*1$ $2*2*9$

			$4 * 3^2$ $8/2 * 3 * 3$ 36
IsEquivalentTo Is equivalent to {{**x**|NumericExpression}}	**x:** $6 - (-4)$	10 $6 + 4$ $6 + 2^2$ $3 * 2 - (-4)$ $100/10$	$6 - 4$ $6 + (-4)$ 100
	x: $3 * 10^{-5}$	$3 * 10^{-5}$ 0.00003 $3/(10 * 10^4)$	$3 * 10^5$ $2 * 10^{-5}$ $5 * 10^{-3}$
	x: $1000 + 200 + 30 + 4 + 0.5 + 0.06$	1234.56 $123456/100$ $61728/50$ $1234 + 56/10$ $1230 + 4.56$	123456 $1000 + 200 + 30$
	x: $2 * 2 * 3 * 3$	$2 * 2 * 3 * 3 * 1$ $2 * 2 * 9$ $4 * 3^2$ $8/2 * 3 * 3$ 36	$2 * 2 * 3$ $2 * 3 * 3 * 3$
ContainsSomeOf contains at least one of the terms present in {{**x**|NumericExpression}}	**x:** $6 - (-4)$	$6 + 4$ $-(-4)$ $6 - (-4)$	4 $-6 + 4$ 10
	x: $3 * 10^{-5}$	$3 * 10$ $3/10^5$ 10^{-5}	$30 * 10^{-6}$ $2 * 10^5$ $10^{-6} * 10$
	x: $1000 + 200 + 30 + 4 + 0.5 + 0.06$	$1000 + 234.56$ $1000 + 200 + 30 + 4.56$	1234.56 $123456/100$
	x: $2 * 2 * 3 * 3$	$2 * 2$ $2 * 3 * 3$ $2 * 2 * 3 * 3 * 1$	$4 * 3^2$ 36 $72/2$
	x: $6 - (-4)$	6	$6 - (-4)$

OmitsSomeOf omits at least one of the terms present in {{**x**|NumericExpression}}		$6 - 4$ $6 + 4$	$(-4) + 6$
	x: $3 * 10^{-5}$	$3 * 10^{5}$ $30/10^{6}$	$3 * 10^{-5}$ $3/10^{5}$ $10^{-5} * 3$
	x: $1000 + 200 + 30$ $+ 4 + 0.5 + 0.06$	$1000 + 200 + 30$ $+ 4 + 0.56$ 1234.56	$1000 + 200 + 30$ $+ 4 + 0.5 + 0.06$ $0.06 + 0.5 + 4$ $+ 30 + 200 + 1000$
	x: $2 * 2 * 3 * 3$	$2^{2} * 3 * 3$ $4 * 9$	$2 * 2 * 3 * 3$ $2 * 2 * 3 * 3 * 1$

Numeric Term: Each part of an expression separated by anyone of $+, -, *, /$ signs is considered to be a numeric term. If present, a parenthesized portion of the expression is considered to be a single term unless the parentheses are redundant as mentioned in the algebraic term definition.
These would be considered as single numeric terms: $2,\ 2^{2},\ 5/4,\ (4\ +\ 2)^{3}$.
These would not be considered as single numeric terms: $3\ +\ 5,\ 3^{2}\ -\ 2,\ 3 * 2,\ (2\ +\ 4)/3$.
Examples of splitting an expression:

- $1\ +\ 2\ +\ 3 \Rightarrow [1,\ 2,\ 3]$
- $(3\ +\ 2)\ -\ 4 \Rightarrow [(3,\ \ 2,\ \ -4]$
- $4 * 5\ -\ 6 \Rightarrow [4,\ 5,\ \ -6]$
- $2/(3\ -\ 5)\ +\ 4 \Rightarrow [2,\ (3\ -\ 5)^{-1},\ 4]$
- $4 * (5\ +\ 3) \Rightarrow [4,\ (5\ +\ 3)]$

Example Walkthrough 1

Question: "Now, Miguel," Ms. Grocer said, "let's say you have $3, and then you buy a turkey for $24. How can you show this situation by adding a positive number and a negative number?" (Don't do any math, just write out the calculation. You don't need to include the dollar signs.) (*Taken from the "What is a negative number" exploration which uses the Text Input interaction).*

- If LA "matches exactly with" $3 + (-24)$
    - Feedback: "That is Correct!"
- Else if LA "matches exactly with" $3 + 24$
    - Feedback: "That is incorrect. Notice that Miguel has bought a turkey for $24 which indicates subtraction of money and you seem to have added it."
- Else if LA "is equivalent to" -21

    - Feedback: “Your answer is correct but not in the required format. You need to write a numeric expression to show the operations that have taken place.”
- Else (none of the conditions match)
    - Feedback: "Answer is incorrect."

Example Walkthrough 2

Question: Break down 1234.56 into component place values.

- If LA “matches exactly with” $1000 + 200 + 30 + 4 + 0.5 + 0.06$
    - Feedback: "That is Correct!"
- Else if LA “contains at least one of the terms present in” $1000 + 200 + 30 + 4 + 0.5 + 0.06$
    - Feedback: "Your expansion is incomplete. Make sure you’ve broken down the decimals into tenths and hundredths places as well."
- Else if LA "is equivalent to" 1234.56
    - Feedback: “Your answer is correct but not in the required format. You need to break the number down to its component place values.”
- Else (none of the conditions match)
    - Feedback: "Answer is incorrect."

Example Walkthrough 3

Question: Write the prime factorization of 36.

- If LA “matches exactly with” $2 * 2 * 3 * 3$
    - Feedback: "That is Correct!"
- Else if LA “omits at least one of the terms present in” $2 * 2 * 3 * 3$
    - Feedback: "Please make sure that your answer contains prime factors."
- Else if LA "is equivalent to" 36
    - Feedback: “Your answer is correct but not in the required format. You need to factorize 36 with its prime factors.”
- Else (none of the conditions match)
    - Feedback: "Answer is incorrect."

Learner’s Journey

An exploration that uses the Numeric Expression Input interaction would look exactly like the Algebraic Expression Input interaction as shown here.

There will also be on-submit checks for when the learner enters the expression. A warning will be shown if the learner enters an invalid numeric expression.

IF LEARNER'S ANSWER	WARNING DISPLAYED
Contains variables(latin letters).	It looks like you have entered some variables. Please make sure that your answer contains numbers only.
Contains any two operators ($+,-,/,\cdot,$ ^ , .) consecutively	It looks like you have entered these operators incorrectly: {{first instance of invalid operator sequence}}
Contains an unbalanced bracket sequence.	It looks like you have an extra closing/opening bracket at position: {{1-based index of the first invalid bracket pair}}
Contains any of ($=,<,>$)	It looks like you have entered an equation/inequality. Please enter a math expression instead.
Contains a term that has a denominator that equates to zero.	It looks like you've divided by zero in your expression.

The warning message would look exactly like the one shown here for the Algebraic Expression Input interaction.

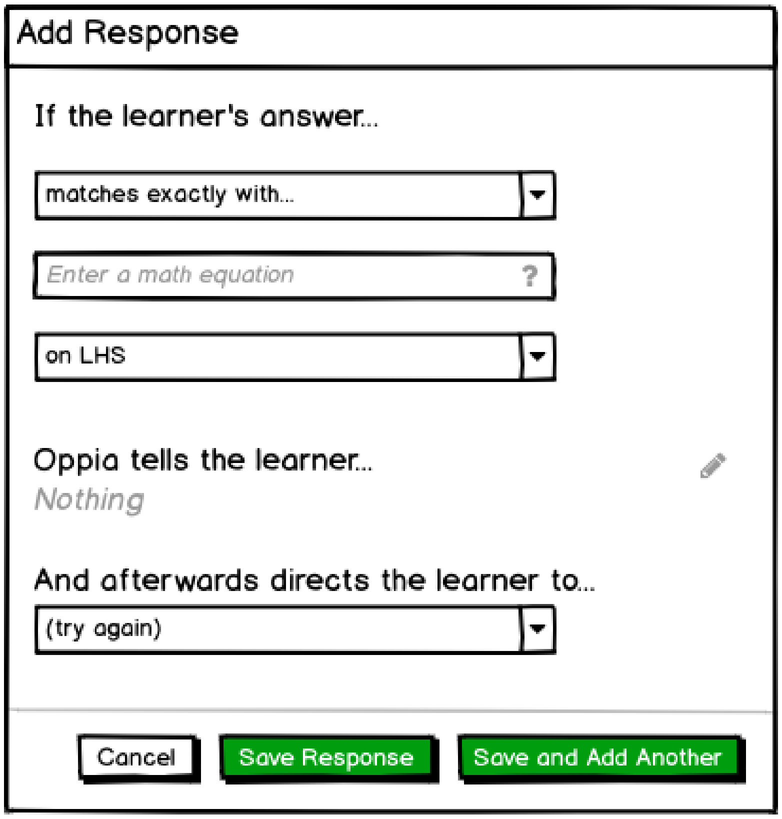
Math Equation Input Interaction

The Math Equation Input interaction is meant to be an extended version of the Algebraic Expression Input interaction, specifically, to support the validation of equations. This is why there will be some overlap between the two interactions.

Creator's Journey

Upon choosing the *Algebraic Equation Input* interaction from the Math tab in the *Choose Interaction* modal, the *Add Response* modal will directly open up, since this interaction needs no customization args. This will allow the creator to provide insightful feedback by anticipating various kinds of learner answers. The modal would look something like this:

Rules: MatchesExactlyWith, ContainsSomeOf, OmitsSomeOf



Rules: IsEquivalentTo, MatchesWithGeneralForm

Same as the corresponding mocks in Algebraic Expression Input Interaction. The only difference would be that this would accept equations instead of expressions.

This interaction would also support the following rules:
Note: The general comparison of the answer with the given equation will be done by splitting the equation into LHS and RHS, and then treating each side as a separate expression. We can then perform checks similar to the Algebraic Expression Input Interaction.

- **MatchesExactlyWith, ContainsSomeOf, OmitsSomeOf**: These rules will all accept two arguments, *MathEquation* and *PositionOfTerms*. The position argument will be a can have one of the following values:
    - **on LHS**: Only the left-hand sides will be compared between the learner's answer and the creator's input.
    - **on RHS**: Only the right-hand sides will be compared between the learner's answer and the creator's input.
    - **on both sides**: Both sides of the equations will be compared independently.

This is basically “on LHS **AND** on RHS”.

  - **with reordering allowed around ‘=’**: All terms will be brought on one side for both the equations and then that side will be considered as an expression and then the comparison will take place. With this argument, $x - y = 0$ and $x = y$ would be considered to match exactly with each other.
- **IsEquivalentTo**: This is similar to the reordering argument stated above, but with an allowance to expand terms on either side of the equation.
- **MatchesWithGeneralForm**: This rule would be implemented same as its counterpart in the Algebraic Expression Input Interaction. That process would be applied to both sides of the equation independently.

The following table aims to provide an overview of the kind of validations each rule would support with various examples. These examples would be extremely helpful while writing the unit tests for the rules.

RULE	VALUES OF THE PARAMETERS	ANSWERS	
		ACCEPTED	REJECTED
MatchesExactlyWith matches exactly with {{**x**|MathEquation}} {{**y**|PositionOfTerms}}	**x:** $y = mx + c$		
	y: on LHS	$y = mx + c$ $y = mx^2 + c$ $y = 0$ $y = mx - c$ $y^2/y = mx + c$	$y - mx = c$ $y^2 = mx + c$ $y/3 = mx + c$ $x = my + c$ $mx + c = y$ $y/x = m + c/x$ $0 =- y + mx + c$ $y - mx - c = 0$ $y/(mx + c) = 1$
	y: on RHS	$y = mx + c$ $y^2 = mx + c$ $0 = mx + c$	$y = mx^2 + c$ $y = mx - c$ $y = mx$ $mx + c = y$ $y = mx$ $0 =- y + mx + c$ $y = mx + b$ $y - mx - c = 0$ $y^2 = (mx + c)^2$

	y: on both sides	$y = mx + c$ $y = c + mx$ $y^2/y = mx + c$	$y - mx = c$ $mx + c = y$ $y = mx$ $y - mx = c$ $x = my + c$ $y/(mx + c) = 1$ $y + mx + c = 0$ $y - mx - c = 0$ $y^2 = mxy + cy$ $y = (mx^2 + cx)/x$ $y^2 * y^{-1} = mxc$
	y: with reordering allowed around '='	$y - mx = c$ $y - mx - c = 0$ $mx + c = y$ $y - mx - 2c =- c$	$y^2 - mx = c$ $y = mx$ $y = mx - c$ $y + c = mx$ $y - mx + c = 0$ $0 = y + mx + c$ $y - mx - c = 0$ $(y - mx - c)^2 = 0$
ContainsSomeOf contains at least one of the terms present in {{**x**|MathEquation}} {{**y**|PositionOfTerms}}	**x:** $(x - h)^2 + (y - k)^2 = r^2$		
	y: on LHS	$(x - h)^2 + (y - k)^2 = r^2$ $(x - h)^2 - 1 = r$ $(y - k)(y - k) = 1$ $(x - h)^2 + (y - k)^2$ $- r^2 = 0$	$x^2 + (y + k)^2 = r$ $r^2 = (x - h)^2 + (y - k)^2$ $x + (y + k)^2 = r^2$ $x^2 + y^2 = r^2$
	y: on RHS	$(x - h)^2 + (y - k)^2 = r^2$ $(x - h)^2 = r^2 - (y - k)^2$ $0 = r * r$ $(x - h)^2 = r^3/r$	$x + y^2 = r(r - 1)$ $r^2 = x^2 + y^2$ $(x - h)^2 + y^2 = - r^2$
	y: on both sides	$(x - h)^2 + (y - k)^2 = r^2$ $(x - h)^2 = r^2 + 2$ $x^2 + (y - k)^2 = r * r$	$(x + k)^2 + (y - h)^2 = r^2$ $x^2 + y^2 = 2r^2$ $x^2 - (y - k)^2 = r^2$ $(x - h)^2 + (y - k)^2 = r$
	y: with reordering allowed around '='	$(x - h)^2 = 0$ $0 = r^2$ $(x - h)^2 + y^2 - r^2 = 0$ $x^2 - r^2 = y^2$	$x = h$ $x^2 + y^2 - 2r^2 = 0$ $((x - h)^2 + (y - k)^2)/r^2$ $= 1$

OmitsSomeOf omits at least one of the terms present in {{**x**|MathEquation}} {{**y**|PositionOfTerms}}	**x:** $x^2 + y^2 = z^2$		
	y: on LHS	$x^2 - y^2 = z^2$ $x\ + y^2 = z^2$ $-x^2 - y^2 =- z^2$ $z^2 = x^2 + y^2$	$x^2 + y^2 = z^2$ $x^2 + y^2 = z^2$ $x^2 + y^2 = 3z$ $y^2 + x^2 = z^2$
	y: on RHS	$x^2 + y^2 = z$ $z^2 = x^2 + y^2$ $x^2 + y^2 = -z^2$	$x^2 + y^2 = z * z$ $x^2 = z^2 - y^2$ $0 = z^2$ $x^2y^2 = z^2 - 52z$ $x^2 + y^2 = \sqrt{z^4}$
	y: on both sides	$x^2 + y\ = zy$ $z^2 = x^2 + y^2$ $x\ + y\ = z$	$x^2 + y^2 = z$ $x^2 + 2y^2 = z^2$ $x^2 + y^2 = z^2$
	y: with reordering allowed around '='	$x^2 - y^2 = z^2$ $x\ + y^2 = z^2$ $x^2 + y^2 = 3z^2$	$x^2 + y^2 = z^2$ $x^2 + y^2 - z^2 = 0$ $y^2 = z^2 - x^2$
IsEquivalentTo is equivalent to {{**x**|MathEquation}}	**x:** $(2x + 1)(x - 3) = 0$	$2x^2 - 6x + x - 3 = 0$ $2x^2 - 6x = 3 - x$ $0 = (2x + 1)(3 - x)$ $2x^2 - 6x + x = 3$ $-2x^2 + 5x + 3 = 0$ $(2x + 1)(-x + 3) = 0$	$x = 3$ $2x + 1 = 0$ $x - 3 = 0$ $x =- 1/2$

	x: $13 + 2w = 34$	$13 + 2w = 34$ $26 + 4w = 68$ $6.5 + w = 17$ $13 + 2w - 34 = 0$ $w = 17 - 6.5$ $w = 10.5$ $w = 10 + 1/2$ $10.5 = w$ $w = 21/2$ $(13 + 2w)/34 = 1$ $w/10.5 = 1$ $w = 21\ 1/2$	$13 + 2w = 0$ $13 - 2w = 34$ $(13 + 2w)^2 = 1156$ $w = 10\ 1/2$
MatchesWithGeneralForm matches the form of {{**x**|MathEquation}} with placeholders {{**y**|SetOfPlaceholders}}	**x:** $x^2 = ay$ **y:** $\{a\}$	$x^2 = 16y$ $x^2 = 3.14y$ $x^2 = 1y$ $x^2 = log(4)y$ $x^2 =- 4y$ $x^2 = 0y$ $x^2 = (3/4)y$ $x^2 = (4 * 4)y$ $x^2 = 4^2y$ $x^2 = \pi y$ $x^2 = y$	$y^2 = 16x$ $x^2 - 16y = 0$ $x = 2\sqrt{y}$ $4x^2 = 3y$

Example Walkthrough 1

Question: Write the standard equation of a line in slope-intercept form.

- If LA "matches exactly with" $y = mx + c$ "on both sides"
  - Feedback: "That is Correct!"
- Else if LA "matches exactly with" $y = mx + c$ "on RHS"
  - Feedback: "The RHS of your answer is perfect. Make sure the LHS is correct as well."
- Else if LA "matches exactly with" $y = mx + c$ "on LHS"
  - Feedback: "The LHS of your answer is perfect. Make sure the RHS is correct as well."
- Else if LA "matches exactly with" $y = mx + c$ "with reordering allowed around '='"
  - Feedback: "Your answer is correct but not formatted properly. Make sure it is in the slope-intercept form."
- Else (none of the conditions match)
  - Feedback: "Answer is incorrect."

Example Walkthrough 2

Question: Formulate the cost equation if the per-unit cost of a shirt is 500, and the per-unit cost of a pair of socks is 100. The total amount spent is 3000. Assume that x shirts and y pairs of socks are bought.

- If LA "is equivalent to" $500x + 100y = 3000$
  - Feedback: "That is Correct!"
- Else if LA "matches exactly with" $500y + 100x = 3000$ "on both sides"
  - Feedback: "Your answer is correct but you seem to have swapped the variables. Remember that x shirts and y pairs of socks are bought."
- Else if LA "contains at least one of the terms present in" $500yx + 100y = 3000$ "on LHS"
  - Feedback: "Your answer is almost correct. Make sure that you've factored in both, the shirt as well as the sock, costs."
- Else if LA "contains at least one of the terms present in" $500x + 100y = 3000$ "on RHS"
  - Feedback: "Your answer is almost correct. Note that the total cost is 3000."
- Else if LA "omits at least one of the terms present in" $500x + 100y = 3000$ "on both sides"
  - Feedback: "Your answer seems to be missing important terms on both sides of the equation. Please recheck."
- Else (none of the conditions match)
  - Feedback: "Answer is incorrect."

Example Walkthrough 3

Question: Write an equation of a line in "slope-intercept form" or "intercept form" that passes through (0, 2) and (3, 0).

- If LA "matches exactly with" $y = -2/3x + 2$ "on both sides"
  - Feedback: "Good job! That is the slope-intercept form!"
- Else if LA "matches the form of" $y = mx + c$ with placeholders as $\{m, c\}$
  - Feedback: "Your answer is in the slope-intercept form but the values are incorrect."
- Else if LA "matches exactly with" $x/3 + y/2 = 1$ "on both sides"
  - Feedback: "Good job! That is the intercept form!"
- Else if LA "matches the form of" $x/a + y/b = 1$ with placeholders as $\{a, b\}$:
  - Feedback: "Your answer is in the intercept form but the values are incorrect."
- Else if LA "matches loosely with" $2x + 3y = 6$
  - Feedback: "The equation of the line is correct but it is not in any of the required forms."
- Else (none of the conditions match)
  - Feedback: "Answer is incorrect."

Learner’s Journey

An exploration that uses the Math Equation Interaction would look exactly like the Algebraic Expression Input Interaction as shown here. The only difference would be in on-submit checks. The following warnings will be shown if the user enters an invalid math equation.

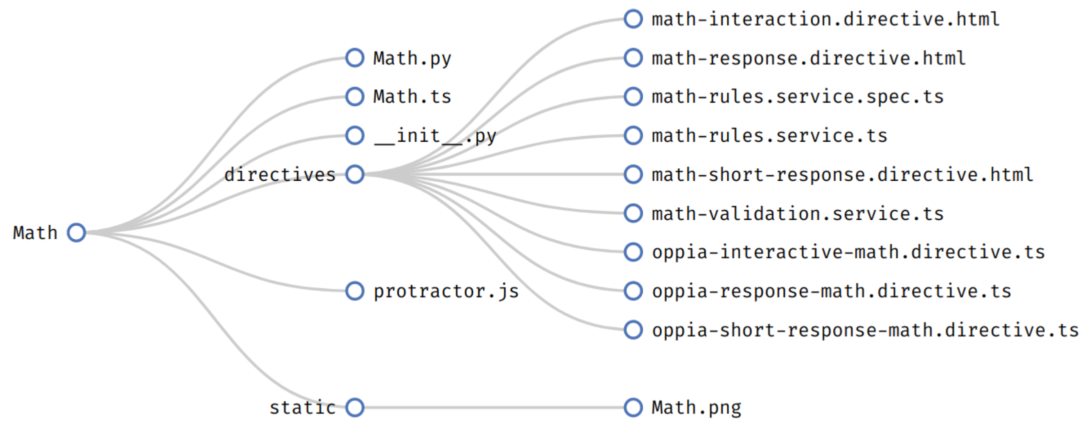
IF LEARNER’S ANSWER	WARNING DISPLAYED
Contains variables that are not present in the equation provided by the creator.	It looks like you have entered the following variables which are wrong: {{list of invalid variables}}.
Contains any two operators ($+, -, /, \cdot, \hat{}\ , .$) consecutively	It looks like you have entered these operators incorrectly: {{first instance of invalid operator sequence}}
Contains unbalanced bracket sequence.	It looks like you have an extra closing/opening bracket at position: {{1-based index of the first invalid bracket pair}}
Has zero or more than one $=$ signs or contains any of ($<, >$)	Your equation contains none or too many = signs, or is an inequality.
$=$ is present in the beginning or at the end of the equation	Either LHS or RHS of your equation is empty.
Contains no variables(latin letters) on either side of the equation.	It looks like you’ve entered only numbers in the equation. Make sure your equation contains some variables as well.
Contains a term that has a denominator that equates to zero.	It looks like you’ve divided by zero in your equation.

The warning will also look exactly like the one shown in the previous interaction here.

Technical Design

Architectural Overview

To create a new interaction, a directory will have to be created under `extensions/interactions/` for the new interaction with the corresponding name, i.e. MathExpressionInput or MathEquationInput.
The *general* structure of this directory would be as follows:


Math
directives
Math.py
Math.ts
__init__.py
protractor.js
static
math-interaction.directive.html
math-response.directive.html
math-rules.service.spec.ts
math-rules.service.ts
math-short-response.directive.html
math-validation.service.ts
oppia-interactive-math.directive.ts
oppia-response-math.directive.ts
oppia-short-response-math.directive.ts
Math.png

For each interaction directory, "math" would be replaced with the respective interaction id. Other than that, the structure would be exactly as shown above. A more detailed overview of the contents of these files, for each interaction, is mentioned in the Implementation/Testing Approach section below.

After these additions, the interaction id of the new interaction would be added in the constants files under the Math tab in `ALLOWED_INTERACTION_CATEGORIES` to activate the interaction.

Apart from this, the following files that would reference the new interaction would also be modified:

- `core/tests/protractor_utils/ExplorationEditorMainTab.js`
- `core/templates/domain/exploration/SolutionObjectFactory.ts`
- `extensions/interactions/interaction_specs.json`
- `extensions/interactions/interactionsRequires.ts`
- `extensions/interactions/rule_templates.json`

As for additions in the storage, five new data types will have to be added, *AlgebraicExpression, SetOfPlaceholders, NumericExpression, MathEquation* and *PositionOfTerms*. To add these, new classes will be created in `extensions/objects/models/objects.py`. A detailed explanation of the structure of these classes is provided in their respective sections.

Third party libraries

To implement the single rule that the current math interaction supports, the math-expressions library is being used. But to implement the new rule-set proposed above, a more advanced third-party library will have to replace math-expressions.
Currently, three major open-source libraries support symbolic math for JavaScript. I have listed the pros and cons of each of them in the following table:

LIBRARY	PROS	CONS
nerdamer	● Tons of features ranging from basic algebra to trigonometry and advanced calculus. (This would be useful in the future if and when more advanced rules need to be introduced). ● Functionality to extract variables from an expression (Useful for checking the presence of variables in the learner's answer). ● Functionality to compare two expressions. ● Functionality to algebraically add or divide math expressions. ● Functionality to expand and factor polynomials. ● Active community of developers. ● Reasonably lightweight (minified bundle size: 308 kB). And that's for the entire library, we'll probably only need to import the Algebra module along with the core module.	
mathjs	● Already present as a dependency in Oppia. ● Supports function chaining which makes the code cleaner. ● Support for very large numbers with BigNumber. ● Sandboxed environment to execute expressions which makes XSS impossible.	● Relatively heavy (minified bundle size of 632 kB). ● Only supports basic algebraic functions (simplify, derivative and rationalize).
Algebrite	● Lightweight (minified bundle size: 343kB) ● A vast range of functions available from basic algebra to advanced trigonometry and linear algebra.	● Documentation is very vague.

Considering the requirements, I propose to use nerdamer as the third-party library to implement the aforementioned rules for the interactions.
To integrate nerdamer in the codebase, the following files will have to be modified:

- `app_dev.yaml`
- `manifest.json`
- `dependency_html.html`
- `third_party/static/` (the nerdamer files will be added here)
- `core/tests/karma.conf.ts`

Note: Only Algebra.js and nerdamer.core.js files need to be added for the current use-case.

Implementation Approach

I will now describe the implementation of each interaction separately, starting with the *Algebraic Expression Input* Interaction. But, since all of them are new interaction additions, the process will considerably overlap.

Algebraic Expression Input Interaction

The first step would be to introduce two new data types, *AlgebraicExpression* and *SetOfPlaceholders*, that this interaction will require. The classes for these objects would be added in the `objects.py` file under `extensions/objects/models/`. They would look something like this:

```
class AlgebraicExpression(BaseObject):
    """Algebraic expression class."""

    description = 'A string for algebraic expression.'
    default_value = ''

    SCHEMA = {
        'type': 'unicode',
        'validators': [{
            'id': 'is_valid_algebraic_expression'
        }]
    }
```

```
class SetOfPlaceholders(BaseObject):
    """Class for set of placeholders."""

    description = 'A set (a list with unique elements) of placeholders.'
    default_value = []

    SCHEMA = {
        'type': 'list',
        'items': 'unicode',
        'validators': [{
            'id': 'is_uniquified'
        }, {
            'id': 'is_each_element_latin_letter'
        }]
    }
```

The type will be unicode since the math editor(guppy) will store the input in ASCIIMath format, which is a unicode string. This string can then be parsed and converted to a nerdamer expression object which would be easy to work with.
The validator functions for these classes will be added as static methods inside the Validators class present in `schema_utils.py`.

- `is_valid_algebraic_expression`: This method will return true if the expression:
  - Contains at least one variable(latin letter).
  - Does not contain multiple operators $(+,\ -,\ /,\ *, ^\wedge,\ .)$ consecutively.
  - Contains balanced bracket sequence.
  - Does not contain any of $(=,\ <,\ >)$.
  - Contains no terms that have a denominator that equates to zero.
- `is_each_element_latin_letter`: This method will return true if every element in the set is a single latin letter.

Now, the current math interaction provides the guppy math editor only to the learner, and the creator is required to enter a LaTeX string while creating the exploration. This is infeasible since not all creators would know LaTeX.
This is why we should provide the math editor to the creators as well. To achieve this, and to refactor the guppy code (which is a bit all over the place at the moment), we create a new math editor directive in `extensions/objects/templates/`. The functionality of this directive would be to initialize a new guppy instance with the appropriate configurations. Most of the configurations would be the same as the ones that are currently in production, the only major thing that would need to be added is the display of the helper buttons. These buttons are

inbuilt in the guppy editor and they show a modal with comprehensive instructions for entering all kinds of symbols.

The next step would involve creating the AlgebraicExpressionInput directory under `extensions/interactions/` which will contain the following files that would define the behavior of the interaction:

`/AlgebraicExpressionInput.py`: This will be the configuration file for the interaction where attributes such as name, description, dependencies, etc. are to be added. This file will contain a AlgebraicExpressionInput class that will inherit the BaseInteraction class which contains all definitions of said attributes.
The AlgebraicExpressionInput class will look like:

```python
class AlgebraicExpressionInput(base.BaseInteraction):
    """Interaction for algebraic expression input."""

    name = 'Algebraic Expression Input'
    description = 'Allows learners to enter algebraic expressions.'
    display_mode = base.DISPLAY_MODE_INLINE
    is_trainable = False
    _dependency_ids = ['guppy', 'nerdamer']
    answer_type = 'AlgebraicExpression'
    can_have_solution = True
    show_generic_submit_button = True

    _customization_arg_specs = []
```

Along with this, the attributes of the interaction would also be added in the `interaction_specs.json`.

`/directives/`: This directory will contain the files for

- *Three directives*: interaction, response, and short response.
- *Two services*: rules and validation.

The three directives will each have an HTML template bound to them to define the view of the directive.

The validation service will ensure that the algebraic expression entered by the creator while creating the exploration is valid and does not contradict any other rules that may have been previously added. Since this interaction contains no customization args, it will only return a list of warnings specific to the rule-set provided by the creator.

The validation service will have the `getAllWarnings` function. Since none of these interactions have customization arguments, the `getCustomizationArgsWarnings` function will return an empty list.
The `getAllWarnings` function will return a list of warnings based on the answer provided by the creator. The answer provided by the creator will be validated based on the on-submit criteria for the learner as mentioned here and the corresponding warning will be displayed. The "Contains variables that are not present in the expression provided by the creator." check would obviously be irrelevant here. Also, a validation that checks if the mentioned placeholders are present in the general form expression provided by the creator would be added with an appropriate warning. This can be implemented by using nerdamer's `variables()` function to check for the variables present in the general form expression.

Along with this, the relative order of the rule-set will also be checked. A warning will be generated if the IsEquivalentTo rule precedes any of the other rules and the answer groups of both rules match. Since the IsEquivalentTo rule would make the others obsolete.
The implementation would be done by iterating over the rules in the provided order and by checking for relative positions of the rules.

The interaction directive will define the behavior of the interaction. It will contain the functionality for validating the learner's answer upon submission and for throwing relevant warnings if necessary. The exact list of warnings to be thrown based on a given condition is mentioned in a table here.
The following is a brief description of how these validations would be implemented:

- `isNumeric`: This will return true if the list output from nerdamer's `variables()` function has a length of zero.
- `containsInvalidVariables`: Nerdamer's `variables()` function will be used to retrieve all variables present in the learner's answer and this list will be compared with the list of variables present in the creator's answer. If the learner's answer contains variables not present in the creator's answer, this function will return true.
- `containsConsecutiveOperators`: A simple linear scan to check the consecutive indices of the answer string would suffice.
- `imbalancedBracketSequence`: Nerdamer will throw a `ParityError` while parsing the answer string if there is an imbalance in the bracket sequence.
- `isEquation`: A simple linear scan of the answer string to check for the presence of $(=, <, >)$ would suffice.
- `dividesByZero`: Split the expression into terms and evaluate each term's denominator.

Moving on to the rules service. This service will convert the learner's answer string into a nerdamer object and then verifications will be done according to the rules provided by the creator.

The first step would be to add the rule descriptions in `extensions/interactions/rule_templates.json`.

The exact description of each rule is mentioned in this table.

After describing the rules, they would be defined in the rules service. An overview of how each rule will be implemented is given below:

- `MatchesExactlyWith`: The comparison will be quite straightforward. The learner's answer will be compared with the creator's answer using nerdamer's `eq()` function. This function will be called without expanding either of the expressions since this is an *exact* match.

- `IsEquivalentTo`: The comparison will be done by first expanding both expressions using nerdamer's `expand()` function and then comparing using nerdamer's `eq()` function.

- `ContainsSomeOf`: The learner's answer would be split into separate terms by $+,-$ sign using the `getTerms()` defined here. Now we have two lists of terms: learnerTerms, creatorTerms. By iterating over one of the lists, we would compare each term with all terms present in the other list. If at least one of them matches this rule would return true.
  The match between terms here would be carried out by calling nerdamer's `eq()` function on the terms without expanding them.

- `OmitsSomeOf`: The implementation of this rule would be almost the same as the ContainsSomeOf rule mentioned right above. Here, we would iterate on the creatorTerms list and if there is at least one term that *doesn't* match with any of the learnerTerms, this rule would return true. The truth of this condition implies that the learner's answer has missed at least one term that the creator's expression possesses.

- `MatchesWithGeneralForm`: The learner's answer would be split into terms similar to the way that's mentioned in the rules above, the only difference being, along with $+$ and $-$, $*$ and $/$ would also be considered as the delimiters for splitting the expression. (This would require minor changes to the `getTerms()` function). This rule would use the `termsMatch()` helper function mentioned here. This is how the overall procedure would look like:

```
MatchesWithGeneralForm(
  answer: MathExpression,
  inputs: {x: MathExpression, y: SetOfPlaceholders}): boolean {
    let answerTerms: Array<string> = getTerms(nerdamer(answer).text());
    let generalTerms: Array<string> = getTerms(nerdamer(inputs.x).text());

    if (answerTerms.length !== generalTerms.length) return false;

    // Iterating in reverse because the length of the list
    // changes during iterations.
    for (let i = answerTerms.length - 1; i >= 0; --i) {
      for (let j = generalTerms.length - 1; j >= 0; --j) {
        if (termsMatch(answerTerms[i], generalTerms[i], inputs.y)) {
          answerTerms.splice(i, 1);
          generalTerms.splice(j, 1);
          break;  // If match found, move on to the next answer term.
        }
      }
    }

    // Return true iff all terms got matched.
    return answerTerms.length === 0;
}
```

Helper functions

`getTerms( )`: This will split an expression using $+,-$ as delimiters.

```
function getTerms(expression: string): Array<string> {
    let listOfTerms: Array<string> = [];
    let currentTerm: string = '';
    let bracketBalance: number = 0;

    for (let i = 0; i < expression.length; ++i) {
        const currentVal = expression[i];
        if (currentVal === '(' || currentVal === ')')
            bracketBalance += (currentVal === '(') ? 1 : -1;

        // Split term only if we are not inside parentheses.
        if (bracketBalance === 0 && (currentVal === '+' || currentVal === '-')) {
            if (currentTerm.length !== 0)
                listOfTerms.push(currentTerm);
            // Adding a `-` symbol since the next term will be negative
            currentTerm = (currentVal === '-') ? '-' : '';
        } else {
            currentTerm += currentVal;
        }
    }
    listOfTerms.push(currentTerm);
    return listOfTerms;
}
```

`termsMatch()`: This would check if the answer term matches with the general term. This is done by checking if only the placeholders and constants remain after dividing or subtracting both terms. (*Note*: This is not a completely accurate validation as terms that have placeholders in the power wouldn't pass this validation, but this covers most cases)

```
function termsMatch(
  term1: string, term2: string,
  placeholders: Array<string>): boolean {
    // try dividing
    let divRes: string = nerdamer(term1).divide(term2).text();
    let divVars: Array<string> = nerdamer(divRes).variables();

    if (divVars.every(v => placeholders.includes(v)))
      return true;

    // try subtracting
    let subRes: string = nerdamer(term1).subtract(term2).text();
    let subVars: Array<string> = nerdamer(subRes).variables();

    return subVars.every(v => placeholders.includes(v));
}
```

Numeric Expression Input Interaction

The first step would be to introduce a new data type, *NumericExpression* that this interaction will require. The class for this object would be added in the `objects.py` file under `extensions/objects/models/`. It would structurally resemble the AlgebraicExpression class shown here. The difference here would be that this class will have a different validator: `is_valid_numeric_expression`: This method will return true if the expression:

- Does not contain any variables(latin letters).
- Does not contain multiple operators $(+,\ -,\ /,\ *, ^\wedge,\ .)$ consecutively.
- Contains balanced bracket sequence.
- Does not contain any of $(=,\ <,\ >)$.
- Contains no terms that have a denominator that equates to zero.

The next step would involve creating the NumericExpressionInput directory under `extensions/interactions/` which will contain the following files that would define the behavior of the interaction:

`/NumericExpressionInput.py`: This will be the configuration file for the interaction where attributes such as name, description, dependencies, etc. are to be added. This file will contain a NumericExpressionInput class that will inherit the BaseInteraction class which contains all definitions of said attributes. This class would also structurally resemble the AlgebraicExpressionInput class shown here.

`/directives/`: This directory will contain the files for

- *Three directives*: interaction, response, and short response.
- *Two services*: rules and validation.

The three directives will each have an HTML template bound to them to define the view of the directive.

The validation service will ensure that the numeric expression entered by the creator while creating the exploration is valid and does not contradict any other rules that may have been previously added. Since this interaction contains no customization args, it will only return a list of warnings specific to the rule-set provided by the creator.

The interaction directive will define the behavior of the interaction. It will contain the functionality for validating the learner's answer upon submission and for throwing relevant warnings if necessary. The exact list of warnings to be thrown based on a given condition is mentioned in a table here.
The following is a brief description of how these validations would be implemented:

- `isNumeric`: This will return true if the list output from nerdamer's `variables()` function has a length of zero.
- `containsConsecutiveOperators`: A simple linear scan to check the consecutive indices of the answer string would suffice.
- `imbalancedBracketSequence`: Nerdamer will throw a `ParityError` while parsing the answer string if there is an imbalance in the bracket sequence.
- `isEquation`: A simple linear scan of the answer string to check for the presence of $(=, <, >)$ would suffice.
- `dividesByZero`: Split the expression into terms and evaluate each term's denominator.

Moving on to the rules service. This service will convert the learner's answer string into a nerdamer object and then verifications will be done according to the rules provided by the creator.
The first step would be to add the rule descriptions in `extensions/interactions/rule_templates.json`.
The exact description of each rule is mentioned in this table.

After describing the rules, they would be defined in the rules service. An overview of how each rule will be implemented is given below:

- `MatchesExactlyWith`: The following steps will have to be carried out for this rule:
    1. Split the expression with $+, -$ as delimiters using the `getTerms()` defined here.
    2. Now we have two lists of terms: learnerTerms, creatorTerms. Iterating through the learnerTerms list we try to match it with one of the creatorTerms. If all learnerTerms match with a corresponding creatorTerm, this rule returns true. Here, for checking if two terms match, we would split each term by $*, /$ and then check for an exact string match. For instance, $3 * 10^{-5}$ would be split into $[3,\ 10^{-5}]$ and thus would match exactly with $3 * 10^{-5},\ 10^{-5} * 3,\ 3/10^{5}$. Note, $3/10^{5}$ would also get split into $[3,\ 10^{-5}]$.

- `IsEquivalentTo`: Both the expressions will be evaluated by parsing the expressions with nerdamer. This will return real values which can then be compared for determining mathematical equivalence.

- `ContainsSomeOf`: This would also be implemented in the same way that the ContainsSomeOf rule has been implemented in the Algebraic Expression Input interaction. The difference would be in how the term match is done. The term match here, would be done by splitting on the basis of $*, /$ and then carrying out an exact string match for each part of the term. This is the same as the exact match validation mentioned here.

- `OmitsSomeOf`: This would also be implemented in the same way that the OmitsSomeOf rule has been implemented in the Algebraic Expression Input interaction. Here also, the match will be an exact match.

Note: After these two interactions have been successfully integrated into the Oppia codebase along with its tests, the current math input interaction will be disabled by removing the interaction id from the constants files. This will ensure that all new explorations are created only using new interactions.

Math Equation Input Interaction

This interaction will introduce the *MathEquation* and *PositionOfTerms* data types. The classes for these objects would be added in the `objects.py` file under `extensions/objects/models/`. The *MathEquation* class would structurally resemble the *AlgebraicExpression* class which is defined here. The difference between them would be that this class would have `is_valid_math_equation` validator instead of `is_valid_algebraic_expression`. The validator would be added as a static method inside the Validators class present in `schema_utils.py`.

- `is_valid_math_equation`: This method will return true if expression
  - Does not contain multiple operators $(+,\ -,\ /,\ *, ^{\wedge},\ .)$ consecutively.
  - Contains balanced bracket sequence on both sides of the equation.
  - Contains exactly one $=$.
  - Does not contain $<\ or\ >$.
  - Does not contain = sign at terminal positions.
  - Contains at least one variable(latin letter) on at least one side of the equation.
  - Does not contain a term that divides by zero.

The *PositionOfTerms* class would look something like this:

```
class PositionOfTerms(BaseObject):
    """Class for position of terms."""

    description = 'The position of terms relative to the = sign in an equation.'
    default_value = 'both'

    SCHEMA = {
        'type': 'unicode',
        'choices': [
            'lhs', 'rhs', 'both', 'irrelavent'
        ]
    }
```

To support such input, we would add a new directive in `extensions/objects/templates` that converts the choices into human readable form as follows:

```
[{
    name: 'lhs',
    humanReadable: 'on LHS'
    }, {
    name: 'rhs',
    humanReadable: 'on RHS'
    }, {
    name: 'both',
    humanReadable: 'on both sides'
    }, {
    name: 'irrelevant',
    humanReadable: 'with reordering allowed around ='
}]
```

Similar to the other two interactions, the next step would be to add a new directory under `extensions/interactions/` which will contain the following files that would define the behavior of the interaction:

`/MathEquationInput.py`: This will be the configuration file for the interaction where attributes such as name, description, dependencies, customization args, etc. are to be added. This file will contain a MathEquationInput class that will inherit the BaseInteraction class which contains all definitions of the attributes.
The MathEquationInput class will look exactly like the AlgebraicExpressionInput class defined here. Also, the same attributes for this interaction as well would be added to `interaction_specs.json`.

`/directives/`: This directory will contain the files for

- *Three directives*: interaction, response, and short response
- *Two services*: rules and validation.

The three directives will each have an HTML template to define the view and the two services will each have their corresponding test files. All three HTML templates will be bound to their corresponding directives and with the validation service.

The validation service will ensure that the math equation entered by the creator while creating the exploration is valid and does not contradict any other rules that may have been previously added. Since this interaction also contains no customization args, it will only return a list of warnings specific to the rule-set provided by the creator.

Same as the Algebraic Expression Input Interaction, warnings to be displayed if and when the learner submits an invalid math equation are listed in the Product Design section. The exact warning message is mentioned here. The following is a brief description of how these validations will be implemented:
The following is a brief description of how these validations would be implemented:

- `containsInvalidVariables`: Nerdamer's `variables()` function will be used to retrieve all variables present in the learner's answer and this list will be compared with the list of variables present in the creator's answer. If the learner's answer contains variables not present in the creator's answer, this function will return true.
- `containsConsecutiveOperators`: A simple linear scan to check the consecutive indices of the answer string would suffice.
- `imbalancedBracketSequence`: Nerdamer will throw a `ParityError` while parsing the answer string if there is an imbalance in the bracket sequence.
- `isValidEquation`: This would return true iff the frequency of $=$ in the string is equal to one and the frequency of $< or >$ is zero.
- `isAnySideEmpty`: This would return true if the first or the last index of the string is $=$
- `containsNumbersOnly`: This would return true if the length of the `variables()` function's list for both sides of the equation is zero.
- `dividesByZero`: Split the expressions(LHS and RHS) into terms and evaluate each term's denominator.

Moving on to the rules service. The first step would be to add the rules description in `extensions/interactions/rule_templates.json`.
The exact description of the rules is mentioned in this table.

After describing the rules, they would be defined in the rules service. An overview of how the rules will be implemented is given below:

- `MatchesExactlyWith`, `ContainsSomeOf`, `OmitsSomeof`: Each of these rules will have a value for the position of terms. Based on the position, the equation string will be split into two sides (by $=$) and each of them will be an expression. Then these expressions can be compared between the learner's and creator's answers using the same algorithms that are used by the Algebraic Expression Input Interaction. The details of these algorithms are mentioned here.
  The first step would be common for all four argument values. The given equations would be split into two parts by = sign.

```
let learnerSplitString: Array<string> = answer.split("=");
let creatorSplitString: Array<string> = inputs.x.split("=");

let learnerLHS = nerdamer(learnerSplitString[0]);
let learnerRHS = nerdamer(learnerSplitString[1]);

let creatorLHS = nerdamer(creatorSplitString[0]);
let creatorRHS = nerdamer(creatorSplitString[1]);
```

Next, depending on the value of the position argument, the corresponding expression would be taken and the algorithm related to the rule selected would be applied as if it were being applied on an equation.

- **lhs**: `learnerLHS` would be compared with `creatorLHS`
- **rhs**: `learnerRHS` would be compared with `creatorRHS`
- **both**: `learnerLHS`, `learnerRHS` would be compared with `creatorLHS`, `creatorRHS` correspondingly.
- **irrelevant**: `learnerRHS` would be multiplied with -1 and added to `learnerLHS` and a single expression would be formed. The same operations would be performed to the creator's expression. Thereafter, both of these expressions will be compared directly and after multiplying one of them with -1. This is done so that $x + y = 3$ and $3 = x + y$ can be considered as equal. Since here the single expressions formed would be $x + y - 3$ and $3 - x - y$.

- `IsEquivalentTo`: This rule does not take any positional arguments and will by default consider the relative positions of the terms with respect to the = sign as irrelevant. For this reason, the procedure described in the “irrelevant” point mentioned above would be carried out first. Then the expression formed would be expanded with nerdamer’s expand function and then both expressions(learner and creator) would be compared with nerdamer’s eq function.
  This is essentially a collation of Algebraic Expression Input interaction’s matches loosely with rule and the “irrelevant” positional argument mentioned above.
  If the expressions don’t match with the normal .eq function even after expanding all terms, another check will be applied: Algebraically divide both expressions and if the resultant is a constant, return true. An example where this would be relevant is if the creator’s input equation is $13 + 2w = 34$ and the learner enters something like $w/10.5 = 1$. This wouldn’t get validated without the last division check.

- `MatchesWithGeneralForm`: This rule also does not take any positional arguments and will by default consider the value of the position argument as ‘both’. This implies that the general form match algorithm mentioned here in the Algebraic Expression Input interaction section would be applied on both sides of the equation individually and this rule would return true iff both sides get validated by the function.

Testing Approach

Each interaction addition will follow a similar procedure for adding tests. Each interaction will involve additions of two services, rules and validation. Both of them will have a corresponding spec file for unit testing. Oppia uses Karma for unit testing. These files will contain tests for each rule that the corresponding interaction supports. They will contain detailed test cases based on the examples table that is shown in the product design section of each interaction: AlgebraicExpressionInput, NumericExpressionInput, and MathEquationInput. These comprehensive test cases will ensure that each rule functions exactly the way it is supposed to.

Recently, an error was detected regarding the guppy editor wherein the learner could not type in the symbols when the network connection was disabled. Issues like this would lead to an annoying user experience especially for certain learners/creators that don't have access to stable internet. This is why thorough end-to-end testing is extremely crucial.
To ensure that the interactions behave in the way they are supposed to, end-to-end tests with protractor will be added in the `protractor.js` file present in the interactions directory under `extensions/interactions/`. The following functions/variables will be added to the protractor file for each interaction:

- `expectInteractionDetailsToMatch`: Function to check if the interaction is correctly displayed when an exploration that uses this interaction loads up.
- `submitAnswer`: Function to check if the submission of an answer works properly.
- `answerObjectType`: Variable to check if the type of object returned by the interaction is as expected. For all three interactions, we expect a *MathString* to be returned.
- `testSuite`: Variable to describe various scenarios in which the interaction could be used. It will be an array of objects. Each object will contain various attributes that describe a particular scenario. The attributes that each object will contain are:
    - `interactionArguments`
    - `ruleArguments`
    - `expectedInteractionDetails`
    - `wrongAnswers`
    - `correctAnswers`

  The tables containing various correct and incorrect answers for each rule could be used as a reference for creating such scenarios.

Migrating Explorations

The addition of the new interactions and deprecation of the old one will require a stable migration of the old explorations' schema so that they're compatible with the new ones'.

After the Algebraic Expression Input and Math Equation Input interactions are ready to be shipped into production, we can disable(make invisible to the end-user) the current math interaction to disallow any new explorations from using it.
To ensure that the old interaction isn't visible to the end-user, I would add a conditional in the `customize-interaction-modal.template.html` under `core/templates/pages/exploration-editor-page/editor-tab/templates/modal-templates/`. This would make sure that the both, interaction name and its thumbnail, aren't rendered and since this is just a frontend change, it wouldn't mess with any other references to this interaction. Any explorations that use this interaction would still function.

The important thing now would be stably migrating all the explorations that currently use the old math interaction to use the new interactions. This can be automatized with a one-off job.
I plan on writing a single job that will upgrade all explorations together. The job will be implemented as follows.
The important decision to be made while upgrading explorations is, what interaction a given exploration should be upgraded to. First thing to check would be, if the exploration's input contains an `=` sign. If it does, it will be upgraded to the MathEquationInput interaction. I've already written an audit job that can be used as reference for doing this. If not, we need to decide if the exploration should be upgraded to Algebraic or Numeric Expression Input interaction.
While playing through some of the math explorations, I've observed that most of the explorations that use purely numeric expressions have resorted to using the text input interaction since the current math interaction doesn't have any support for such inputs. This is why we can safely assume that all explorations that use the old math interaction will contain some algebra in their input. Also, Sean has looked at the explorations data and also confirms this approach.
That said, to be absolutely sure, while writing this one-off job, I will verify this assumption again by writing a simple audit job to check for the type of input.

`UpgradeMathExplorationsOneOffJob`

This job will check the schema of each exploration. The explorations that have the interaction id equal to `MathExpressionInput`, will have their schemas changed. The following changes will be done:

- `[interaction][answer_groups][rule_specs][inputs][x]`: This is the input provided by the creator. This is currently in LaTeX format and the new interactions require ASCIIMath. For this reason, the unicode string must be parsed with a LaTeX parser and then converted to ASCIIMath format. To carry out this operation, a third-party library,

pylatexenc, would have to be temporarily introduced in the codebase. This library provides a function latex_to_text which can be used to parse the latex string.
Note: This function would return text instead of ASCIIMath but that's totally fine since the reason for using ASCIIMath was to ease the nerdamer's parsing process which does not work directly on LaTeX. This is why, using text instead of ASCIIMath won't lead to any problems. Also, I've tried this out locally on various examples mentioned here, and nerdamer was able to parse all of them from the text format given by pylatexenc.
- `[interaction][answer_groups][rule_specs][rule_type]`: This is the rule type and has the value `IsMathematicallyEquivalentTo` for all explorations.
The creator's input will be checked; if the input is an expression, the rule_type will be changed to MatchesExactlyWith and if it is an equation, another argument, `PositionOfTerms`, will be added to the rule_type which will have the value of 'both'. This will change the inputs object for that answer_group.
- `[interaction][id]`: This has the value `MathExpressionInput` for all explorations and would have to be changed to the appropriate interaction id based on the value of the input string as mentioned above.

This one-off job will be merged by June 27 and will therefore be run during the release cut in July.

Milestones

Community Bonding Period (May 4 - May 31)

During this period, I will finalize all the technical details with my mentor. Also, since I have been contributing to this community for more than 3 months now, I'm quite familiar with the workflow in Oppia. This is why I will start working on the project right away, with the consent of my mentor, in order to cover-up for any unforeseen delays in the future.

Milestone 1 (June 1 - June 28)

Key Objectives

- The Algebraic Expression Input and Math Equation Input interactions will be ready for lesson creators to use with the following two rules:
    - MatchesExactlyWith
    - IsEquivalentTo.
- The old Math Expression Input interaction will be disabled (will not be visible in the Choose Interaction modal) in order to disallow any new explorations from using it.
- A migration job to upgrade all the explorations that use the old math interaction will be merged for the July release, and will have passed successfully without errors on a test run on production data.

No.	Description of PR	Prereq PR #	Target date for PR submission	Target date for PR to be merged
1.1	The nerdamer third-party library will be ready to use in the codebase.	None	June 1	June 4
1.2	AlgebraicExpression, MathEquation, PositionOfTerms and SetOfPlaceholders object classes will have been added to extensions/objects/models/objects.py with their validations in schema_utils.py. A math-editor.directive with guppy and the position-of-terms.directive with HTML templates will have been added to extensions/objects/templates.	None	June 2	June 5
1.3	The algebraic interaction configurations will have been added to interaction_specs.json and to the configuration file in the new interaction directory under extensions/interactions/ AlgebraicExpressionInput/ AlgebraicExpressionInput.py. The following directives will have been added under extensions/interactions/directives/ ● oppia-interactive-algebraic-expression-input.directive.ts ● oppia-response-algebraic-expression-input.directive.ts ● oppia-short-response-algebraic-expression-input.directive.ts along with their respective HTML templates.	1.1, 1.2	June 6	June 10
1.4	The following rules will have been implemented in algebraic-expression-input-rules.service.ts under extensions/interactions/ AlgebraicExpressionInput/directives/ along with the relevant unit tests: ● MatchesExactlyWith ● IsEquivalentTo The validation service, algebraic-expression-input-validation .service.ts, will have been added under extensions/interactions/ AlgebraicExpressionInput/directives/ along with its unit tests.	1.1, 1.2	June 8	June 12

1.5	The end-to-end tests for this interaction will have been implemented in protractor.js under extensions/interactions/AlgebraicExpressionInput/.	1.4	June 13	June 17
1.6	The math equation interaction configurations will have been added to interaction_specs.json and to the configuration file in the new interaction directory under extensions/interactions/MathEquationInput/MathEquationInput.py. The following directives will have been added under extensions/interactions/directives/ ● oppia-interactive-math-equation-input.directive.ts ● oppia-response-math-equation-input.directive.ts ● oppia-short-response-math-equation-input.directive.ts along with their respective HTML templates.	1.1, 1.2	June 14	June 18
1.7	The following rules will have been implemented in math-equation-input-rules.service.ts under extensions/interactions/MathEquationInput/directives/ along with the relevant unit tests: ● MatchesExactlyWith ● IsEquivalentTo The validation service, math-equation-input-validation.service.ts, will have been added under extensions/interactions/MathEquationInput/directives/ along with its unit tests.	1.1, 1.2	June 15	June 19
1.8	The end-to-end tests for this interaction will have been implemented in protractor.js under extensions/interactions/MathEquationInput/.	1.7	June 20	June 23
1.9	The old Math Expression Input interaction will be unavailable in the *Choose Interaction* modal. A migration job to upgrade explorations that use the old math interaction will be merged for June's release.	1.5, 1.8	June 24	June 27

Milestone 2 (July 4 - July 26)

Key Objectives

- The Numeric Expression Input interaction will be ready to use with all its rules implemented.
- No exploration on the production server uses the old MathExpressionInput interaction.

No.	Description of PR	Prereq PR #	Target date for PR submission	Target date for PR to be merged
2.1	The NumericExpression object class will have been added to extensions/objects/models/objects.py with the validations in schema_utils.py.	None	July 5	July 8
2.2	The interaction configurations will have been added to interaction_specs.json and to the configuration file in the new interaction directory under extensions/interactions/NumericExpressionInput/NumericExpressionInput.py. The following directives will have been added under extensions/interactions/directives/ ● oppia-interactive-numeric-expression-input.directive.ts ● oppia-response-numeric-expression-input.directive.ts ● oppia-short-response-numeric-expression-input.directive.ts along with their respective HTML templates.	2.1	July 12	July 15
2.3	The following rules will have been implemented in numeric-expression-input-rules.service.ts under extensions/interactions/NumericExpressionInput/directives/ along with the relevant unit tests: ● MatchesExactlyWith ● IsEquivalentTo ● ContainsSomeOf ● OmitsSomeOf The validation service, numeric-expression-input-validation.service.ts, will have been added under	2.1	July 17	July 21

	extensions/interactions/NumericExpressionInput/directives/ along with its unit tests.			
2.4	The end-to-end tests for this interaction will have been implemented in protractor.js under extensions/interactions/NumericExpressionInput/.	2.3	July 21	July 25

Milestone 3 (Aug 1 - Aug 30)

Key Objectives

- The following rules will be added to the Algebraic Expression Input and Math Equation Input interaction:
    - ContainsSomeOf
    - OmitsSomeOf
    - MatchesWithGeneralForm
- The old Math Expression Input interaction will be stably deprecated and all related code will be removed from the codebase.

No.	Description of PR	Prereq PR #	Target date for PR submission	Target date for PR to be merged
3.1	The following rules will have been added to algebraic-expression-input-rules.service.ts under extensions/interactions/AlgebraicExpressionInput/directives/ along with unit tests: ● ContainsSomeOf ● OmitsSomeOf ● MatchesWithGeneralForm Along with this, the validation service will have been updated as well.	1.1, 1.2	Aug 5	Aug 10
3.2	The following rules will have been added to math-equation-input-rules.service.ts under extensions/interactions/MathEquationInput/directives/ along with unit tests: ● ContainsSomeOf ● OmitsSomeOf ● MatchesWithGeneralForm Along with this, the validation service will have	1.1, 1.2	Aug 13	Aug 17

	been updated as well.			
3.1	The following references to the old interaction will have been removed from the codebase: ● This, and any other audit jobs that reference the old interaction. ● core/templates/domain/exploration/SolutionObjectFactory.ts and its unit test. ● core/templates/pages/exploration-player-page/services/exploration-player-state.service.ts ● core/tests/protractor_desktop/extensions.js ● core/tests/protractor_utils/ExplorationEditorMainTab.js ● Interaction_specs.json ● extensions/interactions/interactionsRequires.ts ● extensions/interactions/protractor.js ● extensions/interactions/rule_templates.json ● extensions/interactions/rules.spec.ts ● scripts/check_frontend_coverage.py ● extensions/interactions/MathExpressionInput/ (delete this directory)	1.9	Aug 18	Aug 21
3.2	An audit job to surface all explorations that use the new interaction will be ready to run.	3.1	Aug 19	Aug 21

After the audit job is run successfully and it yields all the explorations, I will manually playtest them in order to verify that everything works as expected. I will be able to get this done by August 25.
I have left the end of August empty in order to leave some extra time for any unforeseen bugs that may be found in the newly added interactions.

If all bugs have been fixed and everything is stable, I'd really like to start taking up one of the future works. I will start by authoring new explorations that leverage all of the functionality that these new interactions provide.

I aim to add new collections for:

- Linear equations (Identifying and solving them)
- Handling polynomials (Expanding, factoring and finding roots)
- Perimeter and area (There is an exploration related to this but I think creating an entire collection that would include formulating perimeter and area of various shapes would be quite beneficial)

I also aim to add the following interactions:

- Prime factorization
- Exponents and powers
- Place values
- Formulating algebraic expressions

This is by no means an exhaustive list. If any other topics are required, I will add them as well. The primary goal is to explore the boundaries of these new interactions and see how versatile they are.

Optional Sections

Future Work

I've been closely working on this topic for more than a month now. This idea of revamping math interactions originated from this issue which pointed out some gaping holes in the current interaction. I'm committed to the idea of providing scalable and versatile functionality to create not just math, but any explorations that require expressions/equations (physics, economics, etc). This is why, even after the GSoC period, I would be extremely keen on expanding upon these interactions since the possibilities of enhancements are endless.

Some of the enhancements/additions that I've envisioned for these interactions include:

- Authoring new explorations/collections that leverage the features that these newly added interactions will provide.
- Changing the RTE math editor to use guppy for an intuitive experience instead of forcing the learner/creator to enter a LaTeX string.
- Adding a customization argument that would support swappable variables. For instance, if the correct answer is πr^2 and the learner enters πx^2 , it should be validated as correct.
- Add rules to support inputs on more advanced topics such as trigonometry, calculus, matrices, etc. Additions such as these would be made easier by the addition of nerdamer since it supports a wide range of advanced functions and has an active community.

With the current additions acting as a solid basis, and these additional functionalities, I hope to make the math experience of the learners and creators as intuitive, smooth and fruitful as possible.

Additional Project-Specific Considerations

Security

This project introduces a new math library called nerdamer. This performs operations on mathematical expressions after parsing them, but all these operations are done purely in JavaScript and no server side programming is done, so there should be no security issues related to this library.

Apart from this, these interactions will use the guppy math editor. Now there is some possibility of cross-site scripting while using these kinds of customizable inputs. But, in anticipation of such attacks, the developers of guppy guarantee that all XML documents handled by the editor will be parsed by DOMParser only and will never be added to the root document. This would prevent any cross-site scripting attacks.

Accessibility (if user-facing)

These new interactions will provide an intuitive way of typing in math expressions/equations with the WYSIWYG math editor guppy. This assumes no prior knowledge of LaTeX from the learner or the creator. The editor will also provide comprehensive instructions on how to enter symbols. This would make the process easier for younger learners.

Documentation Changes

This project aims to add two new interactions and will therefore need no additions to the Oppia wiki page.