The University of Edinburgh

Loughborough University

The Open University

Getting started with STACK

AdvanceHE

COLLABORATIVE AWARD FOR TEACHING EXCELLENCE

This guide was written by

Tim Lowe, The Open University
Chris Sangwin, The University of Edinburgh
Ian Jones, Loughborough University

with contributions from

Grahame Erskine (The Open University), Maltheperiod Sporring (The University of Edinburgh)

as part of a Higher Education Academy (now Advance HE) Collaborative Award for Teaching Excellence (CATE).

The CATE was made possible by funding from HEFCE, HEFCW, and DfE and supported by the Higher Education Academy (now Advance HE)

Version 0.3, July 2019
Comments and suggestions on this document are welcomed: please email `tim.lowe@open.ac.uk`

Contents

Introduction

This guide is intended to explain how to use STACK to produce online interactive assessment questions, and to provide a reference for commonly used commands.

1 What is STACK?

STACK (System for Teaching and Assessment using a Computer algebra Kernel) is a powerful system for the online assessment of mathematics and related subjects. It enables the generation of randomised questions, permits entry of algebraic expressions as answers and supports immediate feedback to students based on the mathematical properties of their answers.

STACK is available as a question type for the Moodle Virtual Learning Environment (VLE) and also as a question type for the ILIAS system. It can be used in other systems via the LTI (Learning Tools Interoperability) protocol. An application programming interface (API) is being developed to enable STACK to be easily embedded into other software. For the purposes of this guide, we consider the use of STACK within Moodle.

STACK is a highly sophisticated system. With such a powerful system, there is inevitably some complexity in authoring robust, useful questions. Question authoring requires skills in a variety of areas including:

- teaching mathematics, to identify appropriate questions, common errors and appropriate feedback;
- mathematics, to create and appropriately randomise questions;
- programming using the Maxima computer-algebra system, to implement the question randomisation;
- simple LaTeX, for the display of mathematics;
- the Moodle VLE, for storing the questions and assembling the quiz.

The complete documentation of the STACK system, including a Quick Start guide, is part of every STACK installation. It can be accessed on your system as

```
[moodle-url]/question/type/stack/doc/doc.php
```

where `[moodle-url]` is the URL of your Moodle server. This provides documentation corresponding to the version of the system installed. The current version of the STACK documentation can also be read at `https://github.com/maths/moodle-qtype_stack/blob/master/doc/en/index.md` (Note that this may describe features not yet available in your local STACK installation.)

Tip

When authoring STACK questions, it is recommended that the Firefox or Chrome web-browsers are used, rather than Internet Explorer which can cause some difficulties.

Tip

Internally, STACK has “version numbers” which are specified at `https://github.com/maths/moodle-qtype_stack/blob/master/version.php` . The version number is stored within the variable `$plugin->version`. It is used within the software, and is saved within any question written. This can help when maintaining and upgrading the system.

Question authors do not need to worry about this in general use.

2 Question behaviours

Before beginning to create a STACK question, it is useful to have a little understanding of the different ways in which Moodle quizzes can be used, since this can influence how questions are written. These different ways are known as **question behaviours**. Question behaviours are set at the quiz level, rather than the question level, hence ensuring that all questions in the quiz behave in the same way.

The different question behaviours available include the following.

Interactive with multiple tries Students can have multiple attempts at each question within a quiz (with a hint being given after each attempt) until some limit on the number of permitted attempts is reached. Feedback is given immediately after each attempt. This behaviour is most useful in formative settings and STACK was designed primarily with this behaviour in mind.

Adaptive Students can have unlimited attempts at a question before moving on to the next question.

Immediate feedback Students have only one attempt at answering each question, with immediate feedback being given as soon as each question is answered.

Deferred feedback Students have one attempt at answering each question and no feedback is given until after the quiz closing date. However, students will normally get “validation” feedback about syntax errors and other input validation problems as they enter their answers. This behaviour is most useful for summative settings such as exams.

Immediate feedback with CBM (confidence based marking) As *Immediate feedback*, but students also have to indicate how confident they are in their answer. Marks are given based on the correctness of the answer and the level of confidence given.

Deferred feedback with CBM As *Deferred feedback*, with the addition of the confidence indicator described above.

3 Moodle question bank

STACK questions (along with all other types of Moodle quiz question) are stored within the Moodle **question bank**. There is a question bank associated with each individual Moodle course, and a *separate* bank associated with each individual quiz.

Tip

We recommend storing questions within the *course* question bank, unless there is good reason to do otherwise.

Each question bank can be divided into a number of **categories** which function similar to folders or directories in a computer file system. Categories can themselves be divided into sub-categories.

In addition to helping organise large collections of questions, one advantage of categories is that you can ask Moodle to select a random question from a specified category to be included within a quiz. So questions can be randomly selected from within a category in addition to including randomisation themselves.

Questions can also be allocated *tags* (see Subsection 5.7). The question bank can be searched by tag, and questions can also be randomly selected to be included in a quiz on the basis of their tags.

Tips

- Bear in mind how questions will be used within quizzes when deciding on a category structure.
- Remember that questions can be randomly selected from a category or by tag – an individual question does not have to contain every possible randomisation internally.

The Moodle course question bank can be accessed from the *Website administration* section of the **Administration** block on the course home page. (Quiz question banks are accessed from the *Quiz administration* area of that block, which is displayed whenever a quiz is being accessed.)

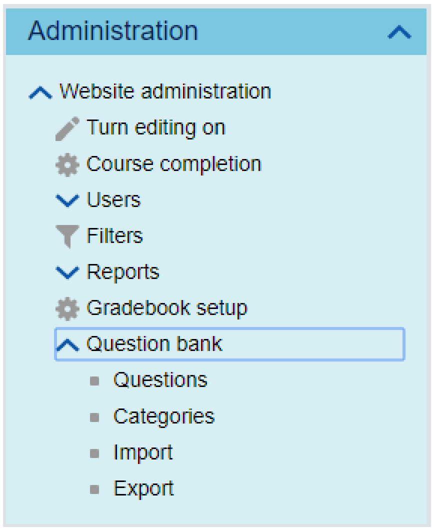
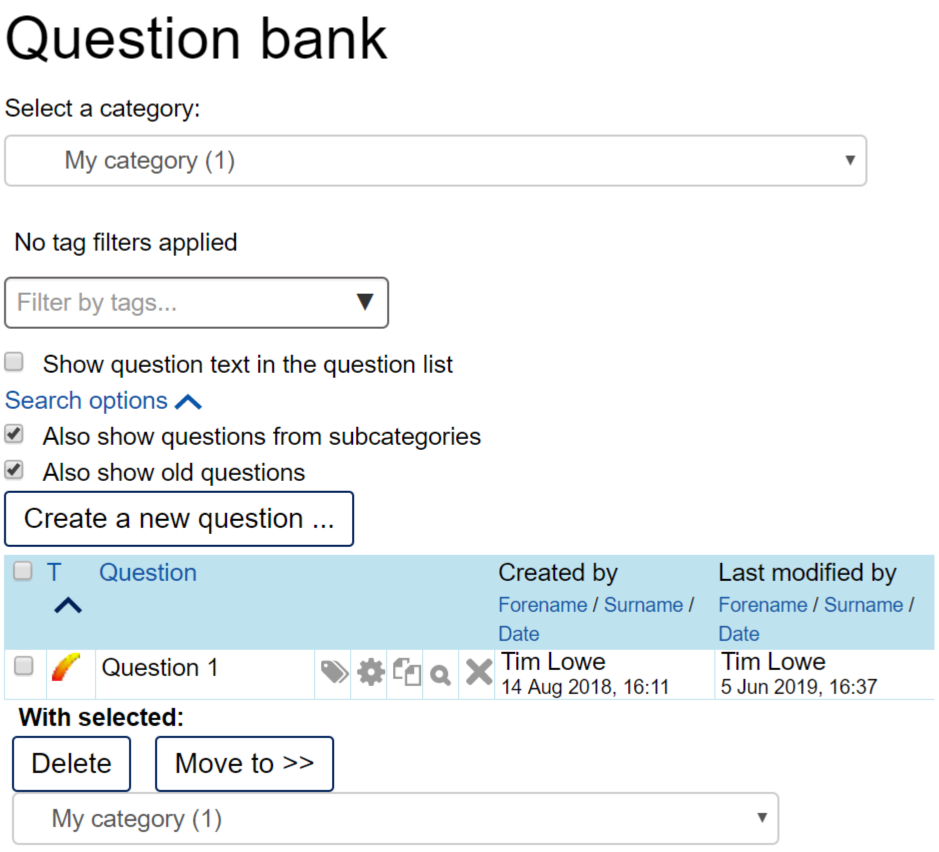


Figure 1 The Moodle course administration block

There are two views of the question bank:

Questions which shows the questions within a given category, as shown in Figure 2;

Categories which lists the categories, and allows categories to be added, removed or re-organised.


Question bank

Select a category: My category (1)

No tag filters applied

Filter by tags...

☐ Show question text in the question list

Search options

☑ Also show questions from subcategories

☑ Also show old questions

Create a new question ...

	T	Question		Created by Forename / Surname / Date	Last modified by Forename / Surname / Date
☐		Question 1		Tim Lowe 14 Aug 2018, 16:11	Tim Lowe 5 Jun 2019, 16:37

With selected:

Delete Move to >>

My category (1)

Figure 2 The questions view of the Moodle question bank

4 Creating a STACK question

There are a variety of means by which STACK questions can be created. They can be written from scratch, created by editing a copy of an existing question or imported from another resource.

4.1 Authoring a question

To author a STACK question, first open the questions view of the question bank (Figure 2), then select the category within which you wish to create the question using the drop-down menu at the top of the form and click Create a new question . . . .

From the list of possible question types offered, select `STACK`.

Tip

Once you have created one question of a particular genre, it is often easier to create the next by duplicating the first and editing it.

You can duplicate a question by clicking next to the question name in the question bank.

You will be presented with a web-form to create your question. The different parts of this form are considered in Section 5.

4.2 Importing a question

STACK questions can be exported from one Moodle course (or question repository) and imported to a new course. Questions should be exported and imported using the `Moodle XML` file format.

To export all the questions within a given category, select *Export* from within the *Question bank* section of the Moodle *Administration* block (see Figure 1) and on the form that is then shown, select the appropriate file format (`Moodle XML`) and the category to be exported.

Individual questions can be exported from the **Preview question** page (see Section 6) which is opened by clicking next to the question name in the question bank. On that page, click the link to `Download this question in Moodle XML format`.

Alternatively, individual questions can also be exported using the `Export this question` link in the STACK **Question tests & deployed versions** page. (There are more details about this in Section 7.) The page can be accessed using the `Question tests & deployed versions` link either below the *Question name* on the question editing form (Subsection 5.1.1) or at the top right-hand corner of the *Preview question* page .

Questions are imported to a Moodle course using the *Import* link from within the *Question bank* section of the Moodle *Administration* block. In the form that is shown, again select `Moodle XML` as the file format and then upload the file containing the previously exported question(s).

5 STACK question form

Here we explain the details of the web-page used to author STACK questions. We will illustrate the process using an example question, one random instance of which might be

Example

Expand the brackets in the expression

$$(x + 2y)^2.$$

Tips

- When creating a question, start with the worked solution to the problem. If different random versions need different worked solutions, then consider having two or more different questions. If these different questions are put in the same question bank category, or given the same tag, then the quiz can be configured to randomly select one of them to be offered to the student.
- Similarly, questions do not have to include all randomisation possibilities. You can have different questions which are randomly chosen by the quiz.

5.1 General settings

This first *General* section of the STACK authoring form contains the question text, question randomisation and general feedback given to the students. By default, this section is expanded when the STACK authoring form first opens. It contains the following elements.

5.1.1 Question name

This is the name of the question that will appear in the question bank listing of questions. The question name box must be completed.

Tip

Choose a meaningful name, bearing in mind the end of a long name may not be visible in the question bank listing of questions.

5.1.2 Question variables

Within this box, Maxima code is used to randomise the parameters of the question and define variables whose value can then be used within the question statement and any worked solution. Question variables are optional.

Maxima is a computer-algebra system, so it can process algebraic expressions and equations etc. Any otherwise undefined variables are treated as mathematical variables.

Tips

- Use Maxima variable names consisting of more than one character. Students cannot enter any variable name defined in *Question variables* that has more than one character within their answer. Hence, if multi-character variable names are used, students cannot accidentally enter the name of the variable in which the answer is stored!
- Assign each mathematical expression you wish to use in the question or worked solution to a Maxima variable. Future maintenance of questions is easier if Maxima code is restricted to this area, rather than also including code in the question statement or feedback areas. In particular, assign the correct answer (or a sample correct answer) to the question to a variable.

Most Maxima commands can be used in STACK (some are prohibited for reasons of security). A list of useful Maxima commands is given in Appendix A, and a summary of the most essential is given in the following box. A more complete introduction to Maxima can be found at

`http://maxima.sourceforge.net/docs/tutorial/en/minimal-maxima.pdf`

Maxima syntax

- Variables and function names can be a combination of letters followed by a combination of numbers. (This is a STACK restriction on the usual Maxima convention.)
- Lines of Maxima code should end with a semi-colon (`;`).
- Values are assigned to variables using a colon (`:`), for example `aa:4;`
- Functions are defined using colon-equals (`:=`), for example `f(x):=x^2;`
- Equals (`=`) is used within equations, which could themselves be assigned to a variable, for example `eqn:x^2-4=0;`
- More complicated functions requiring several commands (similar to subroutines) can be defined using the following syntax

```
f(x,y):=block( [list of local variables],
            command, command,
            return(value to be returned) );
```

- Comments can be included in the code using `/*` *comment text* `*/`
- Lists are a powerful data structure and are delineated by `[...]`, for example `[1,2,3]`

In addition, STACK adds some additional commands, which in particular are very helpful when randomising problems.

STACK randomisation commands

- `rand(`*list*`)` selects a random element from the given list, for example, `rand([1,2,3])`.
- `rand_with_prohib(`*start, end, exclude*`)`, generates a random integer between *start* and *end* but excluding any of the elements of the list *exclude*. For example `rand_with_prohib(-10,10,[-1,0,1])`.

For the full complement of STACK commands see the documentation on your version of STACK.

Example

For our example problem, we might decide that the general form of the question to be asked is

> Expand the brackets in the expression
>
> $(x + ny)^2$

where n is a random integer between -9 and 9, but excluding $-1, 0, 1$.

This can be achieved using the following code in the *Question variables* box.

```
ordergreat(x);

nn: rand_with_prohib(-9,9,[-1,0,1]);
term: (x+nn*y);
epn: term^2;
ans: expand(epn);

simp:false;
work: x*x+x*nn*y+x*nn*y+y*y;
```

Here, the expression to be expanded is stored in the variable `epn`, and Maxima used to calculate the answer which is stored in `ans`.

The command `ordergreat(x)` changes the default ordering of terms in Maxima, so that the term in x is shown before that in y.

`simp:false` turns off the automatic Maxima simplification of the expression, so that the next line, which defines a variable needed for the worked solution, is not simplified.

You can experiment with developing Maxima code for STACK offline using a local desktop version of Maxima (for example wxMaxima). To have a local version of Maxima work in exactly the same way as STACK, with the all additional commands etc, you can download the STACK sandbox, as described in the STACK documentation: `[moodle-url]/question/type/stack/doc/doc.php/CAS/STACK-Maxima_sandbox.md`

5.1.3 Random group

This is used to link two (or more) separate STACK questions so they have the same values of randomised variables. It can be useful in creating a sequence of linked questions each addressing different steps within the same instance of a larger problem.

Every question with the same string in this box has the same seed for the random number generator, and hence if the *Question variables* code is the same in both questions, then both questions will have variables with the same random values.

Example

For our simple example, leave this box empty.

5.1.4 Question text

This is the text of the question that will be displayed to students.

Mathematics can be included using LaTeX syntax (see Appendix C). Use the LaTeX delineator `\( ...\)` for inline maths and `\[ ...\]` for displayed maths.

The values of any Maxima variables defined in the *Question variables* section can also be included here. Using `{@...@}` displays the value of the variable correctly formatted using LaTeX. For, example, the correctly formatted value of the variable `epn` can be included using `{@epn@}`. This can be included both within and outside of a LaTeX environment. In both cases, the output will be formatted using LaTeX.

Tips

- It is better to assign randomised mathematical expressions to a question variable rather than attempt complicated LaTeX constructions in the question text.
- It is better not to use Internet Explorer for authoring STACK questions as its default behaviour is to insert an HTML link to an email address whenever `@` is entered.

It is also possible to insert the unformatted *value* of the variable using `{#...#}`. This will use Maxima's display representation rather than using LaTeX. For example, if `p:sin(n*pi)` then `{#p#}` will be replaced by `sin(n*%pi)` and `{@p@}` will be replaced by `\(\sin(n\cdot\pi)\)`.

A text box for students to enter their answers is included using, for example

```
[[input:ans1]]
```

This will cause the student's input to be assigned to the variable `ans1`.

When using STACK, students' inputted answers are first subject to a validation check to ensure the answer is appropriate (as defined by the settings discussed in Subsection 5.4). Invalid answers are usually highlighted to the student who can then address the problem and revise their answer before it is submitted for marking.

Separating validation from "correctness" is a key design feature of STACK, and helps students significantly. The validation feedback can be shown even when using the *Deferred feedback* question behaviour.

Each input box needs to have a corresponding `[[validation:ans1]]` tag indicating the position in which the input validation feedback will be displayed.

Tip

The question text box is pre-populated with

```
[[input:ans1]] [[validation:ans1]]
```

when the question is created.

Example

For our example, ensure the *Question Text* box contains the following.

```
Expand the brackets in the expression
\[ {@epn@}.\]

\( {@epn@} = \) [[input:ans1]]
[[validation:ans1]]
```

Note that repeating the expression before the answer box provides a hint to the students they should not enter both the question and the solution, linked with equals signs as their answer.

STACK questions can have more than one input box for students to type their answers into. To include a second box, include additional `input` and `validation` tags, for example `[[input:ans2]] [[validation:ans2]]`, in the question text. This example will cause the answer input in the second box to be stored in the variable `ans2`.

After completing these first few elements in the *General* section of the form, it is useful to test the question is working before continuing.

5.2 Saving and testing

STACK does not allow the question to be saved until certain parts of the form have been completed.

You might like to continually save and test your question as it is developed.

5.2.1 Saving

To enable the question to be saved, temporarily complete the following elements of the form.

- In the *Question note* box (the last box within the *General* section of the form) enter some text that the system will use to distinguish between different random versions of the question.

  Example

  For our example, enter

  ```
  \( {@epn@}={@ans@} \)
  ```

  This will be considered further in Subsection 5.3.6.

- Ensure the section of the form corresponding to the variable in which the student's answer is stored (titled *Input:ans1* by default) is expanded by clicking the down arrow to the left of the section name, and in the box labelled *Model answer* enter the name of the variable which holds a correct answer to the question. This is considered in more detail in Subsection 5.4.2.

  Example

  In our example, this is `ans`

  (If you have included more than one answer box, hence have more than one student answer variable, you will need to do this in each section relating to an input answer variable).

- Expand the section of the form labelled *Potential response tree: prt1*. In the part labelled *Node 1* (with the coloured background) enter 0 in the boxes for both *Sans* (the student's answer) and *Tans* (the teacher's answer). This is where the marking of the answer is done, and here, temporarily, we will be marking the answer as correct if 0 (*Sans*) equals 0 (*Tans*), that is, any valid answer will temporarily always be marked as correct. This will be reconsidered and the answer marked properly in Subsection 5.5.

Now scroll to the bottom of the form and click the [Save changes and continue editing] button. This should save your work, and keep the STACK form open for more editing. The alternative [Save changes] button at the bottom of the page saves your question and returns you to the question bank.

If there is an error in your question, the question will not be saved and red error text will be shown at the relevant point in the question form. Correct the error and re-save.

5.2.2 Preview and Testing

You can preview and test the question by either

- in the question form, click `Preview` which should be shown next to the $\boxed{\text{Save changes and continue editing}}$ button once the question has been successfully saved; or
- from the question bank, click the preview icon near the question name.

This opens a *Preview question* page which displays the question. Check that this is showing a correctly randomised version of your question.

If the Maxima code fails, for reasons other than syntax errors, the question preview may display the text

```
CAS Error:  No expressions were returned.
```

The following subsection may help you resolve any issues.

5.2.3 Debugging

One way of identifying the location of errors in the *Question variables* code is to comment out sections (by surrounding them with `/*` ... `*/`), and re-saving the question. You might also wish to temporarily edit the *Question text* to display the values of appropriate variables. The code can then be uncommented in stages until the error is found.

If the question text shows the name of a variable rather than its value, this means that the variable has not been assigned a value. This might be due to a typographical error in either the *Question variables* or the *Question text*.

Another useful aid is an additional STACK page that displays the values (for one particular question instance) of all variables defined in the question. To access this, click `Question tests and deployed variants` at the top right-hand corner of the *Preview question* page. Scroll down until you find the section called *Question variable values* which shows the variable values for one particular randomised instance.

If changes are made to the question code, both the *Preview question* and *Question tests and deployed variants* pages can be updated to reflect the changes using your browser refresh function (usually `F5`). The *Question tests and deployed variants* page will be considered in more detail in Section 7.

5.3 General settings, continued

The remaining elements of the *General* section of the STACK editing form are as follows.

5.3.1 Default mark

This is the mark to be given if the answer to the question is fully correct. Since the mark can be scaled when the question is incorporated into a quiz, it is usual to set this to be 1.

Example

For the example question, set this to 1.

5.3.2 Specific feedback

This is the feedback given to a student in response to their particular given answer. This field is common to all Moodle quiz questions, and whether this feedback is shown to the student or not is set at the Moodle quiz level.

In STACK, this feedback is normally generated via the Potential response tree marking algorithm, described in Subsection 5.5.

To display the output from the potential response tree (which has the name `prt1` by default), include the tag

```
[[feedback:prt1]]
```

in the specific feedback. This is included by default when the STACK question is created, and usually no other text need be included here.

In multi-part questions, with many inputs and many potential response trees, it is often more sensible to put specific feedback directly next to the inputs. Hence these tags can be moved into the *Question text*. However, in moving the tags you lose some control in how the quiz can be configured. You cannot control both the timing and the position of the feedback (which is one unfortunate constraint of working within Moodle).

Tips

- If you want specific feedback to be shown right next to the input, then move the tag.
- If you want control over the timing of when specific feedback is given, using the quiz settings, leave the tag in this box.

Example

For this example, leave the `[[feedback:prt1]]` tag in the *Specific feedback* box. Nothing else need be added to this box.

5.3.3 Penalty

When the question is used with the *Interactive with multiple attempts* question behaviour, this specifies the (absolute) number of marks to be lost for each incorrect attempt.

Tip

One common use is to give students three attempts at the question, in which case the penalty should be set to 0.333. STACK takes any penalty between 0.33 and 0.34 to be equal to $\frac{1}{3}$, so that $3 \times 0.333 = 1$, and three failed attempts loses all marks, assuming the *Default mark* is set to 1.

Note

The actual number of attempts the student is permitted is determined by the number of *Hints* specified, as described in Subsection 5.6.10.

5.3.4 General feedback

This is the text given to the student once they have used all their permitted attempts at the question, or once they have given a correct answer. It is typically used to give a worked solution to the particular randomised question instance asked.

This section behaves similarly to the *Question text* section: LaTeX formatting can be included as well as values of any question variables.

Tip

You may need to define more question variables than are needed to specify the question in order to give a sufficiently detailed worked solution.

Example

For the example question, enter the following worked solution

```
\[
\begin{align*}
{@epn@} & = ({@term@}) \times ({@term@})\\
         & = {@work@}\\
         & = {@ans@}
\end{align*}
\]
```

Tip

After completing the general feedback, you may wish to save the question and preview it to check the feedback is displaying as expected.

5.3.5 ID number

This is a unique identifier which can be given to a question. It is used to identify the question when using it outside of a quiz, for example when embedded within other areas of Moodle.

Example

This can be left blank.

5.3.6 Question note

As mentioned in Subsection 5.2.1, the **Question note** is text unique to a particular randomisation of a question, and is used to distinguish between different randomised question instances. Two instances of a question are considered different if their *Question notes* are different. The *Question notes* are visible in the listing of deployed question variants, see Subsection 7.2.

Tip

It is often useful to include a summary of the randomised question and its answer in the question note.

Example

For the example question, if this box was not completed earlier, enter the following for the *Question note*.

```
\( {@epn@}={@ans@} \)
```

5.4 Input: ans1

This section of the form specifies the properties of the student input, which by default is stored in the variable name `ans1`. (Throughout this section, it is assumed that `ans1` has been used to store the student input.)

If you have changed the name of the answer variable or included more than one answer box, click the

Verify the question text and update the form

button at the end of the *General* section to update the rest of the form before continuing.

If you have more than one answer box, hence more than one answer variable, you will need to complete one of these sections for each answer variable. These additional sections will be added to the form when the

Verify the question text and update the form

button is clicked, or when you attempt to save the question.

Note that STACK separates the processes of ensuring a student's input is a *valid* answer to the question from that of deciding if the answer is *correct*. Some of the settings in this section help specify what a valid answer may be, and whether students are informed of an invalid answer before it is marked for correctness. Such a warning (if enabled) is displayed in the area specified by the location of the `[[validation:ans1]]` tag in the *Question text*.

5.4.1 Input type

This specifies the type of input associated with this variable. For example, whether it is a box allowing students to enter algebraic or numerical expressions, or a set of predefined options the student can select from. The available *Input types* are given in Appendix D.

Example

For our example, we require students to enter an algebraic expression, so select `Algebraic input`.

5.4.2 Model answer

This is the (or a) correct answer to the question. Generally, this will be stored within a variable defined in the *Question variables* section, so give the name of that variable here.

Example

If this field was not completed earlier, enter `ans` here.

5.4.3 Input box size

This is the length (in characters) of the input box displayed.

Example

The default of 15 is probably appropriate for this example.

5.4.4 Strict syntax

This specifies whether STACK should expect students to use strict Maxima input syntax when entering an answer or not. It affects the range of cases in which stars (for multiplication) might be automatically inserted, if enabled. See Subsection 5.4.5 following.

5.4.5 Insert stars

This setting works in conjunction with that above, and is used to determine how, if at all, the system automatically inserts multiplication stars into a student's answer where implicit multiplication might have been intended.

There are several options.

Don't insert stars

Insert stars for implied multiplication only This allows `2x` to be interpreted as $2 \times x$. It also means that `sin x` and `sinx` are invalid.

Insert stars assuming single character variable names Extends the above to consider `xy` to mean $x \times y$ and not a single variable with a two-letter name. A consequence is that `sin x` is invalid due to the space, but `sinx` is interpreted as $s \times i \times n \times x$.

Insert stars for spaces only This considers `xy` as a two-letter variable name but `x y` as $x \times y$. Note that `sinx` and `sin x` are considered invalid.

Insert stars for implied multiplication and spaces Here, `sinx` and `sin x` are invalid.

Insert stars assuming single character variables, implied and space This considers both `xy` and `x y` as $x \times y$. `sin x` is invalid due to the space, but `sinx` is interpreted as $s \times i \times n \times x$

To interaction between the *Strict syntax* and *Insert stars* settings is illustrated in the table below, for a range of possible inputs.

	Strict syntax	
	No	Yes
Don't insert stars	`2x` invalid `xy` $= xy$, a single variable `f(x)` invalid `2e3` invalid	`2x` invalid `xy` $= xy$, a singe variable `f(x)` $= f(x)$, a function `2e3` $= 2000.0$
Insert stars assuming single character variable names	`2x` $= 2 \times x$ `xy` $= x \times y$ `f(x)` $= f \times x$ `2e3` $= 6e \approx 16.310$	`2x` $= 2 \times x$ `xy` $= x \times y$ `f(x)` $= f(x)$, a function `2e3` $= 2000.$

Example

In this example, since no functions are present but terms such as xy will arise, set

- *Strict syntax* to `No`,
- *Insert stars* to `Insert stars assuming single character variable names`.

5.4.6 Syntax hint

This is text which will be shown within a answer input box, before the student starts typing. It can be used to give a hint such as "Type your answer here", or a partly completed expression with ? as a placeholder to assist students in using the correct syntax.

Example

This can be left blank.

5.4.7 Hint attribute

This indicates the nature of the *Syntax hint* given above. The options are

Value in which case students can edit the syntax hint when giving their answer;

Placeholder when the syntax hint is deleted as soon as the student begins to type within the answer-box.

> **Example**
>
> Since the *Hint Syntax* has been left blank, this setting becomes irrelevant. Leave the default setting.

5.4.8 Forbidden words

This is a comma-separated list of words (text strings) that are not permitted in a student's answer. Any answer containing these will be deemed invalid and the student will be asked to revise their answer. Remember, any multi-character variable names defined within *Question variables* are automatically forbidden, unless *Insert stars assuming single character variable names* is enabled, in which case such an expression is interpreted as a product of single-character variable names.

> **Tip**
>
> You can use forbidden words to prevent students from entering specified Maxima commands in their answer. For example, in a question on expanding brackets, you might like to prevent the use of the command `expand` !

The following keywords which forbid common sets of Maxima commands can be used in this field.

`[[BASIC-ALGEBRA]]` which forbids Maxima algebraic commands such as `simplify`, `factor`, `expand`, `solve`, etc.

`[[BASIC-CALCULUS]]` which forbids calculus related commands such as `int` (integration), `diff` (differentiation), `taylor` (Taylor series), etc.

`[[BASIC-MATRIX]]` which forbids matrix related commands such as `transpose`, `invert`, `charpoly` (characteristic polynomial).

> **Example**
>
> In this example, it is helpful to prevent the use of the Maxima command to expand brackets, so enter `expand` here. Alternatively, you may wish to enter `[[BASIC-ALGEBRA]]`.

5.4.9 Allowed words

A comma-separated list of words (text strings) that would normally be forbidden, but which you want to allow students to enter.

Example

This can be left blank for this example.

5.4.10 Forbid float

This determines whether to reject answers as invalid if they contain floating point numbers.

Tip

This is useful if you want to force an answer to be given as fractions, surds etc.

Example

In the example, it seems sensible to forbid floating point numbers, so set this to `Yes`.

5.4.11 Require lowest terms

This determines whether to reject answers as invalid if they contain fractions which are not given in lowest terms.

Example

In the example, there seems no reason to use fractions, in lowest terms or otherwise, so set this to `No`.

5.4.12 Check the type of the response

This specifies whether to reject answers as invalid if they are of a different type to the model answer (as given in Subsection 5.4.2). For example, turning this on would reject an answer which is an equation if an expression was expected.

Example

This is a very useful validity check, so set this to be `Yes`.

5.4.13 Student must verify

STACK has the ability to render the expression entered by the student in traditional mathematical notation so that the student can check their input is as intended before submitting their answer for marking. As a student enters an answer, it is rendered at the position in the question text indicated by the location of the `[[validation:ans1]]` tag. The display is updated as the student enters their answer.

This can be very helpful in aiding a student identify, for example, that they have entered $\frac{1}{x}+1$ rather than $\frac{1}{x+1}$; it might, however, be less useful for simple numerical answers or multiple choice types.

This setting determines whether students must wait for this validation to be displayed before submitting their answer or not.

Example

This is a very useful aid to students, so set this to `Yes`.

5.4.14 Show the validation

This specifies whether to display any input validation errors to the student or not. If *Student must verify* above is set to `Yes`, then the validation must be shown, otherwise the student is in an impossible situation.

The options are:

No
Yes
Yes, with variable list In addition to any input validation, a list of the variables contained in the student answer is displayed. This can help students identify errors in an answer that may not be apparent from a first look at its rendered form.

Example

Since *Student must verify* was enabled above, then this must also be enabled, so select `Yes, with variable list`.

5.4.15 Extra options

Some input types require additional options. If needed, they are given here as a comma-separated list. Whether an input type requires such options is indicated in Appendix D.

Example

`Algebraic input` requires no additional options, so leave this empty.

5.5 Potential response tree: prt1

This section of the form tests the properties of a student's answer, assigns marks and provides answer-specific feedback. It does this through a series of true/false tests forming an acyclic directed graph known as the **Potential response tree** (PRT).

It is possible to have more than one potential response tree. For example, in a question with two answer boxes, you may wish to use one potential response tree for each answer. Alternatively, both answers could be judged in a single tree.

The disconnect between inputs (i.e. boxes into which a student types an answer) and potential response trees (i.e. algorithms which assess students' answers) is a key, and unique, design feature of STACK. The most common situations are

- Each input has a unique potential response tree.
- Every input is used by a single potential response tree.

A potential response tree will only execute when every input upon which it relies is non-empty and valid. Hence, if many inputs are used by a single potential response tree and one is invalid then no partial credit will be awarded. Each input can be used by any number of potential response trees. If an input is not used by any potential response tree then it is essentially a item for information only. STACK includes this possibility. Requiring non-empty and valid inputs drastically increases the reliability of the assessment process.

If required, a second potential response tree called, for example, `prt2`, can be created by including a second feedback tag `[[feedback:prt2]]` in the *Specific feedback* box (Subsection 5.3.2). You will then need to click the

Verify the question text and update the form

button at the end of the *General* section to update the form and create a new section relating to the new potential response tree. There will be one *Potential response tree* section of the question authoring form for each tree. Each of these sections will need to be completed.

Tip

When creating a marking algorithm, it is usually better to test the mathematical properties of the student's answer rather than equivalence to the model answer.

5.5.1 Question value

This is the value (between 0 and 1) returned by the tree for a correct answer. The mark returned is multiplied by the *Default mark* set within the *General* section of the form to give the score for the question.

Example

Set this to be 1.

5.5.2 Auto-simplify

This determines whether to permit STACK to apply Maxima's automatic simplification to the PRT evaluations or not.

Tip

For questions testing a student's ability to correctly gather terms, it is not helpful for Maxima to automatically simplify the student input. In such questions, it is useful to set this to `No`.

Example

For our question, set this to `Yes`.

5.5.3 Feedback variables

This box can contain Maxima commands to perform calculations involving the student's answer. It is similar to the *Question variables*, but also has access to the student's answer. Maxima variables defined in the *Question variables* can also be accessed here.

Tip

For complicated tests on students' answers, it can be useful to conduct the test using Maxima code here and set a Boolean variable on which the PRT decisions can be made.

If *Auto-simplify* has been turned off above, then any simplification required in the code contained here needs to be performed explicitly using the Maxima `ev` command, for example `ev(`*`expression`*`, simp)`. Alternately, the simplification setting can be overruled by manually turning simplification on and off using the Maxima commands `simp:true` and `simp:false` within the*Feedback variables*.

Example

For our example, leave this box empty

5.5.4 Nodes

The PRT is formed from a number of nodes, each of which conducts a true/false test on the student's answers, and modifies the mark and feedback given as a result. New nodes can be created by clicking the Add another node button at the end of this section.

Tips

- It is often helpful to plan the PRT on paper before attempting to implement it online.
- It is useful to create the correct number of nodes needed before starting to complete this section of the form.

A diagrammatic rendering of the current PRT is shown above the section of the form in which the nodes are specified. (This diagram will not be seen when Internet Explorer is used.)

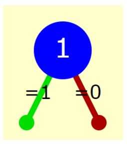


Figure 3 The diagrammatic representation of a simple PRT

Each node consists of three sections, illustrated in Figure 4 and described below.

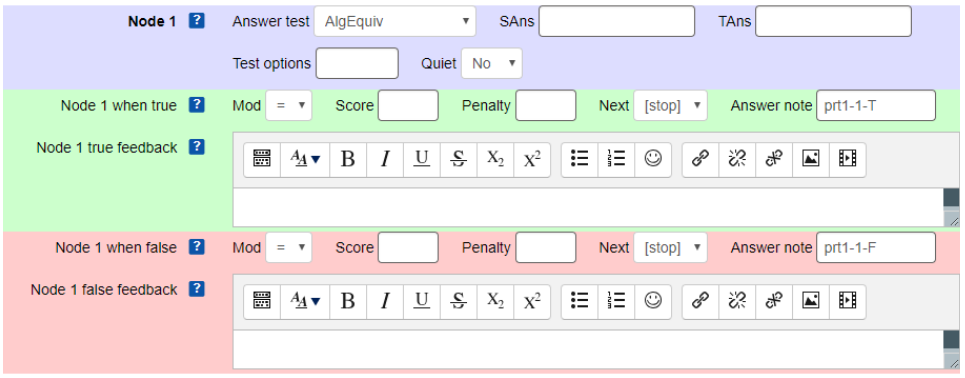

Node 1 ?
Answer test: AlgEquiv
SAns
TAns
Test options
Quiet: No
Node 1 when true ?
Mod: =
Score
Penalty
Next: [stop]
Answer note: prt1-1-T
Node 1 true feedback ?
Node 1 when false ?
Mod: =
Score
Penalty
Next: [stop]
Answer note: prt1-1-F
Node 1 false feedback ?

Figure 4 The section of the form corresponding to a single PRT node

Answer test This section specifies the test to be performed, by selecting from the drop-down menu. The various tests available are outlined in Appendix E. Each test is typically a comparison of the values of two Maxima variables, whose names are entered in the boxes nominally labelled *Sans* (short for student's answer) and *Tans* (teacher's answer). Some tests require additional options (see Appendix E). These are given in the *Test options* box.

Some answer tests give standard feedback to the students, in which case there will be an asymmetry between the arguments *Sans* and *Tans* to a test. If this feedback is not required, then set *Quiet* to `Yes`.

when true This section determines the actions taken when the *Answer test* returns `true`, specifically modifying the question score, the feedback given to the student and what PRT node (if any) is traversed next.

Mod determines how the current score is modified, whether the current score is set to be equal to (`=`) the value given in *Score*, or whether the previous score is increased (`+`) or decreased (`-`) by that value. *Penalty* is any score reduction applied in this case.

Next specifies (via a drop-down list) the next node to traverse, or whether the traversal of the PRT ends here (`[stop]`).

The *Answer note* is a unique string identifying the outcome of this node. By default, it has the name of the PRT, the node number, and whether it is the true or false outcome. This can usually be left unchanged.

The *feedback* is the text given to students if the node test is true. Like the *Question text* and *General feedback*, it can contain the values of Maxima variables, including those defined in the *Question variables*, the *Feedback variables* or the student answer itself.

when false This section of the node is identical to the above, for the case when the node test is `false`.

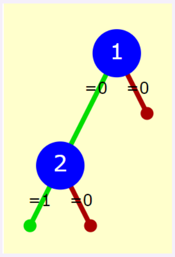
Tip

Care should be taken when using calculations containing the student's answer within the feedback areas. For example, if a student inputs a "placeholder" answer of 999999999.99, exponentiating it will cause Maxima to return a float-point overflow error. Such feedback may need to be "protected" by including another PRT node so that feedback involving student's answer calculation is only given if the answer is within reasonable limits, and more generic feedback given otherwise.

Example

In our example, the required properties of a correct answer are that it is both equivalent to the expression given in the question, and in expanded form.

So we require two nodes: node 1 to test equality and node 2 to test for the form of the answer. Once the PRT form has been completed, the tree of nodes should be as shown below.



First, create the second node, then configure the nodes as follows. Leave the other settings as their defaults.

Node 1

- *Answer test*: Set this to be `AlgEquiv` (algebraic equivalence).
- *SAns*: Enter `ans1`, the students answer.
- *TAns*: Enter `epn`, the variable which stores the expression given in the question.
- *When true*: Set the score equal to 0 (*Mod*:`=`, *Score*:`0`), and the *next* node to be `Node 2`.
- *When false*: Set the score equal to 0 (*Mod*:`=`, *Score*:`0`), and the *next* node to be `[stop]` to end the traversing of the PRT. Set the *feedback* to be "Your answer is not equivalent to the expression given in the question.".

Node 2

- *Answer test*: Set this to be `Expanded` (expanded form).
- *SAns*: Enter `ans1`. We wish to test that the input stored in `ans1` is expanded.
- *TAns*: Enter `ans1` again. The `Expanded` test does not actually use this value, but the form requires an (any non-empty!) entry here.
- *When true*: Set the score equal to 1 (*Mod*:`=`, *Score*:`1`), and the *next* node to be `[stop]` to end the traversing of the PRT.
- *When false*: Set the score equal to 0 (*Mod*:`=`, *Score*:`0`), and the *next* node to be `[stop]` to end the traversing of the PRT. Set the *feedback* to be "Your answer is equivalent to the expression given in the question, but you have not expanded the brackets fully.".

You may wish to use other nodes to give feedback on common errors.

5.6 Options

This section of the form specifies some global properties of the question and also sets the hints given to students after incorrect question attempts when using the *Interactive with multiple tries* question behaviour.

5.6.1 Question-level simplify

This determines whether Maxima's automatic simplification is applied throughout the question, other than in the PRT. This is similar to the corresponding setting for the PRT (Subsection 5.5.2).

Tip

The decision on whether to turn off question-level simplification can be a delicate one. For some discussion of the issues involved, see the STACK documentation at
`[moodle-url]/question/type/stack/doc/`
`doc.php/Authoring/Authoring_quick_start_3.md`

Question-level simplification can be overruled by turning manually turning simplification on and off using the Maxima commands `simp:true` and `simp:false` within the *Question variables* (Subsection 5.1.2) and *Feedback variables* (Subsection 5.5.3) sections.

5.6.2 Assume positive

This specifies whether Maxima should assume all unknowns are positive. In some circumstances, Maxima cannot simplify expressions without knowing their sign.

For example, in the absence of other information Maxima simplifies $\sqrt{x^2}$ to $|x|$. If it knew all the unknowns were positive, the simplification gives x.

5.6.3 Assume real

This is similar to the above and determines whether Maxima should assume all variables are real.

5.6.4 Standard feedback

This is the feedback text to be given when an answer is correct (that is, full marks are awarded), partially correct (greater than 0 but less than full marks) or incorrect (0 marks).

The default text contained in these boxes can be specified in the STACK configuration for a particular installation.

5.6.5 Multiplication sign

This specifies how multiplication signs are displayed throughout the question: dot $(\cdot)$, cross $(\times)$ or nothing.

Tip

A compromise is often needed here. We don't usually write mathematics with every multiplication shown explicitly, yet that is often useful when students are validating their input.

5.6.6 Surd for square root

This specifies how square roots are displayed: as a surd $(\sqrt{\ })$ or a fractional power.

5.6.7 Meaning and display of sqrt(-1)

This specifies what symbol to use for $\sqrt{-1}$.

5.6.8 Inverse trigonometric functions

This specifies the notation to use for inverse trigonometric functions, for example, $\cos^{-1}(x)$, $\mathrm{acos}(x)$ or arccos(x).

5.6.9 Default shape of matrix parenthesis

This allows the question author to specify the shape of brackets used to delineate matrices.

5.6.10 Hints

Hints are text given to students after each incorrect attempt at the question when the *Interactive with multiple tries* question behaviour is used. *Hint 1* is given after the first incorrect attempt, *Hint 2* is given after the second incorrect attempt etc. More hint boxes can be added using the Add another hint button below the last hint.

The maximum number of incorrect attempts permitted is one more than the number of non-empty hint boxes.

Tip

The number of hints used (and hence attempts allowed) should be consistent with the score penalty specified in Subsection 5.3.3 to be applied after each incorrect attempt.

Example

Most of the *Options* settings can be ignored for our example question, but you should enter text in *Hint 1* and *Hint 2* to enable students to have three attempts at the question when it is used in *Interactive with multiple tries* mode.

5.7 Tags

This section of the form allows user-specified **tags** (identifier texts) to be added to a question. Within the *Question bank*, questions can be searched for by tags. Tags can also be used within a quiz, to allow an included question to be randomly selected from all questions with particular tags.

To add a new tag to a question, type its name in the tag box and press `Enter`. Alternatively, an existing tag can be added by selecting from the drop-down menu.

Tags can also be added to a question via the *Question bank*, by clicking the 🏷 icon next to the question name.

Example

For this example, you can ignore this section of the form.

5.8 Created/last saved

This section contains automatically generated meta-data about the question.

5.9 Fix dollars

At the end of the form is an option to enable STACK to automatically convert any LaTeX environments delineated with `$...$` or `$$...$$` to ones delineated with `\(...\)` and `\[...\]` as the question is saved.

6 Previewing the question

Note
The `Preview` link in the question form will show the last *saved* version of the question.

The question can be previewed by either

- in the question form, clicking `Preview` which should be shown next to the Save changes and continue editing button once the question has been successfully saved; or
- from the question bank, clicking the preview icon 🔍 near the question name.

This opens a *Preview question* page which displays an instance of the question.

Below the question is a set of buttons allowing you to start the question again, automatically fill in a correct answer and submit the question as if in a quiz.

Further down the page are various options to change how the question behaves, which replicate many of the options available when forming a quiz from a set of questions.

In particular, within the *Attempt options* section is the option to change the question behaviour used.

7 Question tests and deploying variants

Questions tests and **deployed question variants** improve the quality assurance of STACK questions, both when they are written and in the future. *Question tests* (which are *unit tests* in software engineering terms) ensure that the question is behaving correctly: that is, correct and incorrect answers are being marked as such, with appropriate PRT paths being traversed, and appropriate marks and feedback being given. If your STACK installation is updated with a new version, you can run all the question tests of all of your questions to reassure yourself all is working correctly. (Or, alternatively, to find and resolve any problems!)

Deploying variants of questions allows the set of randomly generated variants to be reviewed by the teacher before being offered to students. This ensures that unforeseen special-cases etc. are not accidentally given to students.

Tip

All questions should have question tests and deployed variants.

The settings for both the question tests and the deployment of variants are contained within the **Question tests & deployed versions** page which can be accessed by the link of that name either below the *Question name* on the question editing form or at the top right-hand corner of the *Preview question* page.

7.1 Question tests

To add a *Question test*, click the Add a test case ... (or the Add another test case ...) button within the *Question tests* section of the *Question tests & deployed versions* page.

A new page is shown summarising one particular variant of your question, together with your *Question variables* code and the particular values taken by the variables for this specific instance.

Towards the bottom of the page is the *Test inputs* section where the question test is specified.

The sample student answer(s) to be tested should be entered in the box(es) labelled to correspond to the variable(s) in which student input is stored, for example *ans1*. Since the (correct) answer to a question probably varies between different question instances, the sample student answer will probably be the name of a Maxima variable defined within *Question variables*. The expression given in the box will not be automatically simplified by Maxima, so if you wish to calculate an expression based on the values of question variables here, you will need to explicitly evaluate it using `ev(`*`expression`*`, simp)`.

The expected outcomes for this answer (the score and penalty given and the answer note of the final PRT node traversed) can be entered at the bottom of the page. Alternatively, you can click the Fill in the rest of the form to make a passing test-case button to complete the form for you, on the assumption the test case passes. Using this button, however, can lead to PRT bugs being undetected and hence it should be used with caution. Check the data entered into the boxes looks sensible and then click Create test case. You should repeat this for each test case you wish to add.

Tip

As a minimum, a test case should be added for a correct answer and an incorrect answer. You may also want to add others, to check common wrong answers are marked correctly, and to test specific paths through the PRT. Good practice would be to ensure that each node in the PRT follows the `true` branch for at least one question test, and similarly for the `false` branch.

Example

For the example question, it would be sensible to add question tests for the following possible values of *ans1*.

`ans` to check the correct answer is marked as such.
`epn` to check the expression given in the question is not marked correct, and that appropriate feedback is given.
`ev(epn+1,simp)` to check that an answer not algebraically equivalent to the question is marked as incorrect, and appropriate feedback is given.

7.2 Deploying variants

Once *Question tests* have been created, the final step in creating a STACK question is to deploy a suitable set of question variants. This ensures that the “random” questions offered to students are chosen from a known set, which can be explicitly checked by the author prior to student use.

To deploy, say, 10 variants, enter `10` in the *Attempt to automatically deploy the following number of variants* box in the *Deployed variants* section at the top of the *Question tests & deployed versions* page, and click $\boxed{\text{Go}}$. The deployment may stop before the requested number of variants has been created if

- there have been 3 attempts to create a new variant that fail to create one different from those already created,
- any question tests fail.

Once variants have been deployed, they are listed (by variant number and *Question note*) at the top of the *Question tests & deployed versions* page. If necessary, variants can be removed (and hence not offered to students) by clicking the cross icon next to the variant number.

Import, export and duplication of questions

When a STACK question is exported from a Moodle course (see Subsection 4.2), the deployed variants and question tests are also exported, and hence remain part of the question when imported to a new Moodle course.

When a question is duplicated, however, deployed versions are *not* copied to the duplicated version although the question tests are.

Appendix A Useful Maxima commands

Maxima is a mature computer algebra system. If you intend to develop a wide range of STACK questions then it is worthwhile investing a little time to get to know Maxima. The following document is a good place to start.

```
http://maxima.sourceforge.net/docs/tutorial/en/minimal-maxima.pdf
```

Help on Maxima commands can be obtained by typing

`?` *command-name*

within a local installation of Maxima, or by using online manuals such as

```
http://maxima.sourceforge.net/docs/manual/maxima.html
```

Complete Maxima commands should end with a semi-colon (`;`).

Mathematical constants

Constant	*Syntax*
e	`%e`
π	`%pi`
i	`%i`

Mathematical operations and functions

	Example
Addition	`2+3`
Subtraction	`2-3`
Multiplication	`2*3`
Division	`2/3`
Brackets	`2*(3+4)`
Powers	`2^3` *or* `2**3`
Square root, $\sqrt{x}$	`sqrt(2)`
Exponential, e^x	`%e^2`, `exp(2)`
Natural logarithm, ln	`log(2)` or, in STACK only, `ln(2)`
Magnitude/modulus, $\lvert x\rvert$	`abs(-2)`, `abs(1+%i)`
Equality (in an equation)	`2*x+1=3`
sin, cos, tan	`sin(%pi)`, `cos(%pi)`, `tan(%pi)`
cosec, sec, cot	`csc(%pi/2)`, `sec(%pi)`, `cot(%pi/4)`
$\sin^{-1}$, $\cos^{-1}$, $\tan^{-1}$	`asin(0)`, `acos(0)`, `atan(0)`
The arguments of trigonometric functions	*are in radians.*
sinh, cosh, tanh	`sinh(0)`, `cosh(0)`, `tanh(0)`
cosech, sech, coth	`csch(1)`, `sech(0)`, `coth(1)`
$\sinh^{-1}$, $\cosh^{-1}$, $\tanh^{-1}$	`asinh(0)`, `acosh(1)`, `atanh(0)`
Factorial, $n!$	`3!` *or* `factorial(3)`
Binomial coefficient, ${}^nC_r$	`binomial(3,2)`
Greatest common divisor (highest common factor)	`gcd(6,3)`
Convert to a decimal number	`float(sqrt(2))`
Round to nearest integer	`round(1.2)`
Round to integer below	`floor(1.2)`
Round to integer above	`ceiling(1.2)`

Note
Maxima by default uses *banker's rounding*; for example `round(2.5)` gives 2. This is often not what you meant!

Variables and functions

Variables and function names can be a combination of letters followed by a combination of numbers. (This is a STACK restriction on the usual Maxima convention.)

	Example
Assign a value to a variable	`a:2`
Define a function	`f(x):=2*x+3`
Define a multi-line function (subroutine)	`f(x,y):=block([`*list of local variables*`],` *command*`,` *command*`,` `return(`*value to be returned*`) )`
Evaluate a function at a value	`f(1)`

Lists

List are given within square brackets (`[,]`), with elements separated by commas. Lists are ordered, and can have repeated elements.

	Example
Assign a list to a variable	`L:[1,1,2,3,5]`
Length of a list	`length(L)`
Element of a list (e.g. 3rd)	`L[3]`
Create a list using a general term	`makelist(2*n,n,1,100)`

Sets

Sets are given within curly brackets (`{,}`), with elements separated by commas. Sets are unordered (so `{1,2}`≡`{2,1}`). When a set is simplified, the elements are sorted and repeated elements removed.

	Example
Assign a set to a variable	`S:{1,3,5}`
Number of (distinct) elements in a set	`length(S)`
Convert a list to a set	`setify(L)`
Convert a set to a list	`listify(S)`

Matrices

	Example
Assign a matrix (e.g. $\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}$) to a variable	`A:matrix([1,2],[3,4])`
Size of a matrix	`matrix_size(A)`
Row of a matrix (e.g. 2nd)	`A[2]`
Element of a matrix	`A[3,4]` *or* `A[3][4]`
Matrix addition	`A+B`
Matrix subtraction	`A-B`
Multiplication by a scalar	`2*A`
Matrix multiplication	`A.B`
Matrix power	`A^^3`
Transpose of a matrix	`transpose(A)`
Determinant of a square matrix	`determinant(A)`
Inverse of a square matrix	`invert(A)` *or* `A^^(-1)`

Algebraic manipulation

	Example
Expand brackets	`expand((x+y)^2)`
Factorise	`factor(x+2*x^2)`
Simplify a rational expression	`fullratsimp((x+2*x^2)/x)`
Numerator	`num(x/(x+2));`
Denominator	`denom(x/(x+2));`
Simplify an expression involving exponentials or logarithms	`radcan(log(2x^3))`
Combine logarithms	`logcontract(log(a)+log(b))`
Expand trigonometric functions of sums and multiples of angles	`trigexpand(sin(A+B))`
Reduce powers of trig functions to functions of multiples of angles	`trigreduce(sin(x)^2)`
Simplify using trig identities	`trigsimp(sin(x)^2+cos(x)^2)`
Simplify a rational trig expression	`trigrat(sin(2*x)/sin(x))`
Left-hand side of an equation	`lhs(2*x+3=x-4)`
Right-hand side of an equation	`rhs(2*x+3=x-4)`
Substitute (e.g. `4` for `x` in `x^2`)	`subst(4,x,x^2);` *or* `x^2,[x=4]`
Solve an equation for a variable	`solve(2*a*x-3=0,x)`
Solve simultaneous equations	`solve([x+y=3,2*x-y=0],[x,y])`

Calculus

	Example
Differentiate	`diff(cos(x^2),x)`
Differentiate twice	`diff(sin(x),x,2)`
Integrate	`integrate(x^3,x)`
Evaluate a definite integral	`integrate(x^2,x,0,1)`

Ordering of expressions

	Example
Order in terms of increasing power eg., $1 + x + x^2$	`powerdisp:true`
Give priority ordering to x eg., $x + y$	`ordergreat(x)`

Simplification

	Example
Turn automatic simplification off	`simp:false`
Turn automatic simplification on	`simp:true`
Force simplification (when turned off)	`ev( expand((x+1)^2), simp)`

Display of variables, functions etc

It is possible to change the way a Maxima variable, function or operator is displayed when formatted using LaTeX. For example, STACK is configured to display `log(x)`, the natural logarithm, as $\ln(x)$ by default. This can be changed to $\log_e(x)$ by including the command

```
texput(log,"\\mathrm{log}_e", prefix);
```

within the *Question variables*.

Note the backslash in the name of the LaTeX command `\mathrm` (which typesets text in a mathematics roman font) needs to be escaped using `\\`. The `prefix` argument indicates the function name prefixes its argument. If the display of a variable name is being set, no third argument is needed. The possible values of this argument are as follows.

`prefix` For a function/operator name that precedes the argument.

`infix` For an operator that appears between two variables.

`postfix` For a function/operator name that follows the argument.

`nary` For an operator that appears between each of two or more variables.

`nofix` For where there is no argument. This is equivalent to omitting the third argument.

`matchfix` For where the function/operator surrounds the argument with matching symbols, such as brackets. In this case the second argument of the `texput` command needs to be a list containing the symbol to be used on the left followed by that to be used on the right. Optionally, a third element of the list gives the symbol with which to separate any multiple arguments of the function/operator.

Appendix B STACK-specific Maxima commands

STACK provides some additional Maxima commands which are fully detailed at

```
[moodle-url]/question/type/stack/doc/doc.php/CAS/Maxima.md
```

Some of the more commonly used commands are indicated below.

Randomisation

	Example
Select a random element a list	`rand([1,2,3])`
Generate a random integer between, for example, -5 and 5 but excluding any of the elements of the list `[-1,0,1]`	`rand_with_prohib(-5,5,[-1,0,1])`

Algebraic

	Example
Complete the square in the variable, for example `x`	`comp_square(x^2+2*x,x)`

Housekeeping

	Example
Make all variables in an expression lower case	`exdowncase(X+Y+z)`

Numerical

The following commands are useful when manipulating and displaying floating point numbers to a specified precision. More details can be found at

```
[moodle-url]/question/type/stack/doc/doc.php/CAS/Numbers.md
```

	Example
Round, for example, 1.234 to 2 decimal places,	`decimalplaces(1.234,2)`
or to 2 significant figures	`significantfigures(1.234,2)`
Display a number to the specified number of decimal places, with trailing zeroes if required,	`dispdp(1.2,2)`
or to a specified number of significant figures	`dispsf(1.2,3)`
Display x in scientific notation, with n decimal places in the mantissa	`scientific_notation(x,n)`

Note
Beware Maxima's use of banker's rounding, which may, for example, round 1.245 to 1.24 to 2 d.p.

Appendix C Basic LaTeX commands

LaTeX commands start with `\`. Items are grouped with `{...}`. Optional arguments are given within square brackets `[...]`. LaTeX syntax should be contained within

`\(...\)` for in-line mathematics
`\[...\]` for displayed mathematics

	Syntax	*Output*
Powers	`x^2`	x^2
	`x^ {2y}`	x^{2y}
Subscripts	`x_n`	x_n
Roots	`\sqrt{2}`	$\sqrt{2}$
	`\sqrt[3]{2}`	$\sqrt[3]{2}$
Fractions	`\frac{x}{2}`	$\frac{x}{2}$
Brackets	`(...)`	$(...)$
	`[...]`	$[...]$
	`\{...\}`	$\{...\}$
... variable size brackets	`\left( \frac{x}{2} \right)`	$\left(\frac{x}{2}\right)$
Relations	`<`	$<$
	`>`	$>$
	`\le`	$\le$
	`\ge`	$\ge$
	`\ne`	$\ne$
	`\approx`	$\approx$
Operators	`\times`	$\times$
	`\div`	$\div$
	`\pm`	$\pm$
	`\sum_{i=1}^n`	$\displaystyle\sum_{i=1}^n$
	`\prod_{i=1}^n`	$\displaystyle\prod_{i=1}^n$
	`\int_0^1`	$\displaystyle\int_0^1$
Functions	`\sin`	$\sin$
	etc.	
Greek letters	`\gamma`, `\Gamma`	γ, Γ
	etc.	

Aligned equations	`\begin{align*}`
	`x &= (1+y)^2\\`
	`&= 1+2y+y^2`
	`\end{align}`
		$\begin{aligned} x &= (1+y)^2 \\ &= 1+2y+y^2 \end{aligned}$

Appendix D STACK answer input types

The available input types are as follows. Full details are available at

`[moodle-url]/question/type/stack/doc/doc.php/Authoring/Inputs.md`

When needed, *Extra options* is a comma-separated list of the possible options listed.

Input type	*Expected input*	*Notes*
`Algebraic input`	An algebraic expression	
`Numerical`	A number (possibly given in terms of standard functions, for example, `sin(pi/4)`)	Trailing zeros in student's answer are preserved. *Extra options* include: `floatnum`, answer must be floating-point `rationalnum`, answer must be a rational `rationalized`, denominators must be surd-free `mindp:n`, must have `n` or more decimal places `maxdp:n`, must have at most `n` decimal places `minsf:n`, must have `n` or more significant figures `maxsf:n`, must have at most `n` signfiicant figures Decimal place and significant figure option cannot be used together.
`Units`	A number with units eg, `12.1m/s^2`	Units are treated as multipilers of the number, so students must either enter `*` between the number and unit, or the question needs to automatically insert it (see Subsection 5.4.5). To allow standard prefixes (`k`,`M` etc.) include `stack_unit_si_declare(true)` in *Question variables* code, and ensure stars are inserted assuming single character variable names. *Extra options* include: `negpow`, display, eg.,`m/s` as `ms`$^{-1}$ `mindp:n`, must have `n` or more decimal places `maxdp:n`, must have at most `n` decimal places `minsf:n`, must have `n` or more significant figures `maxsf:n`, must have at most `n` signfiicant figures Decimal place and significant figure option cannot be used together.
`Matrix`	A matrix	Provides a grid of boxes (with the dimensions taken from the model answer) for students to complete. Input is stored as a Maxima matrix.

Input type	*Expected input*	*Notes*
`True/False`	Provides a true/false drop-down menu	Returns the Maxima boolean value `true` or `false`
`Drop down` (list) `Checkbox` `Radio`	A multiple-choice question of the corresponding style	The model answer must be a list of lists. Each element of the outer list has the form `[value, correct, display]` where `value` is the value stored in the answer variable when this option is selected, `correct` is `true` or `false` depending on whether the option is a correct answer or not, `display` is an optional item giving what to display for each item. In its absence, `value` is displayed.
`Single character`	A single character	
`String`	A Maxima string	
`Text area`	A set of algebraic expressions, one on each line	Input is stored as a list, each element being one line of input. The model answer should also be a list.
`Equivalence reasoning`	A set of algebraic expressions, one on each line	The equivalence of each line and the previous line is checked. Currently under development. See `[moodle-url]/question/type/stack/doc/doc.php/CAS/Equivalence_reasoning.md`
`Notes`	Free text input	The input is not stored, so cannot be marked. The answer variable should not be used in the PRT.

Appendix E STACK answer tests

Answer tests are used in each potential response tree (PRT) node to judge and mark a student's answer. They generally compare the expression or variable given in the PRT node box labeled *Sans* ("student's answer") with that given in the *Tans* ("teacher's answer") box. Here, we denote the contents of these boxes with `sans` and `tans` respectively. Some tests only make use of `sans`, but both boxes still need to be completed. (When only `sans` is needed, 0 could be entered for `tans`.)

Some tests require additional information to be given in the *Test options* box.

Full documentation on the answer tests is available at

> `[moodle-url]/question/type/stack/doc/doc.php/Authoring/Answer_tests.md`

The available answer tests include the following.

Equality

Answer test	*Tests for*	*Notes*
`AlgEquiv`	Algebraic equivalence	Always simplifies the given input. (Can fail in some cases, for example nested surds.)
`SubstEquiv`	Algebraic equivalence, up to substitution	Considers x^2 and a^2, for example, equivalent. Note: if you only wish to consider case equivalence ($x \equiv X$), use `AlgEquiv` with `exdowncase(sans)` and `exdowncase(tans)` in the *Sans* and *Tans* boxes.
`EqualComAss`	Equivalence up to commutativity of addition and associativity of multiplication	Considers $a + b \equiv b + a$, but $x + x \not\equiv 2x$.
`SameType`	Whether `sans` and `tans` have the same type	For example, whether both are algebraic expressions or both are lists etc.
`SysEquiv`	Whether two systems (lists) of multivariable polynomial equations have the same solution set	
`CasEqual`	Whether `sans` and `tans` have the same internal representation in the computer algebra system	

Algebraic form

Note: for tests that consider both form and equivalence, you can test for form alone by testing `sans` against `sans`.

Answer test	*Tests for*	*Notes*
`Expanded`	Whether `sans` is fully expanded	`tans` is not used.
`FacForm`	Whether `sans` is fully factorised (over the rationals) and algebraically equivalent to `tans`	Give the variable the factorisation is done with respect to in *Test options.*
`LowestTerms`	Whether all numerical fractions are in lowest terms, with the denominator free of surds and complex numbers	`tans` is not used.
`SingleFrac`	Whether `sans` is a single fraction and algebraically equivalent to `tans`	The test fails for student input of the form `-(a/b)` which is considered to be the negation of a single fraction, not a single fraction.
`CompletedSquare`	Whether `sans` is in completed square form and algebraically equivalent to `tans`	Give the variable the completing square is done with respect to in *Test options.*
`PartFrac`	Whether `sans` is in partial fraction form and algebraically equivalent to `tans`	Give the variable the partial fraction is done with respect to in `Test options`.

Calculus

These test the equivalence of algebraic expressions, but give automatic feedback appropriate to these common operations.

Answer test	*Tests for*	*Notes*
`Diff`	Whether `sans` and `tans` are equivalent	Give the variable to differentiate with respect to in *Test options.* Feedback is given if it appears the student has integrated.
`Int`	Whether `sans` and `tans` are indefinite integrals of the same expression.	Give the variable integrated with respect to in *Test options.* Alternatively, a list of options can be given, starting with the integration variable and possibly including `NOCONT` which condones any lack of arbitary constant

Numerical

When using these tests it is important that no pre-processing of the student input is undertaken, so `sans` should be the raw variable to which student input is assigned. This is to prevent the removal of trailing zeros from floating-point numbers. Corresponding care should to taken throughout the student feedback areas to ensure the correct number of significant figures are displayed.

As always, small numerical errors may be introduced when processing floating-point numbers.

Answer test	*Tests for*	*Notes*
`NumAbsolute`	$\lvert \texttt{sans} - \texttt{tans} \rvert < \varepsilon$	Give the value of the tolerance, ε, in *Test options*. `sans` and `tans` can be numbers, sets of numbers or lists of numbers.
`NumRelative`	$\left\lvert \frac{\texttt{sans} - \texttt{tans}}{\texttt{tans}} \right\rvert \leq \varepsilon$ for $\texttt{tans} \neq 0$ or, $\lvert \texttt{sans} \rvert = 0$ if $\texttt{tans} = 0$	Give the value of the tolerance, ε, in *Test options*. `sans` and `tans` can be numbers, sets of numbers or lists of numbers.
`Num-GT`	$\texttt{sans} > \texttt{tans}$	
`Num-GTE`	$\texttt{sans} \geq \texttt{tans}$	
`NumDecPlaces`	`sans` is given to d decimal places and $\texttt{sans} = \texttt{tans}$ to d decimal places	Give the number of decimal places d (a positive integer), in *Test options*. Trailing zeros are counted, so $1.2 \neq 1.20$. `sans` and `tans` are rounded to d decimal places before the test is applied.
`NumSigFigs`	`sans` is given to s significant figures and $\texttt{sans} = \texttt{tans}$ to t significant figures.	Give the numbers of significant figures (positive integers) as a list `[s,t]` in *Test options*. If $s = t$, then simply give `s`. Common options to use are `[s,s-1]` which permits an error in the final digit `[s,0]` when only the number of significant figures is checked `[s,-1]` which checks the answer has *at least* `s` significant figures and is correct to the accuracy given. The test only supports $\texttt{sans} < 10^{22}$.
`StrictSigFigs`	The number of significant figures given (strict interpretation)	`tans` is not used. Give the number of significant figures (a positive integer), in *Test options*.

Units

These answer tests are for numerical answers with scientific units and need to be used with the `Units` input type. The teacher's answer needs to be specified with units too, for example `12.1*m/s^2`. Both student's and teacher's answers are converted to base SI units before comparisons are made. The Unit tests are based on the Numerical tests above.

Tests names beginning `Units...` permit conversion of units when comparing to the teacher's answer. Those beginning `UnitsStrict...` require the student's and teacher's answers to be given in exactly the same units.

Answer test	*Tests for*	*Notes*
`UnitsAbsolute`	$\lvert\texttt{sans}-\texttt{tans}\rvert<\varepsilon$ after conversion to base units	Give the value of the tolerance, ε, in *Test options*.
`UnitsRelative`	$\left\lvert\frac{\texttt{sans}-\texttt{tans}}{\texttt{tans}}\right\rvert\leq\varepsilon$ for $\texttt{tans}\neq 0$ or, $\lvert\texttt{sans}\rvert=0$ if $\texttt{tans}=0$ after conversion to base units	Give the value of the tolerance, ε, in *Test options*.
`UnitSigFigs`	`sans` is given to s significant figures and $\texttt{sans}=\texttt{tans}$ to t significant figures, after conversion to base units	Give the numbers of significant figures (positive integers) as a list `[s,t]` in *Test options*. If $s=t$, then simply give `s`. Common option to use are `[s,s-1]` which permits an error in the final digit `[s,0]` when only the number of significant figures is checked `[s,-1]` which checks the answer has *at least* `s` significant figures and is correct to the accuracy given.
`UnitsStrictAbsolute`	$\lvert\texttt{sans}-\texttt{tans}\rvert<\varepsilon$ with `sans` and `tans` in the same units	Give the value of the tolerance, ε, in *Test options*.
`UnitsStrictRelative`	$\left\lvert\frac{\texttt{sans}-\texttt{tans}}{\texttt{tans}}\right\rvert\leq\varepsilon$ for $\texttt{tans}\neq 0$ or, $\lvert\texttt{sans}\rvert=0$ if $\texttt{tans}=0$ with `sans` and `tans` in the same units	Give the value of the tolerance, ε, in *Test options*.

Answer test	*Tests for*	*Notes*
`UnitsStrictSigFigs`	`sans` is given to s significant figures and `sans` $=$ `tans` to t significant figures, with `sans` and `tans` in the same units	Give the numbers of significant figures (positive integers) as a list `[s,t]` in *Test options.* If $s = t$, then simply give `s`. Common option to use are `[s,s-1]` which permits an error in the final digit `[s,0]` when only the number of significant figures is checked `[s,-1]` which checks the answer has *at least* `s` significant figures and is correct to the accuracy given.

Strings

Answer test	*Tests for*	*Notes*
`String`	`sans` and `tans` are identical strings, ignoring any leading or trailing whitespace	
`SloppyString`	`sans` and `tans` are identical strings, ignoring case and any leading or trailing whitespace	

Reasoning by equivalence

These tests are for answers consisting of a set of expressions or equations, one per line, which ought to be equivalent. The lines could represent steps within the rearrangement of an expression or solution of an equation. These tests need to be used with the `Equivalence reasoning` input type.

This is an area currently under development and the type of expressions supported is currently limited. See `[moodle-url]/question/type/stack/doc/doc.php/CAS/Equivalence_reasoning.md` for more details.

Lines of working in both the student's and teacher's answers are stored as lists.

The *Test option* for these tests is a list containing all or some of the following

`hideequiv` Do not show line equivalence at validation.

`comments` Allows students to include comments.

`firstline` Forces the first line of the student's answer to be the first line of the teacher's answer.

`assume_pos` Assumes variables are positive, so $x^2 = 4$ is equivalent to $x = 2$. If used, this also needs setting in the question *Options* (subsection 5.6.2).

`assume_real` Assume working over real numbers. If used, this also needs setting in the question *Options* (subsection 5.6.3).

`firstline` can also be used as a *Syntax hint* (subsection 5.4.6) in which case the particular value of the first line of the teacher's answer is provided as a hint.

Answer test	*Tests for*	*Notes*
`EquivReasoning`	Whether all lines with `sans` are equivalent	`tans` is not used.
`EquivFirst`	Whether all lines with `sans` are equivalent and the first line of the student's and teacher's answers are equivalent, up to commutativity and associativity.	

Index